[redacted]

[redacted]

Android OS and Device Technical Risk Analysis

Version 0.14

May 6, 2013

Document History

Document Version #	Issue Date	By	Description Of Revision(S)
Version 0.5	March 19, 2012	Amélie Koran	Final Working Draft
Version 0.7	March 27, 2012	Amélie Koran	ITTI Formatted Draft
Version 0.8	April 23, 2012	Amélie Koran	Added HTC Vivid/Sense Details
Version 0.9	May 1, 2012	Amélie Koran	Spell, Grammar, and Flow Check
Version 0.13	November13, 2012	Amélie Koran	Formatting Fixes
Version 0.14	May 6, 2013	Amélie Koran	Updates Jelly Bean (v4.2) API 17

Table of Contents

1 Executive Overview

This document is intended to provide a risk analysis of the possible operation of the Android™ OS (operating system) and supporting devices within the ***** information systems environment. Due to the constantly changing nature of the platform and the marketplace, this document is structured towards general concerns and if a persistent risk exists in Android's use within *****, it will be discussed and mitigation techniques suggested enabling these technologies to function properly and securely. This is a living document, and will consist of appendices for updates and new versions as major enhancements are added.

2 Introduction

The Android™ OS is a Linux-based operating system that is maintained by Google for use on mobile platforms such as smartphones, tablets and media devices similar to those from Apple (iOS), Microsoft (Windows Phone), HP (WebOS), Symbian (Nokia) and Research in Motion (Blackberry OS & QNX)[i]. Typically, these types of devices run a "real-time operating system" (RTOS)[ii] which is designed around a preference for application and process scheduling for a finite set of functions (and applications), in order to increase performance (and outcome predictability) and responsiveness of the device running it. Notable forms of this type of operating system are QNX (now owned by Blackberry) and Windows CE (parts of which are in Windows Phone, formerly Windows Mobile) and are often bound due to the CPU architecture chosen, in most cases, ARM[iii], which are designed specifically for low power and compact hardware applications. Android™ shares many of the same design characteristics but leverages the Linux kernel, which is usually utilized in complex desktop and server OSes, as a base from which to build system releases upon. This choice creates several benefits, as well as detriments, to the platform, which will be discussed later in the document. The other notable factor, especially to its meteoric rise in the world of smartphones and tablets is, that it is a free and open source software platform (otherwise known as FOSS) which, in turn, is very attractive to companies desiring to decrease overall development and maintenance costs.

3 Scope

The scope of this document is to provide a relatively complete overview of the risk in deploying Android™ OS, Android™ applications, and associated devices within the [redacted] information systems infrastructure. This document will focus on the operating system and applications rather than on the devices themselves; while as the hardware changes rapidly, the operating system is a general constant among them, and is generally the core that brings those devices the desired functionality. However, in regards to hardware, the two most common form factors, that of a smartphone and of a tablet, will be discussed, as the operating system is optimized depending on the deployment form-factor and intended use. The commonality of these hardware platforms is that of an embedded system[iv] designed specifically and purpose built for those functions; which, unlike a desktop or server that are designed for expansion and modification based on their application and use. . Much like those risk assessment profiles generated for iOS devices and associated software, this risk assessment is based on the current knowledge and environment

available at this time and should be updated when there is major release or new issues are discovered.

4 History

The use of, and leveraging system architectures from, older established sources are well-worn concepts in computer and software engineering circles. The use of purpose-built computing devices go all the way to the origin of the first computers, from Babbage's Difference Engine (calculations) to the Bombe[v] machines at Bletchley Park in WWII (crypto analysis), and has maintained that same efficiency of that design to this day. The combination of these two concepts gave rise to embedded computing, taking what was specialized software, and "installing" it on these custom built devices to reduce the cost of operation, increase the maintainability and reliability of system operations, and allow for these systems to be tasked with regulating operations in a larger environments that require a high-predictability of basic operations (such as traffic lights, factory lines, and power plants).

Over the intervening years, as the proliferation of computing devices from the military, factory, and business environments started to come home in the form of desktop PCs (such as the PC Jr., Commodore 64, TRS-80 and even scientific graphing calculators) the consumerization of technology started driving advances in the fields that gave birth to these systems. The migration of video games from the arcade to the home were some of the first overt uses of embedded systems for something other than business activities, and in this case, leisure. The next barrier to fall towards pervasive involvement of embedded systems in people's day-to-day lives was the cell phone. It started leveraging the need to have reliable systems managing phone calls (analog signal conversion, frequency switching, power consumption and battery life, transmission quality, etc.), well before they ever became "smart", and proliferation became quite severe as adoption and costs were driven down[vi]. About the time cell phones were growing in use, the first PDAs or "personal digital assistants" hit the scene, and while very basic at their start, it took even less time for these devices to become more complex and multi-functional. The first one that gained wide acceptance was the Psion Organizer II, which resembled an HP scientific calculator, but actually allowed for the programming of applications specifically for the platform (your first "apps"), and the ability to use modular storage to retain information as well as to store those programs on the device. Oddly enough, the expansive platform, which included a serial port (the famous RS232), was used to interface with other embedded systems, such as measuring and instrumentation equipment, besides also having other ports for connecting telephony equipment, barcode scanners and printers.

Finally, in 1992, Apple released the Newton, the prototype, in a way, of what promise was in store for these classes of devices, even if they were less than a perfect piece of technology. It was the first device to be officially termed a PDA, but also had a form factor and interface layout that, even after 20 years, is present in the current crop of smartphones and tablets; including how the operating system and "apps" work in consort with each other. With Apple's stumbles in this area, Palm and its Pilot devices really defined the multiple roles these classes of devices could play both in business and consumer markets. Nokia, with their 9000 Communicator was the first to bring telephony to a PDA, and thus a "smartphone" was born and this market segment began

to grow. Additions to the functionality included 802.11 WiFi, Bluetooth, IrDA, soft keyboard (touch screen), gesture-based/multi-touch screens, and both wired and wireless syncing of data and other media between computing devices and software (calendars, email, web browsers, music and video players). To almost come full circle, the other top use of smartphones and tablets, besides the communication features noted earlier, are to play video games and view other entertainment. Now the smartphone market sells upwards of 150 million devices each year, and recent research puts Android-based systems at close to 50% of all types of these devices.

5 Current State

Currently, since the start of development of Android™ OS, there have been nine major versions of the platform released in one form or another for use on compatible devices, SDKs (software development kits) and IDEs (integrated development environments). The first commercially viable version was v1.5, or commonly known by the development code name "Cupcake" (2007); however, it really gained prominence among consumers with the release of devices running v2.2, known as "Froyo" (2010), namely due to a wealth of feature additions and performance enhancements. Notably, this release was shortly after the release of Google's own reference hardware platform (used internally among developers, and the platform on which all new releases were based off of called Nexus One (the successor recently being the Nexus S). This was an ARM-based device, much like early iPhones, and was developed as the demonstration tool as to the power and usability of the Android™ OS platform. It was essentially democratizing the Android™ OS by being released as an unlocked (multi-carrier) GSM and UTMS compatible device to work on as many worldwide service providers as possible.

Since that time, the platform's base of manufacturers, models available, form factors, carriers and Android™ versions themselves have increased significantly. Currently, Android™ OS has been released at Version 4.0 ("Ice Cream Sandwich") and is designed to operate seamlessly on phone and tablet format devices, avoiding the necessity to maintain different branches and feature sets within the OS development. Recently, the cycle of major versions has overtaken that of its nearest competitor, Apple, who holds tight control on its platforms, including its own proprietary and custom mobile processor, the "A-series". Due to the low cost of maintaining the reference platform, notably, the cost is borne on Google rather than the handset manufacturers and service providers (as is the case with "feature phones"), it has been very lucrative and attractive, and conversely has allowed for the growth of available devices available to consumers. The ties have grown closer in the intervening years between Google and its partners in this technology through the founding of the Open Handset Alliance. It contains almost every major hardware and service provider short of Apple, Microsoft and RIM, which are obvious competitors. This allows for many compatible devices to exist in development pipeline, from the supply chain through to the service provider delivering them to customers, to have their delivery to the market telegraphed very closely, regardless of how many companies are involved.

However, there are flaws, as there are with any platform, be it through the architecture and engineering, to the sales and marketing of the device, and services in use on the platform. In the most obvious instance, much like the Linux distributions intended for desktops, laptops and servers, each service provider, such as AT&T, Verizon, and Sprint, create carrier specific tweaks

to their release for their support ecosystem of devices that creates a lag between the reference release from Google and updates and patches to the "in service" hardware platforms of the service providers. This leaves a suspicious gap between security enhancements (patches) that are often taken advantage of, both by developers and hackers alike, regardless of the intent of the person leveraging these issues. A study by Bit9[vii] calls into question the long-term supportability and security model of Android™ based on this fracturing of development versus maintenance. As the install base grows and market share potentially increases, the percentage of vulnerable mobile devices operating on cellular and 802.11-based networks will also grow putting companies at significant risk.

6 Analysis

Like iOS, Android™ draws its roots from FOSS operating systems, in this case, a Linux kernel versus one derived from BSD, both with UNIX and UNIX-like capabilities and features. Much like iOS, when moving up the stack from kernel to the "user land" area of the operating system, the platform specific and custom components of functionality and user interfaces begin to significantly differ than those you'd see on a desktop or server OS derived from the same root or core. Also, at the lower level, due to the embedded systems platform, the need for custom drivers for such specialized hardware are required, and those used for commodity platforms can rarely be repurposed. Finally, due to the shortcomings of developing for a mobile platform, direct translation of how the system operates for a user (such as system responsiveness, interface paradigms, and capabilities) do not exist in a format that can be ported from the desktop/laptop/server environment, and require significant adaptation or may be in complete absence on these platforms.

Android™ devices, in particular, suffer due to their technical origins and licensing model among vendors, a level of fracturing that makes maintenance and uniformity among system and security configurations difficult. Much as Linux has had multiple distributions, fracturing the community based on capabilities and packaging (such as Debian and Redhat with .deb and .rpm formats) for applications, the Android™ platform (also Linux-based) has had a split between the official Google release and a more feature rich firmware "fork" called CyanogenMod[viii], which often has features Google refuses to enable in their stock releases. This is somewhat akin to a jailbroken iOS device, thus enabling feature and capabilities that are not approved by Google, such as the case of enabling tethering, that cuts into a service offering from a voice/data carrier. Other fracturing to the Android™ OS appears at the hardware and service provider/carrier level, in which some core platform phones and devices are used between carriers. They have versions of Android™ OS running on them customized for that carrier, with features and tweaks that make patching and upgrading incredibly difficult and lag generally acceptable windows for securely supporting the platform, creating a vulnerability gap on users' devices.

Delivering applications or "apps" in the mobile world, on the Android™ platform, is essentially anarchy in lieu of allowing for an open-platform for developers and users alike. There are several "authorized" Android™ "App Stores", such as those from Google and Amazon, but several "wild" stores have also been available for applications that may have some dubious origins but also cater towards different communities, such as developers, gamers, and those with

different spoken and written language (and cultural) requirements. These may include stores created and curated by carriers and service providers themselves much in the model that existed before iOS and Android™ devices with early smartphones and feature phones. This is, in this case, strictly to protect an established revenue model, but in turn could risk the platform by restricting access to community review and refinement, which often plays a role in OS vendor sponsored stores. In cases where applications have gotten away from Google, there have been instances of a "kill switch" being activated to remove or disable misbehaving or dangerous apps, and it has been reportedly leveraged by Google and partner service providers several times in the last few years. This is a more "post facto" solution to the issue of application management and developer vetting versus a more upfront model in use by Apple and iOS.

7 Risk

In short, there are a number of risks in the operation of an Android™ -based environment, both similar and are also vastly different than that of operating a similar iOS environment. The risk profile for operation stems into three major categories, those of software (OS and applications), hardware (device and accessories), and service provision (data carrier and local provider, such as WiFi and cellular). Some of these risks can be directly affected and managed by *****, and others cannot. While some mitigation capabilities and techniques exist, there still remain holes in truly providing a secure platform. This list is broken down below by the risks in these categories, and weighted based on business impact, user impact, likelihood of occurrence, likelihood to be resolved, and severity of the issue.

7.1 Software

7.1.1 Android™ OS Install Base

As noted earlier in this document, the market share of Android-based systems is incredibly high. Conversely, any issue with the system, such as a major vulnerability or other security or functionality issue leaves a technology environment of millions of potential problems around the world. If infected, the magnitude of the problem could be amplified very quickly, spread rapidly and cover a greater amount of systems than even those during a PC-based virus, as smartphones are typically "always on" and on a network.

As part of the notion of an open system, the Android™ OS in order to provide smart, location-based features, many privacy considerations should be made when utilizing these features. Location-aware applications, GPS, and general service provider components can expose users to the leakage of personal identifiable information (PII) to unknown (to the user) groups, individuals and organizations. Most recently the Carrier IQ[1] debacle, in which a performance monitoring component installed by the service provider on multiple mobile OSes created an uproar once discovered, and resulted in a backlash against the practice by carriers as well as those manufacturers who allowed such components to be installed without the user's explicit knowledge.

[1] http://en.wikipedia.org/wiki/Carrier_IQ

[2] Typically cryptographic keys have shorter lives to avoid eventual reuse or forging

7.1.2 OS Security Model

Much like iOS, Android™ shares the core kernel with a well-understood and open platform, Linux, but due to the demands of the device and intended operations, components are added to the kernel (modules and drivers) as well as new user-land (applications and user interface) features. In this case, as these devices and OS distributions are new, they often haven't fallen under the same level of scrutiny as other Linux distributions, which include specialty drivers and modules that enable the advanced system and communication features. Along with the "newness", since other parts of the operating system are based on older code, they are conversely subjected to the same kernel problems within Linux, which occur quite regularly, and often due to the kernel's role in the OS, are very severe.

As a difference in licensing and access models for Android™ in comparison to iOS, there are several anti-virus and malware programs already released by major vendors, which help with protecting the system when installed and managed correctly. However, these companies also tend to focus on protection for Windows-based systems and their talent for developing useful Linux-based protection schemes should still be looked at very carefully. In other words, just because it exists and can be installed, it isn't a panacea; as they also fall prey to poorly written signatures and heuristic protections which can reduce the functionality of a device at inopportune times (such as quarantining a required system file or library). As with desktop and laptops, the involvement of an active AV/AM system on the device has impacts on performance and battery life.

7.1.3 Application Security Model

One of Android's biggest marketing coups for consumers over iOS was its claim to run Adobe Flash; and thus opened up the usability of Flash heavy sites to this platform over the HTML 5-only iOS. However, most security professionals will note that Flash is one of the most heavily exploited (along with Adobe Acrobat PDFs) formats on the Internet, thus increasing the risk for system compromise and/or failure. iOS has a native PDF rendering library, engineered not to utilize stock Adobe utilities, whereas Flash for Android™ is supplied and maintained by Adobe, and shares the same vulnerable source as its Windows and Mac OS brethren.

Android™ executes processes via the Dalvik VM[ix], a Java-based incarnation of a register-based virtual machine environment. In this case, the use of a register-based VM versus a stack-based VM, helps in unique and direct addressing of known memory locations versus a stack model which can grow due to the dynamic nature of that model. This aids in security (and speed through addressing) by preventing the non-deterministic "stack overflow" conditions inherent in many operating systems that allows for memory to be over written and malicious code to be run or systems to be crashed. Most Android™ hardware devices actually have a hardware implementation of this VM on their boards, but Google actively chooses not to utilize it and virtualizes it via software. This model also limits the amount of lower level functions that can be exposed to developers and can be leveraged to reduce potentially risky operations (predictability and access control), besides just memory access.

Unlike iOS and other mobile OSes, upon installation, the application will cause the Android™ system to bring up a screen that lists what permissions and access the application being installed wants access to. At this point, the user can allow or deny these actions (in some cases disallowing enough to make the application non-functional). However, often these applications request permissions they don't often need, and combined with many users' desire to "just get to the installed application" often ignore or don't understand the implications of allowing certain levels of access. In many cases, a mobile device management (MDM) system could audit these permissions and ACLs (access control lists) and tighten them when needed, but it may be unsure if the one selected and in use by DOI contains that level of granularity.

7.1.4 Application Store (Google Play/ Android™ Market)

Ever since the days where mobile phones have done more than keep a contact list and allow phone calls; service providers, like Verizon, Sprint and AT&T have wanted applications and games on these devices. As technology improved, and the realization that there can be a revenue stream in providing such features (including media such as music and ringtones), external developers were brought in to devise tools to be installed and run on phones. Creation of vendor supported marketplaces (via Apple, Google, RIM, and Palm) have generally supplanted those run by service providers as the supply model has changed to focus on the OS rather than specific hardware or service vendor. In turn the trust model has changed and shifted from somebody you've paid as a consumer of services (such as data and voice) to those who provide the "guts" of the phone. This essentially extends a service agreement well beyond that from which it is drawn upon for the original device purchase; thus indemnification of parties if anything detrimental (such as malware, theft of data or services, etc.) should occur becomes difficult and legally troublesome. In essence, an agreement would need to be entered into with the service provider (data and voice), the application seller (in aggregate, Google and Apple) as well as any implied service and licenses with the 3rd party developer.

With Apple and their iTunes App Store, they market a trusted "walled garden" approach to application submission, approval and distribution, supposedly ensuring no "unchecked code" is released to the public. Google's approach with their application storefront is of a "free for all" that is policed upon detection or reporting of potentially harmful applications, but in essence allows anybody to sell anything. There have been a number of cases where malicious applications have been found in the Android™ Market (both intentional and some developed by security researchers) due to this openness. In these cases, Google (although disavowed by them) leveraged a kill-switch to these applications that deactivated them and removed them subsequently from download on the store. However, Apple has had several issues of their own with the App Store, so the correctness of the operations model or security architecture for either store is still debatable.

As with iOS, applications can be loaded locally without going through any storefront, either through the use of development tools, or other means, such as "rooting" the phone or flashing an unsupported ROM and firmware, or just downloading from an external URL. This can allow for greater control for enterprise deployments, but if not managed carefully can result in the installation of questionable software and possibly impact performance and functionality to a

point where the maintenance of individual Android™ devices becomes difficult. Through these means, as well, if they are preloaded or installed during a modification such as a firmware or similar "mod", the user is not directly prompted for application permissions and access, and places the onus on the installer to vet whatever will be placed on these devices.

7.1.5 Software Development

As noted earlier, with the application delivery model changing from service provider to device/OS vendor, a shift in who delivers applications to the user and devices also has changed to encompass mainly third-party developers. This doesn't differ much from the model traditionally used for desktop computing like those from Apple and Microsoft. However, the actual model for development has changed significantly, which is now a task-based formulation, consisting of the installation and use of smaller limited-function "apps" (in itself a shorted form of "applications") rather than a "suite" or "multi-function" applications that are seen in desktop computing. This often reduces the financial investment of the user or organization, with apps being anywhere from free to costing a few dollars versus hundreds to thousands of dollars for suites. A model such as this allows for the device to be customized based on user task goals (high utilization) versus having everything installed (low-utilization) for some minimal use cases of the device or system. The impact on security of the device is two-fold. First, the attack surface is somewhat reduced for "apps" versus "suites" as there is less unutilized code residing on the device that can be potentially compromised, but secondly with a number of "specialty" applications sitting around from multiple developers, the uniformity to secure coding standards, quality control/assurance, and maintenance (i.e. upgrades and patches) may vary greatly, thus putting the devices at risk.

Elsewhere in this document, it is noted that the implied trust infrastructure for this now more mobile code in the form of apps is quite extensive. Most models, including Google and Apple, models require acquisition of a signing certificate (albeit forgeable and transferrable) and developer agreement for a minimal outlay of money in an attempt to bureaucratically restrict access to devices, and for others to put the risk acceptance of this software development model squarely onto the user or other maintainer of the device and its operating system. The latter is much more the fact for Android™ devices, as their store model allows anybody willing to write an application to have it for sale (or for free) on the Android™ Market and a payment an even smaller developer fee ($25) than those for Apple or Microsoft ($99 each). Albeit that the applications are signed much in the same way Apple utilizes, the life of the private cryptographic key or "crypto period"[x] (at the time of this writing, it is supposed to expire after October 22, 2033) is exceptionally long, over 31 years, and often goes against generally accepted PKI practices[2], including NIST SP800-57.

For "apps" approval, the quality control/assurance level can be considered somewhat lower than that imposed by Apple, although one would attempt to leverage ratings and user feedback within the Android™ Market as an assurance of a functional application; however, as proven in the iTunes App Store, this too, can be manipulated. Since the coding practices are not entirely

[2] Typically cryptographic keys have shorter lives to avoid eventual reuse or forging

enforced, the probability of an application crashing or leaking information is slightly raised, and in turn can compromise the stability and performance of the Android™ device. It is vitally important to manage user feedback and support calls in a "loose device" management paradigm, as systemic issues may not be identified and remedied if not carefully attenuated to.

7.1.6 Configuration Management

As with the deployment of Windows Phone and iOS devices, the use of a mobile device management (MDM) solution is being leveraged in order to maintain some control over the configuration and management of mobile devices. The level and granularity to which the MDM tool can support these various devices is of concern, as some security controls are not exposed in a fashion or to a level that the MDM tools are developed towards. Since multiple platforms are under management, uniformity of policies between devices and what functions and features they will control will differ significantly, and in some cases may not me manageable at all, and in turn leave the device vulnerable or incapable of reporting useful data.

The concept of BYOM/BYOPC (bring your own machine/PC) for personally owned equipment (POE) is gaining traction and has been pushed by OMB to help reduce costs and overhead for IT within various agencies. However, attempting to enforce government-grade policies on devices that were bought with personal funds, which may have personal data co-mingled on the device, and during off-hours, are used for something other than work, treads a very dangerous line in liability if any data is lost, misappropriated, or even if the hardware is modified in order satisfy policy. If users decide to modify their systems in this scenario, such as using CyanogenMod or other feature enhancements, the responsibility of managing software versions and patches also becomes a difficult subject to handle. Will users be required to track current releases based on their modified device, or be forced to stay with a service provider approved version, which often lags in availability and forced deployment? Conversely, if a user has an expectation of privacy and accessibility to their own data, ***** may also be required to treat the PII from that component of the device the same way as if it were USG if a logical or physical separation is not possible. This may run into difficulties including running multiple versions of an application to satisfy both private and work use, and the device may be configured to handle data in two distinct and separate ways.

7.1.7 Architecture

The Android™ OS, as noted above, is based on Linux, similar to another Google project, Chrome OS. The application layer above the core OS (kernel, middleware, API, libraries and drivers) is based on a Java-based virtual machine (VM) called Dalvik and runs just-in-time (JIT) byte code, called dex-code. Until recently, Android™ had its own display and interface system, but recently, in March 2012, had an initial X-Windows system ported to the platform. This reflects the differences between the structure of the Android™ OS architecture, enough so, that Google has now refrained from trying to integrate their code changes (as part of the GPL) back into the mainline branches of the Linux kernel. Because of this "forking" it is Google's responsibility to port Linux specific patches back into the Android™ tree, including those involving security and performance components.

The core of the OS is structured around providing storage, media, communication, and input (interface) layers abstracted through various interfaces to the hardware platform. For the media layer, components are built to support graphic displays, utilized by multimedia, gaming and other visual feedback tools, and audio, which allow for the playback of sounds and music as well as the recording of the same. There's an overlap of this to the communications layer, such as that for analog sound capture and playback, for the phone component, their functions are shared. Also in the communications layer, the radio communications are managed, which include the above analog, but also digital communications, such as those for computer data using protocols such as 802.11, Bluetooth, and cellular data (GSM, EDGE, CDMA, LTE, etc.). The storage component allows for the temporary or semi-permanent keeping of digital data on these devices through the use of NVRAM (non-volatile RAM) and other technologies. This storage component also enables the use of some of the media components by allowing the on-board keeping of multimedia such as MP3s and video files. The interface-layer is involved with allowing the user to access various features of the device and OS, and includes components that manage the control and functionality of input components (such as dock, audio and video connectors) but also any buttons and touch surfaces that are used. This layer can also be described as the layer that unifies the media components into a working visual display for applications (a GUI versus a CLI) and how various methods are used to access that data (a virtual or hard keyboard, multi-touch screen, gesture-based input).

As part of the VM, the applications on the device run in a sandbox as part of a layer designed to separate potentially harmful activities from core functions of the kernel. A permissions layer controls this VM sandbox, where specifically categorized access levels are exposed to the user and applications. The current design expects the user to control these permissions through direct user queries when the calls from the application wish to access these functions. The user (or via pre-loaded policies) can control this access, but because these controls are specifically placed in the hands of users, inconsistent controls may vary from user-to-user and device-to-device. These are rather coarse permissions that are presented. Lower-level fine controls, such as access to individual drivers and system components, are not usually exposed. Sadly, the data produced by components on the device, such as geo-location information for a GPS, is readily available and not precisely controllable in this layer. However recent privacy concerns and news about loose controls and exposure of this data to applications and thus to the developers, has led to a backlash that has Google rethinking how this data is managed.

7.2 Hardware

7.2.1 Short Platform Refresh Cycle

Unlike the Apple model for iOS devices, the array of devices and form factors that are intended to run Android™ differ greatly. In fact, the rise of tablet computers required Google to adapt and release a tablet-only version of Android™ called "Honeycomb" until they could merge back the support into the main branch of Android™ in the "Ice Cream Sandwich" release (Android™ 4.0). The detriment to the consumer, as well as understanding long term support of the device and supporting operating system, is that each incremental improvement and different release by a multitude of vendors may introduce bloat into the OS as well as possibly introduce

vulnerabilities due to the requirement to support "non-standard" hardware architecture for that release, as well as any agreed upon legacy or longer-term support.

The short product cycle also produces a "throwaway" mentality to the platforms, with a push by early adopters to constantly acquire new devices and older ones fall into disrepair. This creates a risk at both ends in regards to support, as bleeding edge devices often have bugs discovered in them within the first few days and weeks of public use, fixes may be long in coming. Conversely, as older units are still in use, their long-term support options begin to wane, but they still provide the basic functionality that made them attractive when current. A mitigation technique for this scenario is a sliding support window that is tightly enforced, cutting off the "very new" and the "reasonably" old from expected support by help desks as well as any management and security features. This will allow for an orderly acceptance of risk and sustainable management.

7.2.2 Form Factor

Much like most devices in this category, they are designed to be portable, and in most cases small. This lends itself to some creative engineering, anywhere from dealing with the internal issues such as the motherboard layout, heat dissipation, power consumption, durability of the case and screen, resolution of the screen, input methods and any exposed external ports, buttons and switches. In all, there are definitely compromises that are made in order to deliver a usable product (and sometimes not) to market and to get all the functionality promised in a small device. This set of design tradeoffs could lead to the utilization of chips and other components that may be less than ideal or reduce the build quality of the device (also of a concern given short product cycles).

The size, as anybody who's carried around a cell phone over the years, is the perfect dimensions for getting lost or stolen. Unlike a computer, desktop or laptop, it is hard to physically secure such devices and a significant responsibility is placed on the user to maintain a level of physical security with the device. As this device is designed to be portable and mobile, there is a higher probability of this occurring than with a comparable computing device.

7.2.3 Durability

Noted previously, with the need for a small form-factor, some liberties can be taken with the durability of the device. This characteristic is also impacted by the desire for the device to operate in a market price point and limit the cost of materials and manufacturing. The pressure on the device design from this angle leads to, as noted above, some extensive levels of creativity. Cheaper components, weaker cases and connectors, are specifically designed towards quick obsolescence in order to keep up with the engineering cycle and releases of new products; in short, it's a pre-destined revenue feeding cycle.

As an upshot from these issues, a secondary market for the providing of protection, in the form of cases, scratch resistant coatings, and other items have sprung up. Consequentially, some cases come with security enhancing features, and are also increasing the durability (and size in some cases) of these devices by protecting ports and buttons and going as far as to have lock and clips to help better secure the original device.

Sadly, with the size of these devices, and their need for mobility, typical physical security techniques for computing devices, such as cable locks, fail to address the issue. So the main focus in reducing risk, such as loosing data from a broken device, would to be to protect the device from breaking and enhancing the current durability architecture by reinforcing the need for protective cases for device. This reduces the cost of overall ownership, as replacement cost will be reduced, but the data will be in transit and reside (stored) on potentially less equipment, so the risks of data existing on multiple devices needing to be wiped and decommissioned is significantly less.

7.2.4 Power

Unlike the iOS devices from Apple, most of the developers of Android™ hardware devices hail from previous smartphone markets and are accustomed to developing systems that can produce relatively high performance in a small package and consume very little power. An offshoot from this, when coping with their previous market segments, those platform developers also allow for user-serviceable batteries that are sometimes marketed with "extended life" versions that allow for longer "untethered" use. This provides for an extended operational capability without interruption to the user, such as docking and charging may require. Of course, there are non-traditional means of replacing or supplementing power use, such as solar panels for charging and power, as well as other supplemental components to mimic or add to current on-board power supplies, but those are cumbersome except to the truly flexible device owner, and add to more things to loose or break.

However, power management, especially for the processors in these devices is the key to long lasting use, and it is heavily tied to the operating system and running applications. Most Android™ hardware devices are shipped "under-clocked", that is, the devices run their processors at a lower rated cycle that the CPU's maximum potential. As an example, a CPU in these devices are designed to reach a maximum computational cycle of 3.0GHz but actually run at 2.1GHz, requiring less power to operate, but conversely running processes at a slightly slower rate (viewable via the 'voltage' tools in Android, and convertible to Watts based on the CPU specs). This is somewhat unperceivable to the casual user, as most generic applications on Android™ use process scheduling in such a way that any lag in responsiveness is compensated by giving the most forward application in the UI the most computing power (assuming that the user is most likely interacting with it in real time).

The security implications of CPU power and battery power consumption are twofold; as changes in the exhaustion of the stored power could indicate issues on the phone and possibly system compromise, as well as compromise to the device if the CPU is "over-clocked" in order to increase temporary performance. For the former, the increase power consumption could indicate applications and service running on the phone that aren't intended (especially if the device is idle) such as malware and Trojans. The latter event could cause actual destruction or unintended modification/alteration of data on the device, or the destruction of the device itself through the compromise of the CPU. Each of these are triggered by software and require no direct alteration of the hardware within the device, although in certain cases, advanced modifications could be

made to the system boards upon direct interaction. This often is through the creation or removal of "jumper" connections between certain components on the system board.

7.2.5 Vendor / Supply Chain

As with most of global trade, most computer equipment is fabricated in Eastern Asia, and most notably China. In this case, there is very little an organization can do, especially if they are concerned about security, to ensure what is operating on an Android™ hardware platform short of a tear-down by the NSA. An organization must accept a reasonable amount of risk due to this factor, and understand that due to diversity of suppliers, the likelihood that a device has been compromised or contains materials that could potentially compromise sensitive data is an extremely high possibility.

Within the supply chain, once a physical manufacturer has shipped the product to the distributor (since the fabrication often does not take place in facilities owned by the "distributor" or "nameplate" company), the distributor, such as HTC or Samsung will prep the devices for sale to a service provider or other sales distribution channel. This often includes packaging, both with what the consumer sees in the way of boxes, documentation and other included components, but also usually involves unit testing and installation of the distributors version of the operating system (if not already contracted to the fabrication plant) and applications. If shipped to a service provider, such as AT&T or Verizon, the devices will be modified (OS and application installation or reconfiguration) to be compliant with their network and system, as well as any services advertised to consumers that must be activated on the device. At all points, the device and underlying system has been modified away from Google's reference platform, possibly introducing unexpected run conditions, vulnerabilities, or in some cases, purposely installing older software that does not contain current patches and fixes.

7.2.6 Architecture

During the evolution of the Android™ OS, the CPU that powers the hardware platform has evolved right along with it. Initially, it was based on a reference ARM architecture (which powered the early Apple Newton tablets nearly a decade previously) and now has branched to support ARM, MIPS and x86 platforms as Android™ OS has moved from smartphones to tablets and to entertainment devices (such as Google TV). This architecture, ARM, is a 32-bit RISC (reduced instruction set) architecture specifically designed for low power applications seen in the mobile environment. This is where Android™ OS powered devices share a lineage with iOS devices, as both draw from the ARM architecture with the most current being the Snapdragon and A5 chips respectively. To consider that the code for each is slightly interchangeable, there was a project to port Android™ Gingerbread to early model iPhones utilizing iBoot, which worked quite successfully, albeit somewhat incomplete.

The processors used in current devices are usually based on the Qualcomm Snapdragon processor utilizing the ARM v7 ISA, which are "system on chip"[xi] (SoC) designs. This SoC design essentially includes an entire hardware operating board on a single chip, often including the digital and analog signal processors, memory components, power management and regulation units, timing sources, and peripheral interface controllers, besides the core processors

(of which, some are multi-core). The fabrication of these designs also affect the potential recoverability of design issues discovered after release or vulnerabilities created during the application of these chips in a hardware system. If integrated as a FPGA (field programmable gate array) design, changes and modifications can be made in the field, thus offering flexibility and a greater chance of recovering from vulnerable design flaws.

Most major changes and/or fixes with any architectural flaws occur with mid-cycle device revisions (or "point releases") that are generally undetectable by end-users (they are slipped into the supply chain without much fanfare). This particular issue may only be of concern when trying to look at cluster events of issues or incidents reported to help desks that could be attributed to hardware flaws. Other cases may arise when applications, some programmed around known issues, fail to work properly on revised hardware. In these cases, it's important to examine the software development cycle and techniques to ensure that "one offs" or platform specific tweaks are not standard practice, such as creating work-arounds for graphic display issues and UI problems (such as pixel density).

Some platforms, such as eReaders, are more susceptible to these architectural issues[3]. In most cases they have a foot in both models; Apple's closed platform (hardware/applications), and those open models favored by the Android™ camp (OS), a term some would apply of "clopen"[4]. While many devices rely on Android™ to power them, and have a lot of the useful features of a full-blown tablet, but remain "crippled" by the vendor (such as a Barnes and Noble, Amazon, etc.), and often suffer from a poor support model for the operation of the device as compared to something provided by a major cellular service provider like Verizon, Sprint, T-Mobile or AT&T. When these devices are jailbroken/rooted, done by users to benefit from the subsidized nature of the device and data connectivity, they become more rogue to management and support.

7.3 Service Provision/Delivery

7.3.1 Device Management

Mobile devices, based on their use model, hardware and software platform management techniques, technology, and policies vary greatly from what was traditionally used for desktops and servers. Typically a mobile device management (MDM) solution is employed to provide basic provisioning and monitoring capabilities, but most lack sophistication enough to account for some of the more specific device and system settings. This lack of detail, and the developers own tools lacking the level of specificity to modify and monitor activity on mobile devices, increase the risk of these devices potentially being configured and used in a manner that is against policy.

Since mobile devices, by nature, will most likely operate off ***** internal network, the typical techniques for monitoring and safeguarding data in transit differ. To equate the access model for these devices to those of ***** internal network-based desktops and laptops, each mobile

[3] http://intrepidusgroup.com/insight/2012/01/android-backdoor-fail-the-kindle-fire-easter-egg/

[4] http://changeorder.typepad.com/weblog/2008/03/sorry-were-clop.html

platform would have to backhaul all of its data, through the ***** internal network over a VPN to ensure that required monitoring continues. This is a requirement of FISMA, and unfortunately, the reporting requirements and solutions for addressing the mobile environment haven't been completely addressed. This is a concern regarding practicality and capacity, as the move towards a more mobile preferred workforce is emphasized by the Executive Office of the President and OMB, a bolstering of remote access capabilities will need be examined and implemented.

The verbosity for device and application on logging mobile devices directly impacts battery life, CPU and use of the radio capabilities (bandwidth). A balance must be struck between requirements and the satisfactory and acceptable performance limits of the mobile device as perceived by the end-user. Compromises can be obtained based on the sensitivity of the data, systems being accessed or the user accessing them, and logging classifications developed, but it would possibly not work well (or at all) within the capability of the MDM tools.

7.3.2 Data Management

One of the largest challenges, with the leveraging of a mobile device environment architecture within an organization, is data management. This involves knowing what information is permitted to be accessed by the device (and subsequent user), how the data will or can be modified in that state, and if any of that data will stay resident on the device. This requires an in depth knowledge and understanding of your user-base, devices accessing the network and data, network and systems architecture, and of course, the data itself (classifications, sizes, types, access methods, etc.). The most often and obvious way of securing data is to require some level of role-based authentication and validation, encrypted transport and storage of the data while being utilized by a user and device. However, most mobile platforms don't support local data encryption and transport encryption "out of the box", and in some cases, what's available is currently faulty or vulnerable to simple compromise.

The leveraging of an MDM solution for Android™ devices should include some level of data access control policies. In the case where that doesn't exist, or allows for a satisfactory level of granularity and protection, this access policy should be enforced at the application level and restrict display and exporting of data to these devices that do not meet security criteria. Unlike iOS, Android™ OS developers are less encumbered by the license and access model, and a number of third-party applications are available (succeeding to differing degrees) to protect data in these states on supported devices. This includes encryption, compartmentalization (storage), access control, anti-virus/malware, and in-program use[5].

There should be in place, the same controls for granting role-based access to classified (by sensitivity, availability, audience, etc.) data and systems that are used for typical desktop-oriented systems and scenarios. This is to say, the leveraging of a data management tools and processes should be universal and be consistent for how data is transferred, handled or modified, and stored from platform to platform, application to application, user to user. This is a tall order

[5] http://developer.android.com/guide/topics/security/security.html

given the nascent Android™ security-focused developer community, which is mainly supported at this time by a few of the traditional desktop security companies, all with limited offerings.

7.3.3 User Provisioning

Along with device provisioning, provisioning users, and their profiles, to mobile devices remain a challenge. Unlike moving users from one Windows desktop to another, mobile platforms vary between many OS developers, and what's required for mobile device profiles differ greatly from those for desktops, including how information is stored in common directory systems such as Active Directory. In some cases, such as those with iOS, Apple recommends creating and attaching an Open Directory server, which can maintain iOS-specific profile and configuration information, while being tied to a master Active Directory for initial pass-through user authentication. This obviously creates more points of failure, and when you work to extend to platforms such as Android, similar situations can exist unless native directory extensions can be created and maintained on the core directory service to supply mobile devices with their required platform information for users.

However, unlike some desktop computing scenarios, mobile devices are more likely to remain linked to a single user due to the use model. This makes developing the provisioning process less onerous due to possibly the one-time nature of setup, but still contains issues when maintaining shared profile information synchronization (such as bookmarks, e-mail, and files) to their desktop profiles. In some cases, where it is difficult to maintain these states (either due to technical, logistical, policy, or financial issues) it should be maintained separately from the general directory and user management services. Some MDM solutions may provide capabilities to provision users via the device profile manager, but this could potentially lock a user to a particular device configuration, eschewing additions and changed made by the user within applications, of which native user profiles usually account for.

7.3.4 Vendor Modifications

If a device is acquired through a data service provider, it has had an operating system modification that differs from the approved Google release, already on top of the modifications developed for the device by the manufacturer. This makes tracking differences, and in turn, features and settings difficult depending on where Android™ devices are sourced. In fact, you can acquire the same device model (albeit with slight branding modifications) from two different service providers and be presented with Android™ devices that operate in entirely different ways. What attracts device developers and, conversely, service providers to Android, its openness and customizability, also creates the greatest maintenance and upkeep problems.

For certain vendors, such as HTC, they create entirely a new UI overlay on top of Android, in this case "Sense"[6], effectively creating an abstract. If the lower OS is patched, modified, or configured in a way that does not conform to this overlay, the phone could begin to malfunction or stop working altogether[7]. This was developed by HTC in order to unify the OS platforms they

[6] http://www.briancee.com/htcsense/what-is-htc-sense/1249/

[7] http://en.wikipedia.org/wiki/HTC_Sense

support (Android, Brew and Windows Mobile) to a single user facing experience, and be agnostic to the technology. However, like the Android™ OS below, the Sense interface requires OTA or out of band (OOB) updates to allow it to maintain security, functionality, and compatibility as time goes on. The addition of an OS overlay such as this also introduces vendor specific applications, widgets and other features, which aren't directly supported by Android™ and Google. This becomes an issue for MDM solutions, when certain UI and other policy components will require a specialized profile and evaluation of settings in order to work in accordance to [redacted] policies.

In certain cases, based on the network data access provider, a particular model of the Android™ device has been customized to work within the peculiarities of the provider's network and service system. In some cases, it's a different radio system to work on the network (GSM vs. CDMA) or slightly different CPUs or graphics chips due to system incompatibility due to the service provider's requirements. This causes further issues when trying to maintain consistency among "fleet maintenance" but could create issues if [redacted] offers to provide support for devices, and in some cases, may be required to due to service area coverage by carriers (such as regional-only providers). Cellular data providers do not guarantee transport security, and as such, any data transmitted should be considered as if it were traversing a public and open network. This is especially relevant to the OTA provisioning process (such as initial profiles, PKI certificates, keys, etc.) as sensitive private and cryptographic material during the establishment and setup phase can occur.

7.3.5 Types of Service

In most cases, Android™ devices are intended to use wireless (RF) communications for data and/or voice transmission if used in a portable form factor (versus control panel/device installs). The options, normally would include 802.11a/b/c/n and cellular data, but could also involve near-field communication (NFC), ZigBee and Bluetooth. In most cases the former requires somebody to provide a base service, such as DOI for 802.11 and a cell phone service from AT&T, Verizon, T-Mobile or Sprint; and in each case there are costs and security concerns in operating both. For 802.11 (discussed further, below) in a non-public setting, access control can be strictly enforced, quality of service (QoS) can be monitored and maintained, and coverage can be architected to provide adequate service where the demand exists. For cellular data, the architecture and quality of service reside in the provider, such is the case in which certain areas may not have coverage, and others may be oversubscribed due to population and use density. When those public services are contracted for, especially for popular technologies such as 3G/4G data services, if business critical applications are intended to be utilized over them, language to ensure QoS and coverage should be included.

As noted about differing types of service, as well as how they are provided, one should include a concern for application and service developers to ensure that their systems adequately perform, and remain secure, when utilizing these networks, as opposed to assumptions that are made in a wired environment. This requires that application developers within [redacted] utilize strong authentication and encrypt the transport of data from the application to the device and assume that the connection is unencrypted. For those same developers, the leveraging of a reduced, slim,

or lightweight UI will work well on data connections on which they have control over. Insuring proper data access controls on Android™ devices could also be enabled through a custom developed application designed and implemented to ***** specifications and existing policies.

7.3.6 Public WiFi/Data Networks

Unless the Android™ device is "personally owned equipment" (POE), policies similar to those for USG laptops in regards to managing access to public-WiFi and data networks (such as for telework) should be adapted to address these mobile scenarios. Unlike Windows-based computers, the level of control in restricting connections to trusted networks may be somewhat limited, either due to the inherent lack of configurability in a “sticky” preference on the device or MDM solution[8].

As noted in a previous section regarding service provision, public networks are the ones that involve the most risk, as there is no legal agreement or understanding on how your device, the data, and communications will be treated while utilizing the service. Utilization of a TLS tunnel or IPSec VPN connection should be utilized, as authentication and encryption only exists between the device and base station and not to the destination endpoint. This can be established by the application being accessed (i.e., HTTPS), via an on-device application, or enforced by a connection policy (written or technical implementation).

Other alternatives to utilization of semi-public access points, such as those provided by network service companies, like AT&T, Verizon and other WiFi resellers, is to establish a service and license relationship that utilizes a connection client that will contain account and connection parameters, and automatically create a secure conduit to the ***** network. This will allow the establishment of some level of trust through legal service provision arrangements and significantly reduces the operational risk and exposure surface when utilizing semi-public services.

7.3.7 Other Services

In most cases, mobile devices, such as the subject Android™ products, will be utilized off of the ***** internal network, and will utilize remote access services and capabilities in order to perform activities involving ***** information systems and services. In many cases these mobile devices will be personally owned equipment (POE) and will need to be treated as untrusted “dirty” devices and require both device and user-level verification and authorization before accessing ***** resources. Some of these concerns can be addressed through the use of user-centric PIV credentials, but also include trust-based solutions for devices, which include PKI certificates for devices, providing access based upon those device trusts and provisioning level. Technical complexities remain, as most mobile devices do not have support (onboard) for hard token/card inputs and will require soft-tokens or certificates in order to function in a required fashion for FICAM compliance.

[8] http://developer.android.com/guide/topics/admin/device-admin.html

7.3.8 Upgrades

Since the product cycle is much quicker for Android™ devices, as well as the field of manufacturers being large compared to competitors such as Apple and RIM, getting lost in the race to maintain the latest-and-greatest device support, as well as ensuring the current profiles and configuration guidelines can effectively address the multitudes of devices, a plan for supporting upgrades and releasing support for older devices should be developed. It should be pertinent to publish a list or matrix of supported devices by the Office of CIO (OCIO) to the staff and [redacted] in order to raise awareness and to inform those who may be handling acquisition decisions. Again, this list will need to be periodically maintained and back-end components (profiles, MDM, etc.) incremented to address any needed changes and modifications to support new devices in-line with OCIO operational policies.

The upgrade concern is also present with major releases and "point" or "dot" releases of the various versions of the Android™ OS for each device. For those part of a data service contract, those are managed by the service provider and delivered OTA (over the air) and in many cases are staggered and not synchronized. Those providers, very infrequently provide updates such as this as a downloadable, and thus, centrally distributable package for an organization. This, in effect can leave large populations of devices having an unbalanced upgrade schedule, and, in some cases, no upgrades at all if they remain out of a service area for long enough. As noted earlier, the data service providers lag significantly behind in providing updates after Google officially releases their version, and managing and anticipating these updates and upgrades very difficult, unless you track mobile news sites and discussions in the developer community. OTA and off-cycle upgrades of the software can cause issues with applications not coded to new Android™ APIs, or rendered to new specifications, or in some cases, result in data loss if a procedure for maintaining user state is not maintained. [redacted] divisions should develop policies and procedures regarding these specific issues.

8 Mitigation

In discussing the various risks in the operation of Android™-based platforms previously in this document, several mitigating techniques and methodologies were explored, but would require adherence to strict and complete implementation in order to achieve any level of success. These techniques will reduce the risk element to operation, but may also increase operating and administrative costs. Upon ***** discussions and an evaluation of [redacted], many of these mitigating and compensating modifications that must be made to allow Android™ device to operate securely are [redacted], and that some issues remain even after low level device provisioning has been completed (rooting the device, modifying low level OS files).

Other, albeit more invasive, methods for monitoring device activities could include options such as forced VPN and web proxying, live logging, and the use of the SEAndroid[xii] (from NSA) distributions in order to maintain a real-time knowledge that the content going to and from the device is adhering to ***** policies. These could reduce the overall functionality of the device by restricting application and system components, shortening battery life, and increasing cellular and other data network usage. In some cases, special releases of the VPN software for various

Android™ device manufacturers are required, putting the OS fracturing concern higher up on the issues list.

Another, more restrictive, option for Android™ use within ***** would be to disable all networking (WiFi, Bluetooth and cellular data) on the device and require all data transport to occur over a USB tethered connection. While this is not optimal, it reduces the attack surface of the device quite significantly and ensures that data is truly shared only between USG devices. This may not be optimal for mobility requirements and real-time communication, but flaws within the platform, available security controls, and lack of established universal secure communication links (including those for internal ***** applications) preclude allowing these devices from running in anything other than this mode.

Finally, there is little to no mitigation (short of blocking via web proxy) to prevent the installation of applications from the Android™ Market (now "Google Play"), which has a very low barrier to allowing any application to be "sold", all of which has led to a prevalence of malware and other less-than-trustworthy software, including fake versions (or knock offs) or popular games and applications. The only capability available to remove this software once installed is using the "kill switch" Google has available that has been built into the system, and that still requires notification to Google and providing proof that the application is malicious or performs in a less-than-acceptable manner. Still, the removal of such software, even with MDM and Google's own tools, is hit and miss, since much of this requires that these devices are reachable via those tools and reside on a network or are connected in a manner that allows for these controls to get through to the end device. As noted with most malware of desktops, these tools will use a self-protection capability to maintain a foothold and resist detection and removal.

9 Use and Deployment

Given the highlighted issues, the deployment of Android™ OS and associated devices should be done very carefully, or not at all due to the security concerns. In the appendix below, serious flaws exist in the core underpinnings of the security infrastructure of the device, serious enough to warrant a total lack of trust in the security model. Until those flaws are fixed or patched, the trust model between application, OS and device are flawed and can compromise the data and the lower-level functions of the device.

Under examination, with this core issue in the operation of these devices, combined with the slow upgrade cycle from vendors to their devices, issues with the security of applications acquired from the Android™ Market, no hard-token (PIV) authentication, [redacted] It is the author's recommendation that Android™ device not be permitted for general use within *****.

10 Summary

Although Android™ OS and supported devices grow in market share, the immaturity and lack of enterprise-grade management, [redacted], and serious security flaws do not provide a trusted platform that should be utilized. In order to secure the platform to minimal standards would incur significant cost of financial, staff, and technology resources in

order to adhere to a level of security expected by NIST and OMB standards for system security. Even with attempting to work around known issues with these devices, there are many more hidden that could be exploited quickly with no mitigation tools to remedy the flaw, and with the fractured Android™ OS support from vendors and service providers, there would always be a number of devices (perhaps large, depending on adoption) that would be operating at a serious disadvantage.

11 Appendix

11.1 Observed Issues – Motorola Xoom

The following observations were made during an initial surface analysis of the operation of the Android™ OS 4.0 (Ice Cream Sandwich)-based WiFi tablet. The initial data was gathered from a device previously configured, but wiped by [redacted]. Originally this device was delivered with Honeycomb, which is Android™ 3.0, but has since been removed via an upgrade.

11.1.1 Boot / Setup

During the initial boot and setup phase from a stock configuration, the Android™ tablet asks for Google Account information in order to customize the tablet. This has the effect of configuring the tablet automatically for Gmail, YouTube and a number of other Google services, but also enables the backup of the configuration and settings (and some application data) to Google's servers. In order to operate location-aware services on the device (such as maps, hot-spot finders, etc.) a separate screen asks whether or not to enable Google Location, which has a double edged sword of allowing for location tracking, but also is a two-way data exchange, sharing performance and that same location data with Google. At this time, there's no concrete accounting for which information is shared back to Google.

Networking is extremely important for these devices, as noted above, as the wireless aspect of this capability allows these devices to be highly mobile. At initial setup, the device wants activate all and any broadband data services, such as 802.11 WiFi and 3G/4G cellular. Thankfully, by default, the less secure Bluetooth stack is left inactivated until done so directly by the user. To note, the device running Android™ OS 4.0 will support 802.1x authentication and has the capability to load soft user certificated from an installed SD card (an issue regarding the security of removable storage also arises). However, curiously, there is also a data metering setting within the control panel, and it's unclear if that information stays local or is also shared with Google for usage statistics.

11.1.2 Device Security

The most important component of such a mobile device is how secure can it be made to reasonably protect the regular operation of it, including transmitting, manipulating, and storing data. For users, the first interaction would be with the device's "lock screen", which can be enabled (or disabled) and include just sliding open, PIN, password, or pattern. The device does not directly support the reading of hard tokens such as a PIV card, but soft certificates (for users or specific applications) can be loaded and accessed via an SD card. These access components

can be used to aid in the encryption of all the data on the device. The minimal requirement for encryption requires a PIN (length unenforced) and the encryption algorithm is unknown or selectable to the user.

Different type of users and access levels based on pre-defined templates are available on the device, however, they do not directly map to those users who use the device, but to elevated privileges granted to applications, such as 3rd party VPNs. Most of all the applications and processes run in some context of the device user, and that in itself allows for very expansive access to manipulate processes and data on the device. Restrictions can also be implemented to prevent the non-Google Play/ Android™ Market applications from being loaded OTA on the device. There are a number of default applications loaded on the device, and removing them would either be done via a custom ROM image for the device or the manual removal during device provisioning by IT staff. A thorough evaluation of the security (and personal data use) of these applications should be completed before approval of the use of these devices.

Other information disclosure settings can be adjusted in order to obfuscate user information, such as "visual" passwords at the login screen, as well as removing the display of ownership information at that same screen. A troublesome issue, which would allow for easy access to all data on the device, regardless of encryption or a network connection, is the use of the "Developer" menu and the ability to debug via USB. Beyond allowing access via Android™ SDK tools, remote shells can also be implemented via the simple tethering mechanism. This allows complete and unfettered access to the device and data at incredibly low level, no jailbreaking or "hacking" required. There are also hardware debug button combinations that can enable modes that allow access and installation of information without an USB cable (via an SD card) including custom firmware.

Other concerns include a similar "voice command" feature to that which exists in the latest version of iOS. The most disturbing discovery during initial research is the ability for an application to be purchased on Google Play, and if the account has an Android™ device associated with the profile of the user, to remotely install the application without any notice being displayed on the target device, in short a silent install. This is of concern given to high incidence of account compromise at Google (among other webmail providers), Google recent consolidation of the Google Accounts and data, and the known prevalence of malware and other code in Google Play that could be forcibly installed by somebody who only needs to compromise a web-based account and does not need physical access to the device. Upon examining Google Play, it is very difficult to determine the validity and lineage of the applications, given Google's democratized model of selling applications on their platform. Unless the "known good vendor" polices their application on Google Play, there is a chance for the legitimate application to be spoofed for a non-legitimate application, and unwittingly get installed by a use, via the device or web-based storefront. At this point, from an enterprise standpoint, very little to be done to prevent exploits and misuse such as this, even with a policy enforcement device such as an MDM.

11.1.3 Application Security Settings

Overall, the OS is just a component that enables the device to perform functions the user requires of it, and as seen with Apple's iOS, the real driver of the use of mobile devices is the "App" environment. Between the two device operating systems, the control of data, system function, and device component access and control are the major differences in their security model. With the Apple environment, iOS has very clearly defined and exposed APIs and functions to which a developer has access to. This effectively then, requires the OS to "lint"[9] in runtime to check for suspicious calls and activities, but in a predefined interface. With Android, each application can ask for a multitude of permissions from the user (and via the UI, even on the tablet, that list can stretch beyond what is visible, and many requests can be overlooked or missed, and insecure permissions granted), and result in combinations that could grant overzealous access to the system, or create run conditions that create an inherent instability to operation.

These user-dependent choices can leave devices in very different states from person to person, and due to the lack of granularity among most major MDM solutions in regards to what access is managed by policy, actual managed security is virtually impossible. Most end-users lack the knowledge of exactly what they may be granting access to and are driven to have the functionality given to them by installing an application rather than considering how that app handles their stored, transmitted and manipulated data. There currently exists no mechanism to actually prevent certain permissions from being granted by a user as a blanket policy control, and even if it was allowed, it would instantaneously break the model Android™ uses to allow applications to work, resulting in a number of support requests, or even the abandonment of these devices by users due to frustration. This is a core security model that is not likely to change with Google and their Android™ development roadmap and may be addressed by third party security companies. This is a distinct possibility given the fact that, unlike Apple, lower level system access isn't inherently restricted and AV and anti-malware applications can be developed that access kernel level processes.

11.2 Observed Issues – HTC Vivid

11.2.1 Device Security

As with the security concerns listed previously in this document, the following details some device specific security concerns for Android on HTC compatible devices. As seen in the past issuance of vulnerability announcements, the use of HTC Sense has introduced security issues not already present in Android, and potentially opening up other vectors, albeit only in the population of those using HTC devices[10]. However, the device manufacturer was very quick to acknowledge and respond to the vulnerability (about 30 days), and indicates a dedication and

[9] http://en.wikipedia.org/wiki/Lint_(software)

[10] http://www.androidpolice.com/2011/10/01/massive-security-vulnerability-in-htc-android-devices-evo-3d-4g-thunderbolt-others-exposes-phone-numbers-gps-sms-emails-addresses-much-more/

attention to security. Another issue cropped up specifically regarding the password handling for 802.11x wireless security[11] in September 2011[12] that resulted in information disclosure (and potential for complete system compromise), and varied effectiveness among different builds of Android on HTC devices. This, unlike the logging issue, actually is of concern due to the specialized build required for these devices that is a central component of communications security, and took until February 2012 for a fix by HTC. Other issues for this device actually revolve around the detailed customizations that are allowed for devices in this class, which may also include custom hardware features not present in other manufacturers' devices or different ways of implementing common features.

Most of the issues can be discovered through the "Settings" widget, which collects and provides a single point of access to review, examine, use/set lower level features on the device. The major issues revolve around applications, network access and use, developer controls, and user accounts, which differ slightly from other reviewed devices. Apps, Widgets, and other components, when accessed from the Settings area, show no indication as to where the sources are from until you create or add a Google Account, then only UI components, such as themes, sounds and background pictures are routed through the HTC Hub without an account. This can lead to potential malware entering in from unknown sources if the requests were hijacked or intercepted, which is impossible to detect from the user interface.

Like the Motorola, the HTC Vivid has support for the use of an SD card for expanding the storage capacity of the device, as well as loading and accessing data without utilizing wireless data networks. This also enables the installation of applications and custom ROM images, as noted above, potentially displacing the approved methods of managing applications and data on the device. On the flip side, this same SD method can be used to load "soft" certificates to enable PKI and two-factor authentication, as well providing, if suitable media is used, a read-only environment for things such as a "thin client" for remote network or VM access.

For VPN, the use of Junos Pulse is still required, and upon initial install, the acceptance of the provided encryption certificate is requested of the user. If a VPN connection is closed on a desktop, the Junos Pulse application will detect it and attempt to hand over the tunnel to the device. At this time, and through testing, the Junos client does not support IPSec directly, and defaults to a SSL/TLS tunnel.

11.2.2 HTC Sense GUI Overlay

The HTC Sense GUI system, developed by HTC to unify the user experience on multi-OS devices, is based upon the TouchFLO 3D framework[13], a carryover from HTC's Windows Mobile development. This not only is an alteration to the standard Google Android UI experience, but several other "components" and features have been added upon that stock

[11] http://www.htc.com/www/help/wifi-security-fix/

[12] http://blog.mywarwithentropy.com/2012/02/8021x-password-exploit-on-many-htc.html

[13] http://en.wikipedia.org/wiki/TouchFLO_3D

distribution[14]. Many of these applications are network-aware, and require active network access in order to sync settings and update data. Some of the other features contain tracking features, marketed as location-aware, that may not be suitable for use in certain operating environments (such as "Footprints"). HTC provides their own purchasing portal for applications and UI enhancements (sounds, wallpapers, widgets, etc.) through HTC Hub without going through the Google supplied stores (Amazon also provides a non-Google Android application store as well in this case), and separate evaluations of terms and conditions, as well as any legal indemnification implied, by using non-Google managed delivery systems. Upon enabling the Google Account on this device, a user will still be prompted to add mail services, as this initial linking enables most Google services, including syncing, but not mail. However, this enablement will allow those services requesting access to Android Marketplace/Google Play, prompt free access after the initial terms of service are agreed to, which is global for all Google supported services (YouTube, etc.).

11.2.3 Carrier Specific Alterations (AT&T)

Unexpectedly, upon receiving the device for evaluation, many of the changes from a stock Android ICS release were rolled up into the HTC Sense components. The device itself has an AT&T hardware branding (logo on the top bezel) and several AT&T specific applications short-cuts pre-installed, which are maintained in the ROM on the device, that are re-installed upon device reset. These applications include AT&T Code Scanner (QR Code reader), AT&T Family Map, LiveTV (AT&T U-verse), AT&T Navigator (Navigation, Guidance and Road Mapping), AT&T Visual Voicemail, and myAT&T (Customer Service Application). Clicking on the icons opens the Google Play/Android Marketplace, requesting and activation or linking of an existing Google account. AT&T Navigator is the only application actually pre-installed, and requires acceptance of a "Terms of Service" for additional billing, and would be recommended that this be removed from the device before distributing to users.

One of the more interesting "bugs" with the AT&T distributed phones is the account configurations and the protection that AT&T doesn't provide. Initially upon receipt of this device, there was at least one SMS spam message received, and another one received during initial testing, all from a brand new device. Since SMS is an add-on feature to a lot of plans, this could potentially result in unintended charges to a customer's account.

Finally, the last AT&T specific setting noticed on the device is in the "Settings" area in which all software updates for the device (Operating System) are gathered and installed from AT&T via the "System -> AT&T Software Update" control panel option. This assumes that all major and "point" releases will be vetted and tested by AT&T before distribution to customer handsets. However, much like the Xoom tablet, forced out-of-band (OOB) updates can be done via SD card ROM installation.

[14] http://en.wikipedia.org/wiki/HTC_Sense#HTC_Sense_on_Android_Devices

11.3 Recent Exploits

Recently, at ShmooCon[xiii] in Washington, DC (January 27-29, 2012), an issue relating to the sustainability and security of the Android™ PKI infrastructure was released. This pertained to the bypass of X509 certificate checking and signing of applications. This discovery involved de-compilation of the application, replacement of text within the application for verifying the certificate, and poor checks from within Android™ OS of the certificate signing chain. Due to the process involved, a tool was released that automated this activity to a point and click interface, making it possible for untrained users or hackers to perform this bypass. Google has been notified by the researcher, but fixes for this issue have yet to be released. As this is part of the core security infrastructure of the operating system, until this issue is fixed, it is recommended to not allow use of Android™ devices within DOI at this time.

"*A Blackhat's Tool Chest: How We Tear Into That Little Green Man*"

Video: http://www.shmoocon.org/2012/videos/BlackHatsToolChest.m4v

Slides: (see attached PDF) [original URL – in Apple Keynote format]
https://docs.google.com/uc?export=download&confirm=no_antivirus&id=0B-__-viMlkWLYWZiYzk1ZjgtODRkYS00YmFmLTgxMmYtNGU0MzNkZGQ0OWQw

A similar "man in the middle" (MitM) certificate attack (via network connectivity, rather than "on device" modification) was described by the Intrepidus Group as well; further weakening the PKI infrastructure for the device and applications on Android. Combine this, with recent compromises of major certificate authorities (CAs) such as DigiNotar and others, and you have a recipe for disaster. Happily this is on Ice Cream Sandwich (Android™ OS 4.0), which is the newest incarnation of Android™, so the delayed update and fracturing of the platform actually assists, somewhat in mitigation. It is, still, however, a major flaw.

"*Man-in-the-Middle (MiTM) and certificate setup on Androidtm4.0*"

Link : http://intrepidusgroup.com/insight/2011/12/mit/

Other recent major Android™ vulnerabilities also revolve around privilege escalation and security control bypassing, especially prevalent with user-installed applications. The security model is greatly exploitable due to Android's model of allowing and requiring users to make device security decisions. Unfortunately, the current MDM solution does not prevent or assist in these decision-making cases.

"*Systematic Detection of Capability Leaks in Stock* Android™ *Smartphones*"

Paper : http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_WOODPECKER.pdf

Recently, researchers at North Carolina State University have found significant holes in the security of the in-application advertising framework in use with Android™ applications. This is due to the lax control on what advertising has access through the application, which in itself is usually granted more extensive permissions than what is really needed to run. This results in the leakage of personal information and data through collection and profiling methods in the advertising framework.

"*Unsafe Exposure Analysis of Mobile In-App Advertisements*"

Paper : http://www.csc.ncsu.edu/faculty/jiang/pubs/WISEC12_ADRISK.pdf

Security firm, Symantec, has warned of discovering polymorphic malware targeting the Android™ platform, which changes on every download. As with most recent malware attempts, this begins with an SMS (spoofed) offering download of a compromised piece of code that exhibits this behavior[15]. While this compromise is more reliant upon social engineering, it speaks to the ease of being able to lure a user to install an unauthorized piece of code on these devices outside of the Google Play/Android Market environment.

"Server-side Polymorphic Android Applications"

Link: http://www.symantec.com/connect/blogs/server-side-polymorphic-android-applications

11.4 Android 4.x Version Updates

11.4.1 Jelly Bean - Version 4.1/4.2 (API Level 16/17)

11.4.1.1 v4.1 API Level 16

Due to multiple issues, namely security, direct support for Adobe Flash ended with switch to Jelly Bean, this also comes on the heels of Adobe itself dropping direct Linux support for the format. This, observably, has a major positive impact on system security, but conversely removes the claim of Android (especially in marketing materials) of supporting Flash over Apple's iOS. For most newer platforms, the push for supporting HTML 5 is primary in their web development roadmaps. To that end, this version also marks the replacement of the Android browser with a completely mobile version of the Chrome browser, and in turn all which the desktop edition has as major advancements now make their way into the Android OS development tree. However this browser replacement only arrives on systems that shipped with a stock Android 4.1 base OS install. The new "Android Beam" functionality[16], primarily aimed at utilizing the NFC capabilities of newer Android hardware devices[17], will require updated policy controls within *****'s MDM system if supported. Direct camera access from the device lockscreen is now supported, however, the use may introduce similar flaws to the operation of the devices and OS similar to issues regularly discovered in iOS once this feature was adjusted. Similar capabilities for the user also allow the addition of application widgets to the lockscreen for quick access, which also will require policy adaptations if the MDM allows, since this application access diesn't require the phone to be unlocked for access.

All available vulnerabilities publically disclosed for version 4.1.x center around "Qualcomm Innovation Center (QuIC)" components, with two of three vulnerabilities scoring CvSS scores of 7.5 and non-authenticated trivial exploits. However, exploiting OS-level issues, at least through previously traditional

[15] http://nakedsecurity.sophos.com/2012/04/18/fake-instagram-app-android-malwar/

[16] http://www.android.com/about/jelly-bean/

[17] http://thenextweb.com/google/2012/09/19/security-researchers-hack-android-via-nfc-samsung-galaxy-s-iii/

means, will be much more difficult due to full DEP (data execution protection) and ASLR (address space layout randomization), common in desktop operating systems, but is sparsely used on current mobile offerings[18]. Other issues are hardware dependent (and tied to applications, some not developed by Samsung directly)[19], with six (6) vulnerabilities found in March 2013 for Samsung-branded devices[20], including privilege elevation and denial of service conditions[21]. Other conditions of potential vectors for compromise continue to exist for USSD (unstructured supplementary service data)[22] codes that place the Android device into command modes that and adversely affect operations on the host system[23], while Google was quick to patch this in the branches and via an OTA push to devices, those that rely on carrier specific release may not or won't get this set of patches, and continued issue considering the Android development and distribution model. These can be spread via URLs, NFC or even QR codes since very rarely are the sources checked and verified before acting upon the contents.

11.4.1.2 v4.2 API Level 17

Google fixed a flaw through addressing UI shortcomings by prompting the user for acceptance of premium SMS messages. This often been a source of service theft for numerous carriers and exists on other platforms as well[24]. However, this hasn't addressed all SMS flaws within the most recent release, with researchers able to use SMS as an attack vector to execute commands via elevated privileges on a victim's device. Google also added the SELinux kernel support[25] for hardening the core OS on the device, as well as strict policy support, called Lockdown, for the built-in VPN clients[26] that controls how a device reaches Internet-based services. For tablet-based systems, this API version supports multiple users, similar to a desktop OS, and similarly applied policies based on a per-user basis[27] (8 users total, 3 simultaneous). Finally, some improvements were introduced to the lockscreen controls (including better gesture support), but at the time of this writing, no immediately discernable changes seem to have

[18] http://www.zdnet.com/charlie-miller-difficult-to-write-exploits-for-android-4-1-7000001073/

[19] http://www.infosecurity-magazine.com/view/29910/samsung-galaxy-iii-smartphone-vulnerability-leaves-millions-open-to-malware/

[20] http://labs.mwrinfosecurity.com/advisories/2012/09/07/multiple-samsung-android-application-vulnerabilities/

[21] http://www.infosecurity-magazine.com/view/31393/six-fresh-vulnerabilities-found-in-samsung-android-devices/

[22] http://en.wikipedia.org/wiki/Unstructured_Supplementary_Service_Data

[23] http://securitywatch.pcmag.com/none/303186-my-android-device-is-vulnerable-to-a-dirty-ussd-hack-now-what

[24] http://randomthoughts.greyhats.it/2013/03/owning-samsung-phones-for-fun-but-with.html

[25] http://selinuxproject.org/page/SEAndroid

[26] http://www.androidauthority.com/android-4-2-potential-security-features-unveiled-selinux-vpn-lockdown-premium-sms-confirmation-123785/

[27] http://www.android.com/whatsnew/

occurred. Convergence of the privacy settings within Android Jelly Bean now place all system known privacy settings under one panel, versus individual locations, yet unknown is if older applications and system components have to update API use to allow for these settings to be conjoined.

As a part of v4.2 improvements, API additions for "app verification" that interact with the Google Play store and applications to ensure they conform to new security features, such as "always on VPN" noted above, are now available (also for those sideloaded via USB). This should eventually show up as an exposed policy settings to MDM APIs to enable only verified apps to be installed and those communication channels appropriately secured. Along with securing "premium" SMS, noted above, other communication channels are now restricted during device encryption for supported platforms, allowing for securely hardening these devices without worry of in-process data siphoning from previously open network connections. However, to take advantage of the full capabilities of these features, specific Jelly Bean compiled binaries will need to be generated, as a compile-time flag needs to be set (PIE - position independent executable[28]), otherwise those features are not implemented[29]. Finally, an update to allow for "certificate pinning", which "pinned" domains receive a validation failure if the certificate presented does not chain to set of expired certificates, helping identify spurious or falsified certificates from potentially compromised certificate authorities (CA). Still, all these releases do not get Google out of the woods yet, with several researchers actively reporting and tracking known bugs in the core system well past incremental point versions[30].

12 Bibliography

Android™ Developers : http://developer.android.com/

Android™ Market : https://market.android.com/

Google Play : https://play.google.com/

Android Developer Jelly Bean 4.1 and 4.2: http://developer.android.com/about/versions/jelly-bean.html

CNN – Why “Android™ fragmentation’ isn’t so bad” : http://www.cnn.com/2012/02/17/tech/mobile/android-fragmentation-gahran/index.html

ReadWrite Mobile – “[Study] Android™ Fragmentation Not as Bad as You Think” : http://www.readwriteweb.com/mobile/2012/02/study-android-fragmentation-no.php

ReadWrite Mobile – “Poll: What Does Android™ "Clopen" Mean, Really?” : http://www.readwriteweb.com/mobile/2012/01/poll-what-does-android-clopen.php

AppleInsider – “Serious security flaws discovered in Android™ phones, Samsung and HTC ignore issue”:

[28] http://en.wikipedia.org/wiki/Position-independent_code

[29] https://blog.duosecurity.com/2012/07/exploit-mitigations-in-android-jelly-bean-4-1/

[30] http://seclists.org/fulldisclosure/2013/Mar/140

http://www.appleinsider.com/articles/11/12/05/serious_security_flaws_discovered_in_android_phones_samsung_and_htc_ignore_issue.html

ZDNet – "Android™ bloatware results in serious security flaws": http://www.zdnet.com/blog/hardware/android-bloatware-results-in-serious-security-flaws/16853

ComputerWorld – "Google pulls 22 more malicious Android™ apps from Market" : http://www.computerworld.com/s/article/9222595/Google_pulls_22_more_malicious_Android_apps_from_Market?taxonomyId=17&pageNumber=1

Google Play – Junos (Juniper) SSL VPN Client : https://play.google.com/store/apps/details?id=net.juniper.junos.pulse.android&hl=en

Android Security - From Linux to Jelly Bean: http://www.slideshare.net/JungPilChoi/android-security-from-linux-to-jelly-bean

---

[i] Wikipedia – Mobile Operating System : http://en.wikipedia.org/wiki/Mobile_operating_system
[ii] Wikipedia - Real-Time Operating System : http://en.wikipedia.org/wiki/Real-time_operating_system
[iii] Wikipedia – ARM Architecture : http://en.wikipedia.org/wiki/ARM_architecture
[iv] Wikipedia – Embedded System : http://en.wikipedia.org/wiki/Embedded_system
[v] Wikipedia – Bombe : http://en.wikipedia.org/wiki/Bombe
[vi] Wikipedia – Feature Phones : http://en.wikipedia.org/wiki/Feature_phone
[vii] Bit9 – "Ginger What? How Android is Left Orphaned and Alone by Manufacturers" : http://www.bit9.com/blog/2011/11/21/ginger-what-how-android-is-left-orphaned-and-alone-by-manufacturers/
[viii] Wikipedia – CyanogenMod: http://en.wikipedia.org/wiki/CyanogenMod
[ix] Wikipedia – Dalvik VM: http://en.wikipedia.org/wiki/Dalvik_(software)
[x] Wikipedia – Cryptoperiod: http://en.wikipedia.org/wiki/Cryptoperiod
[xi] Wikipedia – System on Chip: http://en.wikipedia.org/wiki/System_on_chip
[xii] Wikipedia – Security Enhanced Linux (SELinux): http://en.wikipedia.org/wiki/Security-Enhanced_Linux
[xiii] ShmooCon: http://www.shmoocon.org/