ANALYSIS OF ANDROID VULNERABILITIES AND MODERN EXPLOITATION TECHNIQUES

Himanshu Shewale[1], Sameer Patil[2], Vaibhav Deshmukh[3] and Pragya Singh[4]
MS (Cyber Laws and Information Security) Division, Indian Institute of Information Technology, Allahabad, India
E-mail: [1]ims2012014@iiita.ac.in, [2]ims2012012@iiita.ac.in, [3]ims2012047@iiita.ac.in, [4]pragyabhardwaj@iiita.ac.in

Abstract

Android is an operating system based on the Linux kernel. It is the most widely used and popular operating system among Smartphones and portable devices. Its programmable and open nature attracts attackers to take undue advantage. Android platform allows developers to freely access and modify source code. But at the same time it increases the security issue. A user is likely to download and install malicious applications written by software hackers. This paper focuses on understanding and analyzing the vulnerabilities present in android platform. In this paper firstly we study the android architecture; analyze the existing threats and security weaknesses. Then we identify various exploit mitigation techniques to mitigate known vulnerabilities. A detailed analysis will help us to identify the existing loopholes and it will give strategic direction to make android operating system more secure.

Keywords:
Android, Vulnerability, Exploit, Malware, Linux Kernel

1. INTRODUCTION

Android is a fast growing and largest installed base of mobile platform which powers millions of mobile devices. Based on the Linux kernel, android operating system is open and flexible enough to run on different mobile devices having varied hardware configuration [6]. This increases the popularity and acceptance of android amongst the users.

Android platform provides developers huge opportunity to develop applications to cater to the needs of its ever increasing user base. The Android platform includes applications, middleware and the operating system [7]. Developers use the AndroidSDK, which consist of various tools and APIs to develop applications using programming language like Java. Android provides an open marketplace wherein developers can sell and distribute applications instantly.

While the openness of android provides a favorable atmosphere for users as well as developers, it also attracts attackers and hackers to take undue advantage [1]. The capability of android to run on different devices and with different versions exposes it to varied security issues. Not all devices can be updated to latest version because of the customization done by different device manufacturer. This result in leaving the old users stay unprotected from latest security issues addressed in the new version [2].

The android marketplace lacks rigorous inspection of the applications being sold and distributed by developers [1]. Applications can be published in the marketplace without any third party's review. It leaves a device running android susceptible to stealing of data that is of corporate or personal use. Smartphones store information like location history, contacts, mails, call register, photos, messages or any other file that is important [2]. Malicious applications can gain access to user's private information stored in the device. A malware can even try to gain root privileges and abuse the normal functioning of the device [3].

2. ANDROID PLATFORM ARCHITECTURE

The Android platform was created by Android Inc. which was later bought by Google and called it the Android Open Source Project. The software can be freely obtained from a central repository and modified in terms of the license. The Android platform is based on the Linux kernel, which is modified to meet special needs of better power management, memory management and runtime environment. Also, as Android is designed to be used on Smartphones and tablets, it has many changes and updates to the Linux kernel in order to support different devices [2]. The additions include subsystem to control memory and processor, libraries to manage file systems designed for memories, process management and device management.

The Android software stack can be subdivided into five layers: the Linux Kernel and lower level tools, System Libraries, the Android Runtime, the Application Framework and Application layer on top of all. Each layer provides different services to the layer just above it.

2.1 LINUX KERNEL

The Linux kernel is the basic layer equivalent to an abstract level between hardware layer and other software layers in the system. The Android OS is built on top of this Linux kernel with some changes in the architecture made by Google [7]. The kernel contains a vast array of device drivers which makes interfacing to peripheral hardware easy. The kernel provides basic system functionality like memory management, process management, security, device management, network group etc. [7].

2.2 LIBRARIES

On top of the Linux kernel is a set of Android's native C/C++ libraries. The libraries are specific for particular hardware. For example, the media framework library guides playback and recording of various pictures, video and audio formats. Some other important core libraries include Surface Manager, SQLite, WebKit and OpenGL.

2.3 ANDROID RUNTIME

Android Runtime includes set of core Java libraries. Application programmers use Java programming language for developing apps. It includes the Dalvik Virtual machine and Core Java libraries.

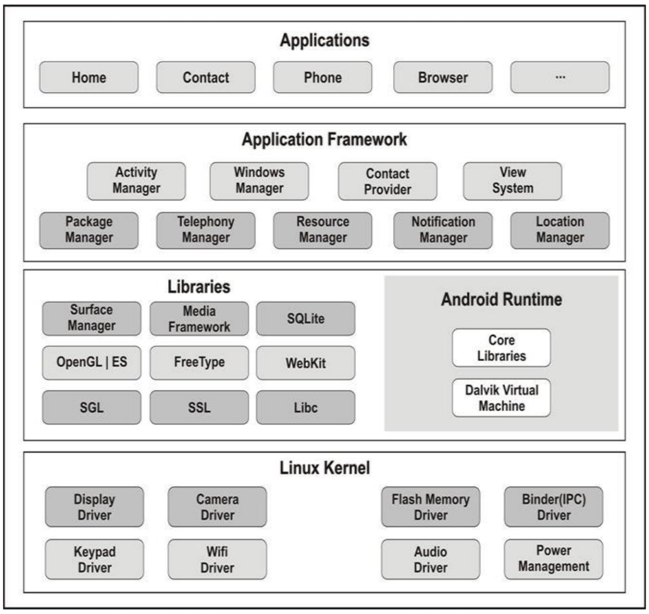

Fig.1. Android Architecture

- **Dalvik Virtual Machine**: A virtual machine runs as if it was an independent device having its own operating system. It permits numerous instances of virtual machine to be executed simultaneously providing isolation, security, memory management and threading support [7]. Each application runs as its own process in a virtual machine such that no other application is dependent on it and in case of application crash, it would not affect any other application running on the device [2]. These features together are called the sandbox.
- **Core Java Libraries**: Most of the functionality defined in the Java SE libraries, including tasks such as string handling, networking and file manipulation is provided by these libraries.

2.4 APPLICATION FRAMEWORK

Application framework manages the functions of the phone such as resource management, voice call management, etc. Applications from the upper layer interact with the Application Framework layer. Some of the important blocks of Application framework are Activity Manager, Content Provider, Telephony Manager, Location Manager and Resource Manager [7].

2.5 APPLICATIONS

Application is the topmost layer of android architecture. An average user mostly interacts with this layer to perform basic functions like making phone call, accessing web browser etc. The Android SDK tools compile the application code and related data or resource file into an android package with .apk suffix [1]. All the contents of an .apk file comprise one application that can be installed on android device. The Android platform has some preinstalled applications by default for the browser, dialer, home, connection manager, etc. Developers are free to use their innovation to build new application according to user's need.

3. ANDROID VULNERABILITIES

We classified the vulnerabilities found in android according to various layers of the android architecture from which they originated. The categories are: Linux Kernel Layer, Libraries Layer, Application Framework Layer, Applications Layer and External Drivers. The purpose of this classification is to identify the weak areas of android implementation. From the sample space of 30 exploits with CVSS scores ranging from 2.6 to 10, the Application Framework Layer was found to have the most number of exploits.

3.1 APPLICATION FRAMEWORK LAYER

The vulnerabilities in Application Framework Layer caused DoS, privilege escalation, code execution and unauthenticated access. For example, a recent vulnerability allows a malicious app to bypass intended access restrictions and remove the device locks activated by any user. Similarly, a Bluetooth service flaw compromised the user's contact data which is considered as sensitive data. An old exploit allowed an app to bypass several permissions and allow the attacker to access the camera and microphone without making permission requests. Other vulnerabilities found are CVE-2011-3975, CVE-2011-0680, CVE-2011-4804, CVE-2009-2999, CVE-2009-2656 and CVE-2009-1754.

3.2 APPLICATION LAYER

The Application Layer exploits occurred mostly through browsers which allowed attackers to execute arbitrary code and provide unauthenticated access to some protected resources. It also includes the vulnerability of the Picasa app where username and passwords were sent in clear text when transmitting the authToken obtained after ClientLogin [8]. As a result, personal pictures and gallery could be accessed by anyone who sniffs the authToken. This famous exploit had the highest CVSS score of 10 as it completely compromised the CIA parameters. A bug in the Android browser allowed man in the middle attack and monitoring of user activities on the browser. It could not restrict modifications to HTTPS session cookies allowing user to inject arbitrary cookies for the sessions. Application layer includes vulnerabilities like CVE-2012-6301, CVE-2012-3979, CVE-2011-2357, CVE-2011-2344 and CVE-2008-7298.

3.3 EXTERNAL DRIVERS

The flaws in the implementation of external drivers like Qualcomm and PowerVR have also caused vulnerabilities. Some specially crafted arguments for a local kgcl_ioctl call caused DoS in Qualcomm graphics kernel driver which can further execute arbitrary code for the attacker in the kernel context. In this, using an application, an attacker could dereference several untrusted pointers from the user space and perform further computations without verification. The Levitator exploit targeted the PowerVR graphics card driver which led to privilege escalation through kernel memory corruption caused in /dev/pvrsrvkm device [22]. Some other vulnerabilities in this category are CVE-2013-3666, CVE-2012-4222, CVE-2012-4221, CVE-2012-4220, CVE-2011-1352 and CVE-2011-1350.

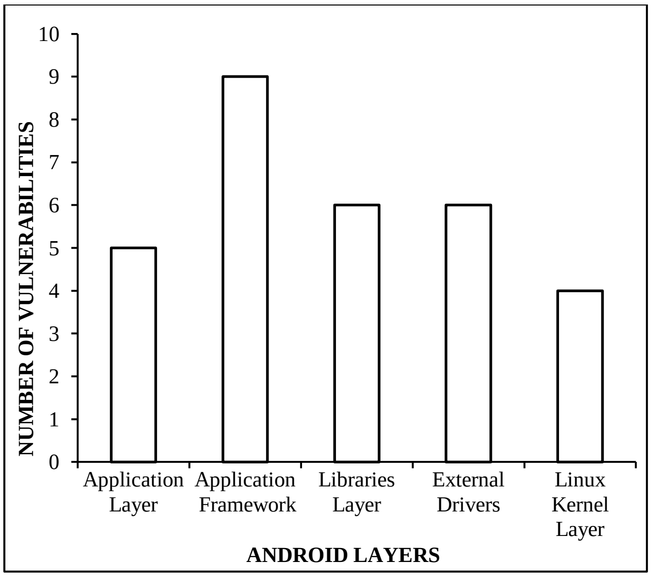

Fig.2. Total vulnerabilities at each layer

3.4 LIBRARIES LAYER

The vulnerabilities in Libraries layer with very high CVSS scores had a huge impact. For example, the ZergRush exploit found in 2011 performed DOS and its exploit code is also available at [18]. ZergRush exploit performed stack buffer overflow, and followed by code execution, by passing wrong number of arguments to a particular API. The GingerBreak exploit CVE-2011-1823 was even used by some well-known malwares to get the root of devices. It took advantage of the fact that the vold volume manager daemon trusted messages that were received from a PF_NETLINK socket, which allowed gain root privileges by bypassing a signed integer check with negative index [11]. The most recent CVE-2013-4787 allowed any person to modify a developer-signed APK file and upload malicious content in it without modifying the signature. This vulnerability, famously known as the “Master Key” vulnerability, can allow attackers to execute arbitrary code on the victim machine. Other vulnerabilities falling in this category are CVE-2011-0419, CVE-2010-1807 and CVE-2009-3698.

3.5 LINUX KERNEL LAYER

As we know that the Android kernel is derived from the Linux kernel itself, choosing the most secure kernel has always been neglected. The wrong permission sets owned by the zygote socket allowed any application to send any number of fork requests without verifying its identity. It resulted in DOS in fork requests from legitimate processes. The KillingInTheNameOf exploit by Sebastian Krahmer of 743C team targeted vulnerability in ashmem that allowed any user to remap shared memory region belonging to init with PROT_READ and PROT_WRITE permissions [11]. This vulnerability was also demonstrated by the psneuter exploit.

4. ANDROID SECURITY USING EXPLOIT MITIGATION TECHNIQUES

The main target for implementing Android security is to protect the user data, system resources, and providing application isolation. For this, Android has timely updated its security controls with each patch and every version it has released. The earlier versions of Android had very little or no security features to protect against advanced attacks because the development was still on and also very few people had android devices.

4.1 ANDROID CUPCAKE

Android 1.5 CupCake had propolice and safe_iop as two security features against buffer overflows [9]. ProPolice prevents stack buffer overruns and safe_iop stops integer overflows. But the two were ineffective as ASLR was still not present.

4.2 ANDROID GINGERBREAD

In Android 2.3 Gingerbread, hardware based No eXecute (NX) was enforced. So even if some application is able to perform buffer overflow and put the exploit code on the stack or heap, the exploit would not execute because of this protection. It also added the format string vulnerability protection as users were able to input specially crafted strings which the application evaluated as a command allowing code execution, reading the stack or cause segmentation faults in the running application. Apart from this, mmap_min_addr is introduced at a basic level to check NULL pointer dereference bugs in kernel space. It protects the system against crashes caused due to triggering of one of such NULL pointer defects.

4.3 ANDROID ICE CREAM SANDWICH

Android 4.0 Ice Cream Sandwich became the first version to implement ASLR. Address Space Layout Randomization (ASLR) randomizes memory addresses of stack, heap, etc each time the memory allocation is done for a process/module. But the first implementation did not live up to the expectations because of lack of randomization of the executable and linker memory regions. Hence attacks using Return Oriented Programming (ROP) were still possible.

4.4 ANDROID JELLY BEANS

With the enforcement of a few more exploit mitigation techniques in Jelly Beans 4.1, the hard work of vulnerability hunters even got harder. The Position Independent Executables support technique allows binaries to be compiled/linked with the PIE flag to ensure the executable mapping will be randomized when executed. Also, kernel.randomize_va_space is being set to 2 to enable heap randomization. In the same way, lib/mmap and linker are also randomized. The GingerBreak exploit as mentioned earlier used a Global Offset Table (GOT) overwrite to perform code execution in vold daemon. To put a check on the exploit, ELF hardening is done where each binary is compiled with the RELRO and BIND_NOW flags. Relocation Read-Only (RELRO) flag tells the linker to make the relocation segments that are used to resolve dynamically loaded functions read-only [8]. With BIND_NOW flag, programs can resolve all dynamic links at start-up itself so that the GOT can be made read only when combined with RELRO.

Jelly Beans also added dmesg_restrict and kptr_restrict protections to avoid leaking of kernel addresses. Dmesg_restrict protection [22] restricts unprivileged access to kernel syslog which prevents leaking of kernel information into user space. Kptr_restrict protection [23] restricts exposing some kernel pointers through various interfaces in /proc.

The next version 4.2 of Jelly Beans brought some more security enhancements. Installd daemon hardening is done to reduce potential attack surface through root privilege escalation. All system libraries and applications during compile time are checked with FORTIFY_SOURCE that detects and prevents a subset of buffer overflows before they could do any damage. Further, init script hardening is also introduced by applying O_NOFOLLOW semantics to prevent symlink related attacks.

Apart from the above mentioned memory protection schemes, vulnerability fixes in the open source libraries like WebKit, libpng, OpenSSL and LibXML are also included. Users can also verify the apps before installing them to check if it is malicious or not.

The most significant security update came in Jelly Beans 4.3 when SELinux was reinforced for Android Sandbox. SELinux [15] implements Mandatory Access Control (MAC) in the Linux Kernel which added more robustness to the present security model, but taking care that it is still compatible with existing applications. Setuid and setguid features were removed so that no low privileged user can execute a file/module with high privileges. Had this been implemented in the earlier versions of Android, exploits like RageAgainstTheCage and ZimperLich would not have occurred.

4.5 ANDROID KITKAT

At present, the latest Android 4.4 KitKat is launched with significant enhancements to the existing security enforcements. KitKat runs SELinux in enforcing mode restricting activities of all modules and applications in accordance withSELinux policies. The AndroidKeyStore is upgraded to include the advanced ECDSA and DSA algorithms [9]. KitKat has also introduced an experimental security feature, dm-verity, to protect against advanced malware/rootkits which stay persistent in a system. It is basically a kernel level protection which can detect modifications on the file system by maintaining a cryptographic tree.

The impact of these mitigation techniques is huge and statistics proves this as well. There were 13 listed vulnerabilities in 2011 when the latest version was GingerBread 2.3. After Ice Cream Sandwich was released, not more than 5 vulnerabilities with high CVSS scores have been reported each year. ASLR protection feature in ICS 4.0 has significantly improved the level of security in Android. This mainly strengthened the Application Framework Layer as the vulnerabilities originating from this layer have reduced to almost nil and it has become very difficult to perform buffer overflows, DoS attacks and code execution.

The library layer API security level has increased because of security features like ELF hardening, FORTIFY_SOURCE and installd hardening. Only a few vulnerabilities originating from libraries layer have been discovered after the launch of ICS.

With the implementation of SELinux in Android, MAC was enforced as a security mechanism instead of DAC. It indeed was a difficult challenge to implement SELinux because of the discretionary nature of the Android APIs. The system-wide policies implemented through the MAC are able to put a check on many exploits like GingerBreak, Exploid, RageAgainstTheCage, Zimperlich, KillingInTheNameOf, Levitator, and Psneuter [15]. SELinux has put a check on the privilege escalation vulnerabilities, as it only gives the permissions which are mandatory and avoid giving discretionary powers at runtime.

5. ANDROID MALWARES OVERVIEW

Android Malwares have been a major security concern for all the users of the OS. Malwares can hide in the phone memory without the user’s knowledge and spy on all his browser and system activities. Some malwares even sent premium SMS messages without user permission to communicate with C&C server. Malware infected APK files were initially distributed by third-party Chinese Android app market, but Android.DroidDream was the first malware to be distributed through the official Android Market. Users must make sure to check what permissions the app wants before installing it, for example, a system cleaner tool won’t require permissions for GPS service. Other types of malwares use exploits like RageAgainstTheCage and KillingInTheNameOf to perform privileged actions [20]. More advanced malwares like DroidKungFu, Geinimi, NickiBot and FakePlayer have compromised user privacy.

6. CONCLUSION

After studying in detail the various Android vulnerabilities, it’s clear how dangerous its impact can be. To tackle the number of increasing vulnerabilities, Android must timely introduce new security enforcement and exploit mitigation techniques. The kernel of Linux OS itself is so vulnerable that every week new exploit is discovered. The vulnerability fixes released for these should be patched in Android's Linux Kernel as well to avoid replicating the same vulnerabilities again. To stay away from malwares, users need to be aware about the importance of looking over the permissions granted to an app during installation time and to download apps from the official Google play store. In the coming years, we see Android to be a very secure OS, which the users can trust enough to do even their banking transactions from smart phones.

REFERENCES

[1] Han Bing, “Analysis and Research of System Security Based on Android”, *Proceedings of Fifth International Conference on Intelligent Computation Technology and Automation*, pp: 581-584, 2012.

[2] Ruben Jonathan Garcia Vargas, Ramon Galeana Huerta, Eleazar Aguirre Anaya and Alba Felix Moreno Hernandez, “Security Controls for Android”, *Proceedings of Fourth International Conference on Computational Aspects of Social Networks*, pp: 212-216, 2012.

[3] Takamasa Isohara, Keisuke Takemori and Ayumu Kubota, “Kernel-based Behavior Analysis for Android Malware

Detection", *Proceedings of Seventh International Conference on Computational Intelligence and Security*, pp: 1011-1015, 2011.

[4] Khodor Hamandi, Ali Chehab, Imad H. Elhajj and Ayman Kayssi, "Android SMS Malware: Vulnerability and Mitigation", *Proceedings of 27th International Conference on Advanced Information Networking and Applications Workshops*, pp. 1004-1009, 2013.

[5] Xiali Hei, Xiaojiang Du and Shan Lin, "Two vulnerabilities in Android OS kernel", *Proceedings of IEEE International Conference on Communications*, pp. 6123-6127, 2013.

[6] Android the world's most popular mobile OS, Available at: http://www.android.com/meet-android/

[7] Android Architecture, Available at: http://www.android-app-market.com/android-architecture.html

[8] Android Vulnerabilities list, Available at: http://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html

[9] Android Security Overview, Available at: http://source.android.com/devices/tech/security

[10] Exploit Mitigataion Techniques in Android Jelly Beans 4.1, Available at: https://www.duosecurity.com/blog/exploit-mitigations-in-android-jelly-bean-4-1

[11] Massimiliano Oldani, "Android Attacks", Available at: www.immunityinc.com/infiltrate/archives/Android_Attacks.pdf

[12] Timothy Strazzere and Timothy Wyatt,"Geinimi Trojan Technical Teardown", Available at: https://blog.lookout.com/_media/Geinimi_Trojan_Teardown.pdf

[13] Yajin Zhou and Xuxian Jiang, "Dissecting Android Malware: Characterization and Evolution", Available at: www.csc.ncsu.edu/ faculty/jiang/pubs/OAKLAND12.pdf

[14] SANS, "Dissecting Andro Malware", Available at: https://www.sans.org/ reading-room/whitepapers/malicious/dissecting-andro-malware_33754

[15] Stephen Smalley, "The Case for SE Android", Available at:www.selinuxproject.org/~jmorris/lss2011_slides/caseforseandroid.pdf

[16] Stephen Smalley,"Implementing SELinux as a Linux Security Module", Available at: www.nsa.gov/research/_files/ publications/ implementing_selinux.pdf

[17] Tsukasa Oi, "Yet Another Android Rootkit", Available at: https://media.blackhat.com/bh-ad-11/Oi/bh-ad-11-Oi-Android_Rootkit-WP.pdf

[18] Revolutionary dev team, zergRush.c, Available at: https://github.com/ revolutionary/zergRush/blob/master/zergRush.c

[19] Jon Oberheide, cve-2009-1185.c, Available at: https://jon.oberheide.org/files/cve-2009-1185.c

[20] Current Android Malwares, Available at: http://forensics.spreitzenbarth.de/android-malware

[21] Jon Larimer and Jon Oberheide,"Levitator.c Android < 2.3.6 PowerVR SGX Privilege Escalation Exploit", Available at: https://jon.oberheide.org/files/levitator.c

[22] Enabling the kernel's DMESG_RESTRICT feature, Available at: https://lists.ubuntu.com/archives/ubuntu-devel/2011-May/033240.html

[23] kptr_restrict for hiding kernel pointers, Available at:http://lwn.net/Articles/420403/