IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 11, Issue 5 (May. - Jun. 2013), PP 25-29
www.iosrjournals.org

Android open-source operating System for mobile devices.

Navnath S.Bagal, Prof.N.D.Kale
1(PG Student), Computer Engineering, Padmabhooshan Vasantdada Patil Institute Of Technology Pune Maharashtra, India;
2(Associate Professor), Computer Engineering, Padmabhooshan Vasantdada Patil Institute Of Technology Pune Maharashtra, India;

Abstract: *The number of Android based smart phones is growing rapidly. They are increasingly used for security critical private and business applications, such as online banking or to access corporate networks. This makes them a very valuable target for an adversary. Up to date, significant or large scale attacks have failed, but attacks are becoming more sophisticated and successful. Thus, security is of paramount importance for both private and corporate users. In this paper, we give an overview of the current state of the art of Android security and present our extensible automated exploit execution framework. First, we provide a summary of the Android platform, current attack techniques, and publicly known exploits. Then, we introduce our extensible exploit execution framework which is capable of performing automated vulnerability tests of Android smart phones. It incorporates currently known exploits, but can be easily extended to integrate future exploits. Finally, we discuss how malware can propagate to Android smart phones today and in the future, and which possible threats arise. For example, Device to device infections are possible if physical access is given.*

I. Introduction

During recent years, the share of smart phones in overall handheld mobile communication device sales has drastically increased. Among them, the Android operating system by the Open Handset Alliance, prominently led by Google Inc., is market dominating. In Q3 2011, 52.5% of all devices sold were Android devices, followed by Symbian (16.9%) and Apple's iOS (15.0%), according to Gartner analysis with the widespread use of smart phones both in private and work related areas, securing these devices has become of paramount importance. Owners use their smart phones to perform tasks ranging from everyday communication with friends and family to the management of banking accounts and accessing sensitive work related data. These factors, combined with limitations in administrative device control through owners and security critical applications like the Mobile TAN for banking transactions, make Android based

Smart phones a very attractive target for attackers and malware authors of any kind and motivation. Up until recently, the Android Operating System's security model has succeeded in preventing any significant attacks by malware. This can be attributed to a lack of attack vectors which could be used for self spreading infections and low sophistication of malicious applications.

However, emerging malware deploys advanced attacks on operating system components to assume full device control. We developed an extensible exploit execution framework to test existing and future exploits in a controlled environment. This framework can also serve to analyze exploitability of devices with specific test sets and payloads. Additionally, common malware behavior is emulated, such as dynamic configuration and exploit download from a remote web server. This paper gives a short, yet comprehensive overview of the major Android security mechanisms. It also discusses possibilities to successfully infiltrate Android based Smart phones through recent attack means. Both pre infection (propagation) and post infection (threat) scenarios will be illuminated. We will also take a look at the implications by current developments for future propagation mechanisms.

II. Android and Android Security

Mobile operating systems preinstalled on all currently sold smart phones need to meet different criteria than desktop and server operating systems, both in functionality and security. Mobile platforms often contain strongly interconnected, small and less well controlled applications from various single developers, whereas desktop and server platforms obtain largely independent software from trusted sources. Also, users typically have full access to administrative functions on non mobile platforms. Mobile platforms, however, restrict administrative control through users. As a consequence, different approaches are deployed by the Android platform to maintain security. This chapter briefly introduces the Android platform and its major security measures, also giving an overview on the measures' limits, weaknesses and even exploitability. Discussion will be limited to those functions related to threats elaborated on in this paper.

2.1 Android Platform

The Android operating system is illustrated in Figure 2.1. Apps for Android are developed in Java and executed in a virtual machine, called Dalvik VM. They are supported by the application framework, which provides frequently used functionality through a unified interface. Various libraries enable apps to implement graphics, encrypted communication or databases easily. The Standard Library ("bionic") is a BSD derived libc for embedded devices. The respective Android releases' kernels are stripped down from Linux 2.6 versions. Basic services such as memory, process and user management are all provided by the Linux kernel in a mostly unmodified form. However, for several Android versions, the deployed kernel's version was already out of date at the time of release. This has lead to a strong increase in vulnerability, as exploits were long publicly available before the respective Android version's release. Detailed information on all security features can be retrieved from the Android Security Overview on the official Android Open Source Project website.

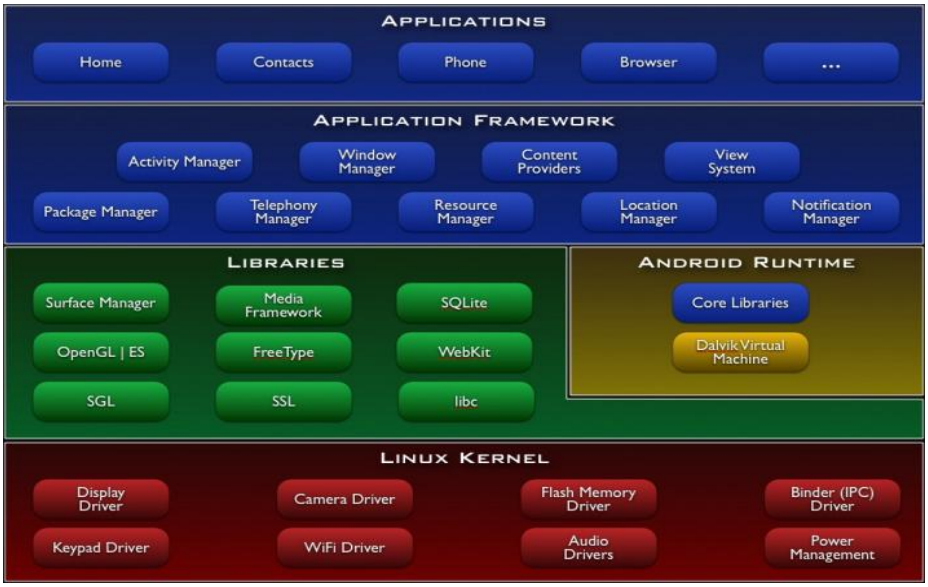

Figure 2.1: Android System Architecture

2.2 File System and User/Group Permissions

As in any Unix/Linux like operating system, basic access control is implemented through a three class permission model. It distinguishes between the *owner* ofm a file system resource, the owner's *group* and *others*. For each of these three entities, distinct permissions can be set to read, write or execute. This system provides a means of controlling access to files and resources. For example, only a file's owner may write (alter) a document, while members of the owner's group may read it and others may not even view it at all. In traditional desktop and server environments, many processes often share the same group or even user ID (namely the user ID of the user who started a program). As a result, they are granted access to all files belonging to the other programs started by that same ID. In traditional environments with largely trusted software sources this may suffice, though Mandatory Access Control approaches trying to establish a more fine grained model do exist, such as App Armor and SELinux.

However, for mobile operating systems this is not sufficient. Finer permission distinction is needed, as an open app market is not a strongly monitored and trustworthy software source. With the traditional approach, any app executed

Under the device owner's user ID would be able to access any other app's data. Hence, the Android kernel assigns each app its own user ID on installation. This ensures that an app can only access its own files, the temporary directory and nothing else – system resources are available through API calls. This establishes permission based
File system sandbox.

2.3 Android API Permission Model and Manifest File

On installation, the user is presented with a dialog listing all permissions requested by the app to be installed. These permission requests are defined in the Manifest File AndroidManifest.xml, which is obligatory for shipping with every Android app. However, this system has a few flaws:

• ***All or none policy*:**

A user cannot decide to grant single permissions, while denying others. Many users, although an app might request a suspicious permission among much seemingly legitimate permission, will still confirm the installation.

• Often, users cannot judge the appropriateness of permissions for the app in question. In some cases it may be obvious, for example when a game app requests the privilege to reboot the smart phone or to send text messages. In many cases, however, users will simply be incapable of assessing Permission appropriateness.

• Circumvention: Functionality, which is supposed to be executable only given the appropriate permissions, can still be accessed with fewer permissions or even with none at all. This point will be explained in detail in the chapter on attack vectors.

2.4 Apps and Native Code

Android apps are developed in Java and executed inside the Dalvik Virtual Machine, a register based Java Virtual Machine. Each app receives its own Dalvik VM instance and user ID, hence separating process memory and files through means of the Linux kernel. Apps authored by the same developer may share the same user ID, if chosen by the developer. System resources may only be accessed by apps through API methods provided by the Dalvik VM. These limitations make attacks on the underlying operating system from Android apps nearly impossible. However, native ARM architecture code can be compiled for Android devices; both the Linux kernel and essential libraries are implemented in C fig.2.1). Native code may even be called from within Android apps, be it for user desired or malicious usage. As native code is executed outside the Dalvik VM, it is not limited to API methods, thus receiving more direct and uncontrolled access to the operating system core. Fraunhofer AISEC

III. Android system services

Android provides the services expected in a modern operating system such as virtual memory,multiprogramming, and threads, all on a mobile platform. Many of Android's services are a result of including the Linux kernel. However the Android team has added the telephony stack in return.

- ***The Linux CPU scheduling algorithm***

Linux employs a number of different methods for scheduling its processes and its algorithm is very nontraditional in the big picture. There are three scheduling schemes for Linux. Each process is assigned a scheduling scheme depending on the type of task it presents. Real time tasks will often run in the SCHED FIFO or SCHED RR scheme. All other tasks will run in SCHED NORMAL [?]. SCHED NORMAL tasks are handled by a special algorithm and are preempted by SCHED RR and SCHED FIFO tasks. In a few words the algorithm behaves like a hybrid priority queue that rewards process which are not CPU-greedy. CPU time is divided up into epochs, which are equal slices of time in which the processes can run and use up some of their allotted time, or times lice. At the end of every epoch, each process' remaining times lice is halved. More time is added proportional to the processes' nice value, a value specified by the user or by the system's default nice value. The interesting rollover time has a rewarding effect to I/O bound processes. Since I/O bound processes will likely spend a lot of time waiting, the scheduling algorithm rewards it by letting it keep some of its timeslice. This way processes which are likely being used by the user are very responsive and will often preempt more non-interactive, CPU-bound processes like batch commands. SCHED RR tasks will preempt SCHED NORMAL tasks and are preempted by SCHED FIFO tasks. These tasks are scheduled according to round-robin rules within their own priority bracket. SCHED FIFO tasks behave quite like the name would suggest. Tasks in this queue are scheduled by priority and arrival time, will preempt any other processes trying to run, and will execute for as long as they please.

- ***Android and file systems***

Android makes use of a multiplicity of file system types, namely due to the expectation of external memory with unsure file system types. As a result Android relies on the standard Linux package to provide for file systems such as ext2 and ext3, vfat, and ntfs. Android itself uses the yaffs2 filesystem, which is not a part of the standard Linux kernel, as its primary file system. YAFFS is a file system optimized for NAND and NOR flash memory. At the time when it was developed, file systems did exist for flash memory but most of them catered toward chips which were small enough to use small block sizes. This was unsuitable for large NAND flash chips. YAFFS attempts to solve this by abstracting storage to "chunks" which scale according to the page size. To scale the method up for considerably large NAND devices, YAFFS has a tweak in the way that it addresses pages. For instance, we might use a page address size of 216 and we may have 218 chunks to address. We do not have sufficient capabilities to address pages individually, but we can address groups of four pages and search for the desired page from there. The useful

Thing about this is that now we have the option of neglecting to use RAM at all to augment our file system. We can pretty easily used some sort of indexed referencing scheme to build a file from a string of chunk IDs, and any scanning for particular pages within that chunk will be a limited set of a size equal to the number of chunks divided by the address size. (218 ÷ 216 = 4). This linear probing may seem like a bad idea. In practice, the inefficiency is too small to notice on such a small set of chunks [?]. YAFFS is preferable as a file system in Android since it optimizes the use of NAND devices as storage and also has great efficiency in memory usage.

- ***The radio interface layer***

Consistent with the rest of the design of Android, the Android team has created a way to abstract a phone call. Since the way that handset manufacturers implement the radio device on cellular devices will inevitably vary, Android has to remain ignorant to the way software interacts with hardware to place the call. The Android team solves this problem by creating the Radio Interface Layer Daemon (RILD), which is the way that the applications framework interfaces with the shared libraries. The RILD's main function is to provide event-driven middle ground between the applications framework and the radio drivers. The RILD is part of the Radio Interface Layer (RIL), which consists of two major parts: the Android RIL and the Vendor RIL. The Android RIL includes the applications framework which makes the request to the RILD to place a phone call, while the Vendor RIL is the part that is up to the vendor to implement [?]. The Vendor RIL includes the driver that the vendor ships for the hardware implementation of the radio antenna. This model allows any hardware to be implemented for placing phone calls so long as the vendor writes drivers which follow the model.

IV. Similarities to iPhone OS

In many ways, it seems as if Android was designed to compete directly with the iPhone OS. TheAndroid team saw that iPhone OS presents an intimidating level of excellence in user interfacing that Android would inevitably have to compete with.

- ***The touch screen***

Android has the full capabilities of interfacing with a touch screen-enabled device, and Android is multi touch-capable[?] but is unimplemented. Many believe this is for Google to remain in good stead with Apple. In any case, the Android implementation on the HTC Dream does show some similarity to the iPhone in the touch-gesture design, such as flicking your finger across the screen to flip to the next page of items. A notable gesture that Android lacks compared to iPhone OS is the zooming gesture, where two fingers are touched on the screen and moved away or toward each other to zoom in and out of the image. In the iPhone implementation of Google Maps, this gesture is available, but in the Android version,"zoom in" and "zoom out" buttons take its place.

- ***Modular application design***

The iPhone is heralded for its multitude of stand-alone "apps" that are available. Android seems to have mirrored this model exactly. However Android differs in its underlying design for Android offers third-party applications unbridled access to the applications framework layer. Modular applications tend to ease the process of understanding to the user and help the user visualize the application as a single, contained unit rather than a string of dependent software.

- ***The Android Market vs. The App Store***

Similarly both operating systems offer a way for developers to publish their applications either for free or for a price. Also users can visit and download applications from the store. Many of the popular apps on the iPhone store are beginning to pop up on the Android Market as well.

V. Pushing for standardization

Android was designed with openness in mind. It would become the first widespread general purpose open operating system designed specifically for the mobile device. In the spirit of keeping the entire project open, the Android team has a few design goals to help support this ideal.

- ***Design goals***

The Android team often lists these four main points[?] as the main ideas of Android: *Open:* Android tries to be especially developer-friendly, thereby making the device completely open for their applications to utilize. As well, Android is open-source and open to implement on most any hardware. *All applications are created equal:* Android's core applications and third-party applications are on equal playing ground. They share the same API, have the same levels of access, and can do anything that the other does. Any core application can be replaced by any third-party application. Breaking *down applications boundaries:* Android does not try to

hide any functionality from the developer. The developer should easily be able to tap into the telephony system, GPS system, or essentially anything that has been implemented in the applications framework. *Fast and easy application development:* The applications framework makes it very easy to do
some fairly sophisticated tricks like reporting the phone's current location and much more.

This speeds up the development process for the developer and makes it easier to focus on design rather than implementation details.

- ***B. Target hardware***

Android makes no attempt at being specific to any one hardware and ships with drivers for many of the common types of hardware seen on the mobile phone. Android is not specific to any one type of mobile device but it is specific to the mobile device in general. However it is most likely that the next line of Android phones will be produced by members of the Open Handset Alliance and that Android will be well-suited to run on these platforms. The operating system itself does not have any preferences.

VI. An open business model

Android is fronted by a coalition of 47 technology and mobile companies, the Open Handset Alliance, many of them with extremely high revenue as it is. Money is not the primary interest In the development of Android as it is free and open-source. However, Android presents a mutual interest between all these companies to create a mobile operating system that is extremely viable and widespread so that it becomes easy to concentrate a business model around some sort of standard. Since Windows, Mac OSX, and Linux are the biggest operating systems in the home computer and server market, they have well-cultivated developer bases. The Android team hopes Android will take off in the same way. It would be beneficial for a company to focus on developing software for one operating system that nearly everyone uses rather than several operating systems which have equal parts in user base. There is room for profit in the Android Market. Currently, the Android Market charges 25 USD to be a registered developer. As well, a developer can purchase and Android Developer's Phone for almost 400 USD. Royalties could potentially be charged for non-free applications on the Android Market as well.

References

[1] Ben Elgin, "Google buys android for its mobile arsenal," http://www.businessweek.com/ technology/content/aug2005/tc20050817_0949_tc024.htm, August 2005.

[2] John Cox, "Why google's gphone won't kill apple's iphone," http://www.networkworld.com/ news/2007/100807-google-gphone-iphone.html, October 2007

[3] Mike Cleron, "Androidology: Architecture overview," http://www.youtube.com/watch?v=Mm6Ju0xhUW8, November 2007.

[4] Josh Aas, "Understanding the linux 2.6.8.1 cpu scheduler," Retrieved from http://www.angelfire.com/folk/citeseer/linux_scheduler.pdf, February 2005.

[5] Charles Manning and Wookey, *YAFFS Specifications*, Aleph One, Bottisham, UK, version 0.3 edition, February 2002, Retrieved from http://www.yaffs.net/yaffs-spec.

[6] Google, Mountain View, CA, *Radio Layer Interface*, March 2009, See path '/development/pdk/docs/telephony.html' in Android source tree.

[7] Ryan Gardner, "Multi-touch proof-of-concept," htttp://ryebrye.com/files/multitouch_archive.zip, November 2008, With instructions at http://www.ryebrye.com/blog/2008/11/30/g1-multitouch-proof-of-concept-soure-code-posted/.

[8] Open Handset Alliance, "Android overview," http://www.openhandsetalliance.com/

[9] Gartner, "Mobile Device Sales Q3 2011," November 2011. http://www.gartner.com/it/page.jsp?id=1848514.

[10] Android Security Team, "Android Security Overview," December 2011. http://source.android.com/tech/security/index.html.

[11] Android Team, "What is Android?" February 2012. http://developer.android.com/guide/basics/whatisandroid.html.

[12] National Security Agency, "SELinux," January 2009. http://www.nsa.gov/research/selinux/.

[13] C. Mullaney, "Android.Bmaster: A MillionDollar Mobile Botnet," February 2012. http://www.symantec.com/connect/blogs/androidbmastermilliondollarmobilebotnet.

[14] H. Lockheimer, "Android and Security," February 2012. http://googlemobile.blogspot.com/2012/02/androidandsecurity. html.

[15] J. Oberheide, "remote kill and install on google android," June 2010. http://jon.oberheide.org/blog/2010/06/25/remotekillandinstallongoogleandroid/.

[16] T. Bray, "Exercising Our Remote Application Removal Feature," June 2010. http://androiddevelopers. blogspot.com/2010/06/exercising gourremote application. html.

[17] T. Vidas, D. Votipka and N. Christin, "All your droid are belong to us: A survey of current android attacks," in *5th USENIX Workshop on Offensive Technologies*, Carnegie Mellon University, August 2011. http://www.usenix.org/event/woot/tech/final_files/Vidas.pdf.

[18] S. Smalley, "SE Android release." SELinux Mailing List, Mailing List Archives (marc.info), January 2012. http://marc.info/?l=selinux&m=132588456202123&w=2.

[19] National Security Agency, "SEAndroid Project Page," January 2012. http://selinuxproject.org/page/SEAndroid.

[20] S. Smalley, "The Case for SE Android," January 2012. http://selinuxproject.org/~jmorris/lss2011.