

Smartphone Security Limitations: Conflicting Traditions

Nathaniel Husted
School of Informatics and
Computing
Indiana University,
Bloomington
nhusted@indiana.edu

Hassen Saïdi
Computer Science Laboratory
SRI International, Menlo Park
saidi@csl.sri.com

Ashish Gehani
Computer Science Laboratory
SRI International, Menlo Park
ashish.gehani@sri.com

ABSTRACT

Smartphones are becoming a dominant form of mobile computing in the United States, and, more slowly, the world. The smartphone, as a platform, blends a traditional general computing platform with a specialized mobile phone platform. However, each platform comes with its own tradition of social practices and policies. The general computing tradition is historically open, allowing its owners, i.e., users and administrators, to install whatever software they choose, and to add or remove hardware as they please. The cellular tradition has historically been very tightly controlled and locked down since telecommunications networks are considered critical national infrastructure. These two competing ideals clash on the smartphone platform and this clash is exemplified by the Android OS platform created by Google. The Android platform attempts to be “open” while conforming to the traditional policies of mobile phones. The conflict in philosophies between general computing platforms and mobile phones has led to fundamental limitations in the platform security of the phone. Our paper looks at these fundamental limitations and how they relate to the challenge of reconciling governance practices in use on general-purpose computers and mobile phones. We also provide certain policy guidelines and platform architecture suggestions that will help create a more secure smartphone platform.

1. INTRODUCTION

Smartphones are becoming as powerful as laptops and desktops. Smartphones ship with multiple cores and powerful graphics processors. They have robust sensor platforms containing GPS, near field communications (NFC), WiFi, Bluetooth, and cellular capabilities. They are a vault for large amounts of personal information about banking, social networks, and inter-personal communication. The capabilities and information value of the modern smartphone make it an attractive target for Internet miscreants.

Miscreants already attack smartphones with money-making malware [1]. Attacks targeting smartphones will become

more advanced as authors target software vulnerabilities in smartphones at large. Already, some malware has been seen taking advantage of escalation of privilege attacks on Android™ [2, 3],¹ Google’s smartphone Operating System (OS). Normally, patching takes care of these vulnerabilities. Patching on Android is impossible until a manufacturer/cellular provider sanctioned update is released; 40% of Android phones are still using Android 2.2 [4] even though Android 2.3.4 has been released and already supported by third-party versions of Android (e.g., CyanogenMod [5]). This slow patch progress is problematic, as Android 2.2 currently has 88 “high risk” vulnerabilities [6]. This provides a potential pool of 88 attacks that malware can take advantage of because owners are unable to patch their devices without official releases from their cellular provider.

The cellular tradition brings with it a slow patch speed due to the historical use of targeted OS software and platform governance by regulators, carriers, and manufacturers. Few feature phones² require patches unless there are extreme security vulnerabilities. When these phones are patched, it requires less work as feature phone OSs are considerably smaller than smartphone OSs. The general computing tradition, however, brings with it a more frequent patch process. Microsoft patches its OS the second Tuesday of every month. Linux-based OSs, which Android could be classified as, typically update their software incrementally as soon as a security fix becomes available. Security technologies exist to mitigate the risk of unpatched vulnerabilities. These technologies are developed by security researchers and professionals.

These security technologies can be installed on systems because those systems either have open access to software installation at any level of the platform, or provide completely open access to the platform, i.e., available source code and the ability to build a derivative OS like Ubuntu Linux. The openness exists because the general computing tradition leaves platform governance to the *device owner*. Android does not have this ability, although source code is available. The smartphone device on which the Android OS runs does not allow modification to its OS except from the manufacturer or cellular carrier. The closed access to Android is representative of the cellular tradition while the available source code is representative of the general computing tradition; both are in conflict.

¹Android is a trademark of Google Inc.

²Feature phones are limited functionality phones used primarily before the smartphone gained popularity. One example is the Motorola Razr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GTIP '11 Dec. 6, 2011, Orlando, Florida USA
Copyright 2011 ACM 978-1-4503-1082-6/11/12 ...\$10.00.

The conflicts between both the cellular tradition, a platform of *control*, and the general computing tradition, a platform of *openness*, create large security limitations. Our contribution to the literature is threefold: 1) we provide a definition and criteria to judge an “open” platform and then apply them to the Android OS; 2) we provide a discussion of how the conflicts in philosophies between the traditional phone platform and the general computing platform create security limitations in smartphones; 3) we provide a potential solution to solve these security limitations in a way palatable to all stakeholders. We focus specifically on the Android OS platform as it is currently the market leader in smartphone OSs [7] as well as the smartphone most targeted by mobile malware [8].

Sec. 2 provides an overview of the smartphone platform in general, as well as the Android OS in particular. It starts by outlining conflicts between the cellular and general computing traditions. Sec. 3 provides details on the specific threats the Android OS faces, many of which are similar to threats found on general-purpose platforms. Sec. 4 provides our definition of an open platform and reiterates the conflict between the two traditions. Sec. 5 describes in detail the security limitations that the conflicts between traditions have put on the Android OS. Sec. 6 provides both short-term and long-term solutions to the conflicts between the platform that will remove a number of the security limitations and be palatable to all stakeholders. Finally, Sec. 7 provides a summary of our arguments.

2. SMARTPHONES AND THE ANDROID OS

A general description of the smartphone platform can be seen in Fig. 1. The smartphone platform consists of two elements. The first element is a general-purpose computing environment. The second is a cellular environment used to provide radio access to the cellular network. The general-purpose computing environment handles a vast majority of the owner’s interaction with the smartphone including the user interface (UI), mathematical computations, and graphical rendering. The cellular environment consists of the baseband chip. This chip’s primary function is to interact with the cellular network. The baseband chip differs from traditional device drivers such as a wireless chip in that the baseband is generally tied to a specific cellular carrier and network type.

To make phone calls, the OS hands the audio stream to the baseband chip for processing and transmission. The OS interacts with the baseband through a standard device interface. The baseband chip’s interaction with the cellular network is the reason for its protection. This protection is to help secure the device owner’s communications as well as protect the cellular carrier’s network. This protection also creates a conflict with the traditional “openness”, i.e., the ability for owners to access and modify elements of their systems, that has occurred historically on the general computing platform. The existence of the baseband also allows a specific set of controls over the interactions between the phone and the cellular network. The separation of these platforms is due to both reliability and regulatory reasons.

The Android OS by Google is one example of a smartphone OS. Android was developed as an “open” alternative to OSs such as iOS that runs on the Apple iPhone. The Android OS uses the Linux kernel. Google has added certain drivers to support the mobile devices the OS runs on

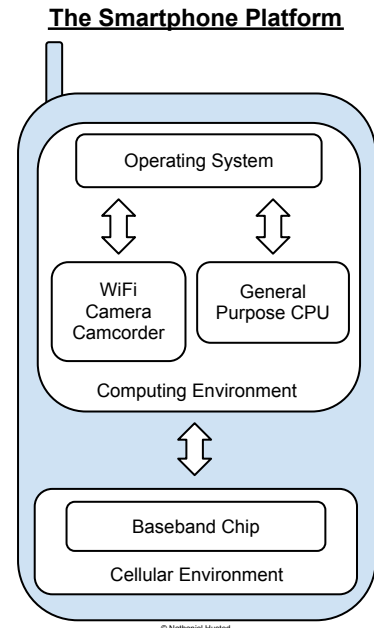


Figure 1: Smartphones differ from more traditional platforms in that they attempt to combine two major environments in one device. There is the traditional computing environment that is similar to a laptop and a cellular environment that is more akin to traditional feature phones. Each of those environments comes with two historically different philosophies on platform openness.

as well as modifications to OS subsystems. Google added its “binder” service as a new form of protected interprocess communication. Running above the kernel level are a number of native code applications that handle flash memory, debugging, log printing, and the Dalvik (a Java derivative) virtual machine. The Dalvik virtual machine spawns users’ applications in their own protected user space. For user applications to access certain functions of the system, they must access those systems through the Android Java Framework. The framework checks to make sure each application has permission to access the resource it is requesting. The Java Native Interface exists for developers to write libraries in native code that they access via Java interfaces. A visual hierarchy of Android, its vulnerabilities, and security researcher technologies can be found in Fig. 2.

The conflict between the two traditions and their visions of governance is no more apparent than in the Android OS. Fig. 2 shows that while Android is open source, to an extent, at the user and system levels, it is closed source at the hardware level. Individuals are unable to access the source code that interfaces with the base band, camera, camcorder, and other hardware devices in the system. Also, while the owner can modify elements at the user application level, the owner cannot modify elements at any other level of the OS. These access limitations create a fundamental break with the notion of an “open” platform, and the general computing tradition, but are put in place by the cellular tradition. The access limitations lead to security limitations that are inherent to the general design of the platform and not any particular component.

Android OS Architecture Heirarchy

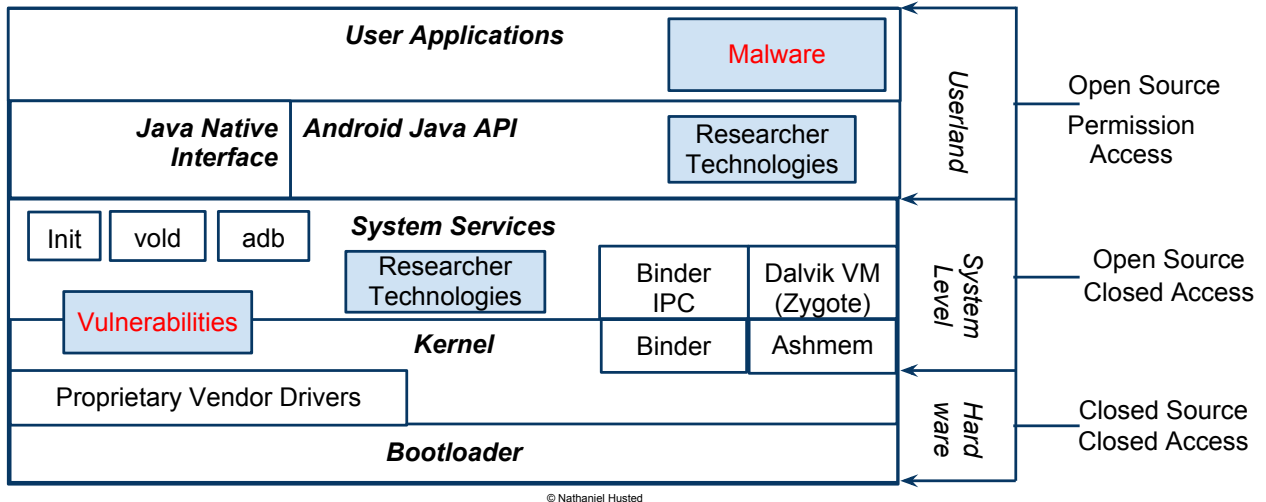


Figure 2: Architecture hierarchy of the Android platform. It covers userland at the top through boot partitions at the bottom. Android displays an open source userland but closed source drivers and bootloaders. Owners receive permission-based resource access in userland (open). All system or bootloader resource access is disallowed (closed). Security technologies exist to protect the userland from Malware, but do not help with vulnerabilities at the system level. Closed access makes mitigating these vulnerabilities difficult.

3. SECURITY THREATS

Threats *currently* found “in the wild” are rudimentary and remain in userland except for rare cases. Many threats are *very similar to threats that attack fixed computing platforms such as notebooks and desktops*. Most smartphone malware masquerades poorly as legitimate software and relies on the phone’s owner to ignore permission listings. Some requested permissions are blatant warnings, such as “services that will cost you money”. The AdSMS malware [9, 10] is an example of premium-rate text message malware that relies on dangerous permissions. Once installed, the application will collect personal data on the phone’s owner as well as send premium-rate text messages to international numbers, with the malware authors profiting from a cut of the charge. These behaviors are very similar to traditional desktop “dialers”.

Malware has even appeared in Google’s market. Two examples are DroidDream [11, 12] and Droid Dream Light [13]. DroidDream appeared on the Android market in early 2011 and Droid Dream Light appeared roughly six months later. Both applications steal personal data and are very much like traditional trojans seen on the desktop. Some researchers have referred to the Droid Dream series of trojans as a “mobile botnet” [14]. Google removes these applications from the market when found.

More advanced malware has appeared in third-party markets in east Asia. One example is DroidKungFu [2, 3], which takes advantage of an escalation of privilege exploit called “Rage Against The Cage” [15]. At least four other well-known escalations of privilege could have been used [15]. Like the Droid Dream malware, the DroidKungFu malware collects personal information about the phone’s owner and sends it to a remote server. We expect more dangerous malware like this to show in the Google market as well.

Once malware that can obtain root access gains predominance, the Android security landscape will change dramatically. Root access allows malware access to low-level OS

functionality and the ability to circumvent most OS protection. Root access allows Android malware to exhibit traditional rootkit behavior. In a worst-case scenario, the malware could flash the device ROM and replace the OS with a poisoned mimic. It could also ruin the phone as a very rudimentary denial-of-service (DoS) attack. As higher-bandwidth technologies such as 4G become more prevalent, we will see more traditional botnets on smartphone systems as well as botnets with nontraditional functions taking advantage of a smartphone’s sensor platform [16, 17]. None of these future threats, in all their potential incarnations, are *fully* mitigated by current security technologies.

4. OPEN PLATFORMS AND ANDROID

The threats faced by the Android platform discussed in Sec. 3 are not new to the field of computer security. They are modifications of threats targeting general computing platforms. Security researchers working in the general computing tradition have used *open platforms* as a base to design new security technologies and implement proof of concepts. If open-platform solutions have helped security in the general computing tradition, then they must also be useful on the mobile space if general computing threats are the focus. Thus it is important to know if a mobile platform can be considered open.

4.1 Definition of an Open Platform

We define fully open platforms as systems where i) a complete set of source code required to run all features of the platform must meet the Open Source Definition [18]; ii) the software must include at least a minimal set of build and use instructions; iii) the device owner must be able to modify the software on the device without violating warranties, use agreements, software controls, or hardware controls. The first requirement is for foundation. If the code of the platform does not meet the community definitions of open source then the platform itself could never be fully open. The sec-

ond requirement is needed because for a platform of any considerable size and complexity, a lack of documentation de facto bars owners from modifying their platforms. For example, attempting to install the Gentoo Linux distribution, one that requires everything to be installed from scratch, on a desktop and having no documentation available. The third requirement is the core to extending the original Open Source Definition to the mobile environment. If individuals can modify the source code that runs on their devices but *never install and run the modifications*, then the platform is not truly modifiable. If the platform is not modifiable, then it can never be open.

A well-known example platform that would meet our definition is GNU Linux running on desktops and laptops. The source code is freely available under the licenses that meet the Open Source Definition. Tools and documentation are available that allow an individual to build elements of the GNU Linux platform or the platform itself (i.e., a GNU Linux distribution). Owners can modify and recompile the source code to add or remove any functionality they choose. This can be done for any part of the system. They can then release this source code and know that other owners can apply it to their systems in a similar manner. Other examples of open platforms are OpenBSD, FreeBSD, and OpenSolaris (running on the hardware noted above). It should be mentioned, however, that open platforms require a symbiosis of hardware and software. If the hardware disallowed owners from modifying a GNU Linux distribution installed on it, then the device would not be an open platform, even if the GNU Linux distribution meets the Open Source Definition.

4.2 Why Open Platforms

Open platforms provide the ideal environment for defense creation because all elements of the system are available for study. Open platforms can be completely rebuilt from scratch. This allows any security modifications to be built in, tested against a threat model, and modified easily. After the system is benchmarked and documented in research, the patch for the open platform can be released. Owners can either install this patch or the patch can be built in to future user-centric security technologies. This can all be done, generally, with a minimal expenditure of resources (in context) by the researcher. Open platforms are a cornerstone of the general computing tradition.

If the platform was not open, this process would be made considerably more difficult. The researcher would have to reverse engineer large parts of the system to learn how it worked. The reverse engineering would have to be done without the help of source code as a guide. When, or if, a new security technology was created, it could not be easily implemented as the researcher does not have access to core system code. For example, if Alice and Bob found a new way to enhance security in the iOS kernel, they would not easily be able to implement and test the modification, unless they worked for Apple. However, if Alice and Bob found a new way to enhance security in the Linux kernel on a GNU Linux platform, they could easily patch the kernel, test their patch, and release the patch for others to install.

SELinux [19], Filesystem Access Control Lists (FACLs), Audit, Snort [20], and SpamAssassin [21] are all well-known examples of systems developed in symbiosis with open platforms. The SELinux [19] module for the Linux Kernel is one such example. SELinux and Audit are also two examples of

technologies that required sizable kernel patches. All the examples are widely used technologies.

4.3 Android OS: OS in Conflict

Android is the only mobile OS on the market that could potentially meet the definition of an open platform in Sec. 4.1. Because many view Android to be an open mobile platform and researchers have chosen it as the mobile platform of choice, it is worth determining how well it meets our criteria of an open platform. Even the Open Handset Alliance, which developed Android, claims it is an “open platform” [22]. By relation it is the best fit to the general computing tradition.

Android meets the first criterion, mostly. Android’s source code is available for download [23] and the licenses used fit with the Open Source Definition. This argument could be contentious if one adds the fact that Google has yet to publicly release the source code for Honeycomb (Android 3.x) even though it is available on tablet devices [24]. There is also an issue of Original Equipment Manufacturers (OEMs) releasing proprietary drivers with no open source alternatives available (see Fig. 2).

Android meets the second criterion completely. Google provides enough documentation to build the Android platform from scratch, and numerous communities exist for individuals who ask questions. Third-party distributions have released documentation that fills in gaps or corrects typos [5, 25].

Android has extreme difficulty in meeting the third criterion. System level access is closed to the owner even though the userland provides permission based access (see Fig. 2). Android closes off access to these layers by removing any form of system-level (root) access to the machine. These limitations are in place on *production devices*, i.e., devices that an entity would purchase at any Verizon or AT&T store. The lack of root access is also enforced by the cellular carrier and manufacturer of the phone via policy. If either organization has found that an owner “rooted”³ a phone, any warranty and support is null and void. While not a “physical” limitation implemented in the system it is still tightly interrelated with Android, as nearly all individuals in the United States are tied to a cellular provider.

Limitations have been put in place by manufacturers to limit owners from modifying the boot loader and other core parts of the system. The locks can limit owners in the ability to “root” a phone or install third-party OSs on a “rooted” phone. In rare cases, an owner can update the OS on a phone, but if the owner ever performs a “factory reset” the phone will no longer be usable. Motorola locks the boot loaders on its phones⁴ and HTC appears to be starting this process as well [28].

There are ways to circumvent these locks; however, circumvention has consequences. Circumvention leads to voided warranties and service agreements. With the high price of smartphones, voided warranties become costly. The removal

³Rooting a phone involves using a utility [26] to overwrite core parts of a system or using an escalation of privilege exploit to install the “su” application on the phone. The “su” application enables individuals and other applications to gain root access on a whim.

⁴Motorola has recently announced that it will stop this practice [27].

of warranties and service agreements becomes a policy control.

Android meets only one of the three open platform requirements. If one disregards the proprietary code, with no open source alternatives, required to run hardware components of Android phones, then Android meets two of the three open platform requirements. Android is not able to meet the third criterion due to a combination of *software locks* (lack of root), *hardware locks* (signed boot loaders), and policy.

The creation of locks is a representation of the conflict between the general computing tradition and the cellular tradition. According to governance in the cellular tradition, locks must be in place to protect the communications infrastructure and maintain service. According to governance in the general computing tradition, owners should be allowed to modify the software. Interest in modifying Android OS is seen in the robust third-party communities that revolve around modifying Android phones, even though this behavior violates warranties and service agreements.

5. SECURITY LIMITATIONS

The security limitations inherent in the *design* are caused by conflict between the cellular and general computing traditions. The restrictions of the cellular tradition are not malicious but are done to provide stability to the cellular network and maintain a stable service experience. However, restricting access to the system also restricts owners from taking an active role in securing their devices and they must depend upon governance from the cellular tradition.

5.1 Removing OS Modification

Access restrictions implemented in the boot loader are primarily to disable OS modification. This prevents the owner from easily injecting code that could allow the baseband device to send arbitrary information to the cellular network. The side effect of this limitation is to effectively disable the owner from actively patching the OS software on the owner's phones as the only mechanism to patch a smartphone OS is to reinstall the OS image. Even Microsoft does not actively disallow owners from patching their OS software from a third-party source, although the behavior is discouraged. The ability to update one's OS is a standard keystone to openness in the general computing tradition. Not only are owners able to update their OS on the traditional platform, but they can modify or change an OS at will. Any owner can decide to install a Linux distribution over Windows OS or install Windows OS over Linux OS.

Historically the cellular tradition limited OS installs because there was no way of modifying software on a feature phone. Even if there was an update mechanism, the source code was less available than Android OS, and there was very little choice, if any, in which OS could be installed. Because most of Android's code is freely available, a number of variants or "roms" (e.g., [5, 29]) have appeared on the Internet. These "roms" are modifications to the base Android code that allow for new features or a more stable experience. These variants appeared because there is source openness to the Android platform and variant distributions are a standard practice among Linux-based OSs like Android. Variants are a common occurrence in the general computing tradition. The ability to create variant distributions from

a common base platform is part of the general computing tradition.

The inability to update the OS would not be an issue if OS patches were released consistently and the phone was updated. The slow patch speed, or lack of patches for some products, is due to the structure inherent in the OS developer, manufacturer, and cellular provider development process. Google, upon completing an OS version, releases it to the manufacturers and eventually the public. The manufacturers then add their custom interfaces and adapt the OS to work with their proprietary drivers. The cellular carriers then provide input on what customizations they want in the OS. Finally, after both the manufacturer and cellular carrier have approved the OS, it is released. Sometimes, customers are notified that a new version is released, sometimes the patch is done over-the-air (OTA), and sometimes there is no notification besides niche online news sites. The patch process is common in the cellular tradition due to a history of small, less-sophisticated OSs and less to customize. Android's robustness, size, and frequent updates have broken the cellular patch tradition. Vulnerabilities in the OS now cause more problems and must be patched more frequently. The slow patch speed of the cellular tradition means that vulnerable phones go unpatched for months or years.

The general-purpose computing tradition does not have such issues with the patch process of its OSs as the OSs themselves are not patched as large entities, but each smaller component is patched. This creates a much easier distribution mechanism for patches. However, an incremental patch scheme does not work on smartphones due to other conflicts between the general-purpose computing tradition and the cellular tradition.

5.2 Disallowing Administration

The Android OS platform disallows its owners from administering their device by running userland applications in a permission-based sandbox and removing any ability to modify system-level components on the phone. In the cellular tradition this was never a problem. Feature phones never provided easy access to the internals of the cell phone, nor did they have advanced functionality that required such access. Smartphones now provide owners the same tools and capabilities as a GNU Linux server. This is a conflict between the cellular tradition and the general computing tradition. The general computing tradition assumes that all components can be modified. The cellular tradition attempts to remove all access so the owner does not harm the environment or modify the baseband access. If owners were able to modify the individual components, then patches could be made for individual components that were vulnerable. This incremental update would allow rapid, easy patching of insecure components without the need for a full OS reinstall. The incremental patching structure is a cornerstone of security in the general computing tradition.

The general computing tradition and, in relation, the computing security field assumes that any part of a system can be modified so that security technologies can be implemented by an owner. This is a common assumption by all Android OS security researchers as well. All research is done by modifying components below userland, components that Fig. 2 shows cannot be modified by owners. The inability of owners to access the system components of their devices means they cannot install security technologies such as

TaintDroid [30], QUIRE [31], and CRePE [32]. The process is no easier for traditional security technologies used on the desktop: Snort, SELinux, and Audit, to name a few. These technologies are depended on heavily to secure Linux-based general computing devices. In fact, SELinux and Audit are both mentioned in the NSA’s Security Configuration Guidelines [33]. However, since each of these security technologies requires system-level access to run and install, they cannot be used on the Android OS platform as is without implementation by Google or a cellphone manufacturer. Again, this is an example of governance differences regarding the platform: should the manufacturer and providers govern or the owners govern.

6. EASING THE CONFLICT

The future of smartphone security depends upon easing the conflict between the cellular tradition and the general-purpose computing tradition. While their styles of platform governance are completely different, a compromise can be found. The cellular tradition by its very nature conflicts with the notion of an open platform as feature phones have been very limited in functionality, in comparison to smartphones, and their software highly controlled. The conflict has left the Android OS platform open to any number of security threats by creating basic security limitations to the platform that must be overcome. The cellular tradition, however, does not want owners to have complete access to elements that could potentially disrupt cellular communications. This safety concern has caused smartphone developers to limit access to all parts of the OS, disallowing owners from updating their OS and modifying individual system components. In order to maintain protection of the baseband components while removing the security limitations caused by the denial of the general computing tradition, a compromise between the general computing and cellular traditions must happen. Google, as the OS owner, can be a strong mediator between the two philosophies.

6.1 Incremental Provider Updates

Incremental provider updates are one way of bridging the gap between the general computing and cellular traditions. If Google modified Android to allow more compartmentalization in its application updates, then the cellular carriers and phone manufacturers could easily allow Google to update those components. This is especially useful as system level components and above are software that the cellular carriers and phone providers do not modify (user interface elements aside). An example of how these updates are useful follows: the “Rage Against The Cage” exploit takes advantage of a flaw in Google’s “adb” system level program. If Google were able to patch the “adb” application in the Android OS without having to go through the process of updating a whole new OS image, then it would not take months or years to patch the exploit. Google already performs incremental updates to its “Google Maps”, “Gmail”, and “Youtube” applications. Incremental upgrades would increase demand from businesses for the Android platform by providing a more secure platform with a patch process that minimizes usage disruptions.

Security precautions must be taken if an incremental update process was used. Certain trust-based policies would have to be put in place, and managed by Google, similar to Microsoft’s Windows updates or Ubuntu’s public-key sig-

natures and hash checks. This infrastructure is mostly in place on Android, as the “apk” file format includes signatures identifying the developer.

Besides increased security precautions, another limitation of the incremental upgrade method is that it would be difficult to upgrade a phone’s kernel. On a general computing platform, the OS kernel is freely accessible on the partitions to which OS has access. On the Android OS platform, the kernel exists on a separate boot partition that is updated with a different image than the normal system partition. This creates potential difficulties when using an application store to modify this element of the OS.

6.2 Mobile Hypervisors

The most promising way of bridging the gap between the cellular tradition and the general-purpose computing tradition is to use a small bare metal hypervisor. A bare metal hypervisor is a virtualization platform that runs directly on, and takes control of, a platform’s hardware. The OS then runs on top of the hypervisor. Fig. 3 demonstrates how the hypervisor architecture compares to the current smartphone architecture in Fig. 1. The architecture shown in Fig. 2 will actually sit completely above the hypervisor and is equatable to the “Computing Environment” described in Fig. 3.

If a bare metal hypervisor is used, then all portions of the cellular platform on the phone can be obfuscated away from the rest of the OS. The hypervisor itself will be a small set of code (compared to the OS code base) that interfaces with the hardware and baseband portions of the phone. The hypervisor is also the place to put any telephony functionality on the device, including the ability to make phone calls and send Short Message Service (SMS) messages, i.e., all operations that send information over the voice portion of the cellphone network. This hypervisor would be developed by a consortium of phone companies and other software developers akin to the Open Handset Alliance or by Google. In the worst case each manufacturer would develop its own hypervisor, although it would be easy to show that the consortium approach would be far more beneficial for all parties involved.

The benefits of using a hypervisor platform are fourfold. First, a bare metal hypervisor will provide the least amount of performance overhead. Bare metal hypervisors on desktop and server systems currently perform at near native speeds. Second, it provides the control required by the cellular tradition. The owner and the OS both will be unable to modify the telephony portion of the device, so no low-level access to the device will be allowed. All phone actions will be in the hands of the hypervisor. Third, it provides the openness at the OS level that the general computing tradition requires. If there is no longer a threat of the owner tampering with the telephony operations of the phone, the owner should be able to install any wanted OS. There is no reason to provide limited access to the OS and any software installed in it. Fourth, the only updates the cellular companies and phone manufacturers need to worry about are to the hypervisor itself. The amount of code in the hypervisor would be considerably smaller than the code body of the whole OS. It would be more akin to the size of OSs the cell phone companies and manufacturers were used to in the days of feature phones. All OS development and patching could be left to Google. Google then has the ability to provide a consistent

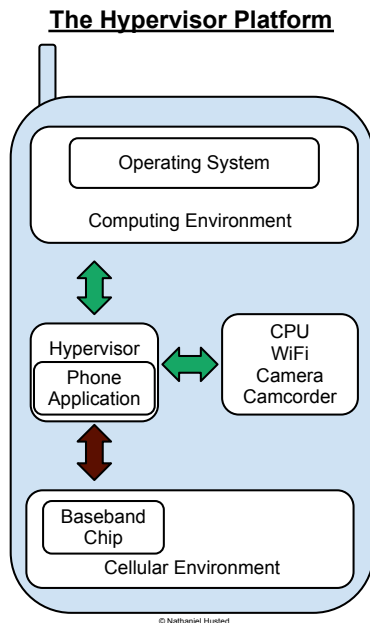


Figure 3: The hypervisor platform uses a bare metal hypervisor to manage the hardware and cellular environments. The hypervisor also controls all telephony operations on the smartphone disallowing the OS any access to the telephony components. The OS can over still access all general-purpose hardware such as the CPU, WiFi, Camera, and Camcorder. The OS will interface with the cellular data network as if it were any other wireless based network.

experience across all phones, thus increasing the demand and customer satisfaction for the mobile OS. Removing development effort from manufacturers allows them to focus on hardware stability and quality.

One drawback to the hypervisor approach is that, as far as the authors are aware, there is currently no virtualization technology for the ARM platform that is similar to Intel’s VT-x or AMD’s AMD-V that exists in released phones. This, however, will most likely change in the future as ARM already showcases virtualization technologies for its processor designs on its website [34]. Another drawback is that mobile OSs will need to be modified to specifically work with each type of hypervisor. Customization is done with special drivers or OS modifications. It is needed to maximize performance. If the mobile hypervisor market becomes fragmented, the customization process could become costly, although chances of this fragmentation are slim. The market pressures in the smartphone market mean that most manufacturers will follow the cheapest cost solution. One final drawback is that mobile hypervisor technology has yet to be implemented to this extent; however, the basic hypervisor technique has been demonstrated by the HoneyDroid [35] and L4Android [36] projects.

7. CONCLUSION

Android OS is currently the U.S. market leader for smartphone OSs [7]. It is the most popular OS platform for security researchers. Its popularity among the research community is because of its relationship to the general computing tradition and its view on “openness”. Android still has

many closed portions to its platform. The closed portions stem from Android’s use in the cell phone market, a market whose traditions favor closed access. The conflict between the openness of the general computing tradition and the restrictiveness of the cellular tradition creates a number of security limitations in Android.

The *security limitations* due to the closed nature of the platform as a whole have not been discussed while *vulnerabilities in individual components* have been covered thoroughly [37–39]. The lack of root access and the sandbox permission architecture, while providing protection against poorly programmed applications, do not prevent malicious applications from exploiting vulnerabilities in the system layer and kernel, but limit the capabilities of security applications installed in userland. These vulnerabilities, if a device also includes boot loader based limitations, could lead to disastrous consequences for the device’s owner. These vulnerabilities exist because of Android’s basis in the general-purpose computing tradition. A tradition that understands vulnerabilities will happen but that they will be patched quickly after being found. The cellular tradition, however, does not have a history of quick patching so when the vulnerabilities from the general computing tradition meet the slow response time of the cellular tradition, owners are left being unable to secure their mobile devices. If the OS of the device were more open, as in the general computing tradition, owners would be able to patch these vulnerabilities without having to await a response from their cellular carriers.

We propose two methods to alleviate security limitations caused by the conflict between the cellular tradition and the general-purpose computing tradition. The first method is to provide more fine-grained updates to the system-level components of the OS. If Google is able to update individual components such as “vold”, “adb”, and “init” independently, just like other phone applications, then there is a possibility for far more rapid patching. There will be no need to wait the year it takes for the complete OS patch to go through the manufacturer and cellular provider, if that process even happens. The second longer-term method is to develop a bare metal hypervisor to run on the ARM platform. This hypervisor will encapsulate all telephony operations while providing the owner free access to modify the OS portion of the hypervisor platform as seen fit. Both these suggestions protect the interests of the cellular tradition in regard to the baseband. While the first suggestion does not support the openness that exists as part of the general computing tradition, it does provide mitigation of one of the security limitations. In the longer term, however, the implementation of a hypervisor will allow full openness at the OS level of the devices.

The conflicts between the two traditions are inherent in all parts of the modern smartphone design. We assume that much of the conflict stems from designers and engineers considering smartphones to be far more phones than they are general computing devices, although this has changed as mobile technology has advanced. These conflicts, however, must be acknowledged and rectified to move on to properly secure the future set of mobile computing devices with telephony functionality.

ACKNOWLEDGMENTS

The project described was funded under contract by the U.S. Department of Homeland Security (DHS) Science and Tech-

nology (S&T) Directorate. The content is solely the product and responsibility of the authors and does not necessarily represent the official views of DHS.

This material is based upon work supported by the National Science Foundation under Grants OCI-0722068 and IIS-1116414. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

- [1] T. Wyatt, "Security alert: Android trojan ggtracker charges premium rate sms messages," <http://blog.mylookout.com/2011/06/security-alert-android-trojan-ggtracker-charges-victims-premium-rate-sms-messages/>.
- [2] "Another android malware utilizing a root exploit," <http://www.f-secure.com/weblog/archives/00002177.html>.
- [3] "Security alert: New malware found in alternative android markets: Droidkungfu," <http://blog.mylookout.com/2011/06/security-alert-new-malware-found-in-alternative-android-markets-legacy/>.
- [4] "Android developers: Platform versions," Nov. 2011. <https://developer.android.com/resources/dashboard/platform-versions.html>.
- [5] "Cyanogenmod: Building from source," http://wiki.cyanogenmod.com/wiki/Building_from_source.
- [6] A. Kingsley-Hughes, "88 'high risk' vulnerabilities discovered in android 2.2 'froyo'," <http://www.zdnet.com/blog/hardware/88-high-risk-vulnerabilities-discovered-in-android-22-froyo/10217>.
- [7] "Android leads in u.s. smartphone market share and data usage," <http://blog.nielsen.com/nielsenwire/consumer/android-leads-u-s-in-smartphone-market-share-and-data-usage/>.
- [8] A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 3–14, ACM, 2011.
- [9] "Quick snapshot of trojan:androidos/admsms.b," <http://www.f-secure.com/weblog/archives/00002174.html>.
- [10] "Old trojan tricks on android," <http://www.f-secure.com/weblog/archives/00002171.html>.
- [11] "Update: Security alert: Droiddream malware found in official android market," Mar. 2011. <http://blog.mylookout.com/2011/03/security-alert-malware-found-in-official-android-market-droiddream/>.
- [12] V. Svajcer, "Aftermath of the droid dream android market malware attack," <http://nakedsecurity.sophos.com/2011/03/03/droid-dream-android-market-malware-attack-aftermath/>.
- [13] L. Dignan, "Malware sneaks by google's android market gatekeepers again," <http://www.zdnet.com/blog/security/malware-sneaks-by-googles-android-market-gatekeepers-again/8696>.
- [14] "New droiddream variant found on android phones," <http://www.f-secure.com/weblog/archives/00002170.html>.
- [15] J. Oberheide and Z. Lanier, "Don't root robots!," <http://jon.oberheide.org/files/bsides11-dontrootrobots.pdf>.
- [16] N. Husted and S. Myers, "Mobile location tracking in metro areas: malnets and others," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 85–96, ACM, 2010.
- [17] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundminer: A stealthy and context-aware sound trojan for smartphones," in *Network and Distributed System Security Symposium (NDSS)*, 2011.
- [18] "The open source definition (annotated) version 1.9," <http://www.opensource.org/osd.html>.
- [19] "Security-enhanced linux (selinux)," www.nsa.gov/research/selinux/.
- [20] "Snort," <http://www.snort.org/>.
- [21] "The apache spamassassin project," <https://spamassassin.apache.org/>.
- [22] "Open handset alliance faq," Nov. 2007. http://www.openhandsetalliance.com/oha_faq.html.
- [23] "Android open source project," <https://source.android.com/>.
- [24] A. Vance and B. Stone, "Google holds honeycomb tight," http://www.businessweek.com/technology/content/mar2011/tc201110324_269784.htm.
- [25] "Android-x86 project - run android on your pc," <http://www.android-x86.org/getsourcecode>.
- [26] "[program] updated:one click root/unroot (mac and pc)," <http://forum.xda-developers.com/showthread.php?t=739304>.
- [27] Z. Stinson, "Motorola says 2011 devices to have 'unlockable/relockable bootloader'," <http://www.androidpolice.com/2011/04/26/motorola-says-2011-devices-to-have-unlockablereckable-bootloader/>.
- [28] J. Stekl, "It wasn't just verizon – the htc incredible s' bootloader and recovery are signed as well," <http://www.androidpolice.com/2011/03/23/it-wasnt-just-verizon-the-htc-incredible-s-bootloader-and-recovery-are-signed-as-well/>.
- [29] "Rom kitchen," Nov. 2011. <http://www.romkitchen.org>.
- [30] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of OSDI*, 2010.
- [31] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. Wallach, "Quire: Lightweight provenance for smart phone operating systems," *Arxiv preprint arXiv:1102.2445*, 2011.
- [32] M. Conti, V. Nguyen, and B. Crispo, "Crepe Context-related policy enforcement for android," *Information Security*, pp. 331–345, 2011.
- [33] "Guide to the secure configuration of red hat enterprise linux 5: Revision 4.1," http://www.nsa.gov/ia/_files/os/redhat/rhel5-guide-i731.pdf, Feb. 2011.
- [34] "Arm virtualization extensions," Sept. 2011. <http://www.arm.com/products/processors/technologies/virtualization-extensions.php>.
- [35] C. Mulliner, S. Liebergeld, and M. Lange, "Poster: Honeydroid-creating a smartphone honeypot," in *IEEE Symposium on Security and Privacy*, May 2011.
- [36] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4android: a generic operating system framework for secure smartphones," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 39–50, ACM, 2011.
- [37] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *Security & Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010.
- [38] W. Enck, D. Ocateau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proc. of the 20th USENIX Security Symposium*, 2011.
- [39] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *Security & Privacy, IEEE*, vol. 7, no. 1, pp. 50–57, 2009.