Securely deploying Android devices

Angel Alonso-Parrizas, Security Engineer, MSc, BSc
GCIH, GCIA, GCFW, GCFA, GSNA, GSEC, CISSP, CISA, CISM
Master In Security and Forensic
School of Computing,
Dublin City University
Dublin, Ireland
Email: parrizas@gmail.com

***Abstract*—Android has become a very popular operating systems for smartphones and tablets but at the same time threats associated to this platform, like malware or exploits, are also growing. During this paper the author will describe how it is possible to improve the security of Android in order so it can be safely used in business where security is a high priority.**

I. Introduction

Nowadays it is necessary for most companies to provide e-mail/Internet access to employees outside of the office, hence many companies provide their staff with BlackBerrys, iPhones, Android or other smartphones with Internet connectivity. But, how can these comply with the company's security policies? How is it possible to provide such functionality without putting the security of the company at risk?. Some vendors such as Apple have released tools [1] which permit a security baseline to be defined for the smartphone, security policies to be managed remotely and the iPhone to be wiped remotely or located through GPS.

Google has released a tool, through Google Apps for business [2], that can implement some security policies in the handset, however, in my opinion, the security baseline that can be set through Google Apps is insufficient as some of the most important risks are not mitigated.

It is very important to take into consideration that a smartphone must implement security controls (such as password protection or auto wipe) in case it is stolen. However, this is not enough. What happens if the mobile is compromised by malware while connecting to the Internet? How can we reduce the risk of this happening? How can we detect it? How safe is it to plug in an external SD card if our company does not allow the plug-in of any external USBs? Is it safe to allow the user to have bluetooth running? These are some examples of the kind of questions we should ask ourselves when permitting the use of smartphones in a company.

A. Security Threats in Android

Android is an Operating System based on Linux so we can say that it has the same functionalities as any desktop running a modern OS with Internet access, but with some additional hardware such as camera, GPS, etc. This means that the same threats that apply to modern OS can be extrapolated to Android, like malware or exploits. However, in the case of smartphones, there is an important difference which can have a high impact: mobility. Mobility means that a static network is not assigned to the device, as in the case of desktops or servers, hence connecting to any networks through 3G/UMTS or WiFi exposes the device to any kind of network attack (sniffing, spoofing, etc). Because the device is not assigned to any fixed network this gives the user some flexibility to move/travel, but it also results in a lack of some security layers of protection such as external firewalls, perimetrical antivirus or IDS/IPS which are found in any business. The article by Bastian Knings, Jens Nickels, and Florian Schaub [3], describes a good example of how it is possible to exploit a vulnerability in Android through a WiFi connection. They explain how an attacker can eavesdrop and access the Google Calendar content and even impersonate the user. Another good example of a tool which takes advantage of LAN/WiFi attacks is Firesheep [4], which can also be run in Android.

Applications in Android can be installed in different ways. The most popular one is the official Google repository, Market. It is also possible to install packages from the shell connected to the USB. Any developer can deploy an application, even malware and distribute it via the repository, as has happened in the past [5]. Some Spanish security researchers introduced malware in Android as PoC in order to demonstrate the lack of security in the Market [6]. In my opinion, this is the most critical part in the security chain. The end user

must never be able to install any application and only authorized software should run in the device. Symantec's report 'The Current State of Mobile Device Security' [7] highlights the same issue.

Camera and GPS are not insecure by default, unless the software running or drivers are vulnerable. The biggest issue with them is the lack of privacy, since GPS can be used to track the device and the photos taken with the camera can be stolen. Bluetooth is a different story because it allows the user to send and receive traffic, hence it can be used to control the device, steal confidential data, etc. Bluetooth is also useful to connect external devices like headphones or to connect to the car handsfree, however the author will not consider those scenarios in this project and bluetooth will be disabled.

Another point to address is the physical security of the device. The same controls applied to laptops should be applied to smartphones: encryption, passwords policies to login, etc, and if possible remote wiping and remote GPS localization. The main problem is that versions of Android below 3.0 do not support encryption by default so it is not possible to encrypt the device itself. We should take into consideration if the SD card attached to the device is secure or not and the risk associated with it.

The latest point to address is how to manage and administer the smartphone from a centralized location. For instance, this will permit the security administrator of the business to install authorized software on the smart phone. By default, Android can be accessed from the shell with the SDK toolkit [8], but this gives the user the possibility of also accessing the device through USB, and this is a security breach of the policies.

B. Android architecture: the security model

'Mobile Security Application' [9] is a good reference book which explains Android architecture (pages 16-47) and other mobile technologies like iPhone.
Android is composed of four layers (see figure 1):

- Application: all applications running (ie: phone, mail, etc).
- Application Framework: provide different packages of service applications.
- Android Runtime and Libraries: contains a core component called the Dalvik virtual machine and each process is executed in a separated instance in the VM. In this layer, there are some libraries like SSL, SQLite or libc.
- Linux Kernel: it abstracts the hardware from the software.

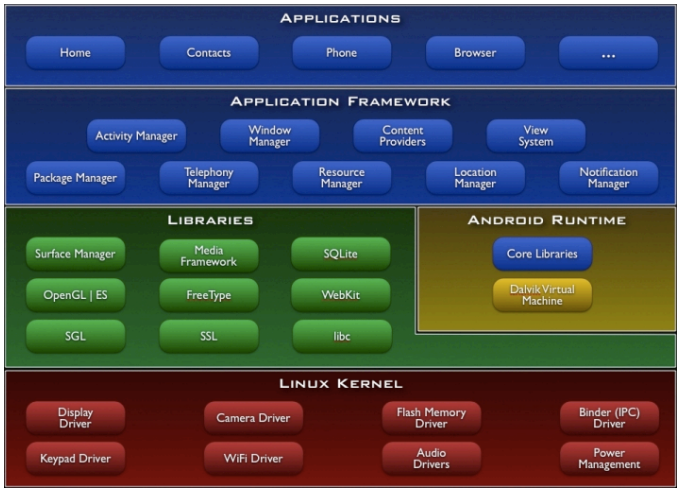

Fig. 1. Android-architecture

Android follows the same idea of user/permissions like a normal Linux system. This is very well explained in Android Developers guide [10], However there are some key differences:

- Android is not a multiuser system unlike traditional Unix/Linux system where multiple external users are connected to the system. However, it uses the concept of UID and GID to assign permissions to each application and process, hence there is isolation between processes/applications.
- Whenever a new application is installed some specific permissions should be allowed. For instance, if the application needs to access the GPS, it will request access to the GPS. However it is the user who installs the application who takes the decision.
- Whenever a new application is installed, specific UID and GID will be assigned to it as if it were a normal user in Linux. Applications have a UID higher than 10000 and system accounts have a UID lower than 10000.
- The concept of permission is similar to the permissions on the Linux file system, however this is extended to be able to perform actions.
- The developer of the application decides which permissions the applications needs and has to define those permissions in a file that will be read at installation time (AndroidManifest.xml). For each application all the permissions are enforced through that file.
- Android runs a Mandatory Access Control Model and there is a reference monitor to check and implement the policies.
- The Android security architecture, by default, im-

plements a DENY policy so there is no permission to perform any operations that would adversely impact other applications, the operating system or the user.

Some key points that must be taken in consideration from a security point of view:

- The model is quite secure by default as there are different permissions for each process and application. From an OS point of view the isolation is performed efficiently and effectively.
- The main problem is when the user installs a new application that asks for more permissions than the necessary. This is the high risk part of the model where the security of Android can be broken.
- Another problem is when a developer creates an application that uses more permissions than necessary. It might be possible to create an application (malware) that ask for privileges to everything but the user has to agree with to those permissions.
- The model might be improved allowing the user to decide what privileges to permit for each application. A higher granularity might improve the security. With the current model all the permissions are granted or denied.

To summarise: the model is quite robust and it is well constructed. However, as in many case, the human factor is the problem. If a user allows applications to access everything or if a developer deploys and application with permissions to access everything the security is broken. The solution for this problem is to control which applications can be installed and not allow the user to install other applications.

C. Tools for Android to enforce security

There are some tools that can improve the security of the Android. Google Labs can define and enforce a password policy including complexity of the password, expiration time, historical, screen lock time out and a limit on the number of invalid password before wiping the device. It also permits the smartphone to be located through GPS, the device to be locked remotely, the alarm turned on or the mobile to be wipe remotely. Other kind of applications can be found on the Market like Norton Mobile Security [12], AVG Antivirus [13], Lookout [14] or Autowipe [15]. These application have the following features:

- Antivirus/antimalware: these scan the application installed and protect the smartphone while browsing.
- Remotely lock and wipe of the phone and SD card through SMS or via web.
- Backups of the information.
- Wipe the of mobile if the SIM is changed. Wipe of the mobile after a set number of failed logins.

II. Objectives of the project

The main targets of the project are:

- Implement a security channel of communication with VPN (OpenVPN [16]) and enforce all the traffic through the tunnel. Implement filters on the incoming and outgoing traffic.
- Lock out access to the device and centralize the access management. Access to the device will only be granted to security administrators through SSH with keys.
- Disable the installation of software. Only authorized software must be run and the user will not be able to install any software.
- Implement a policy password within the company standards. The smartphone will be wiped after a set number of trials.
- Implement remote control. It must be possible to wipe, locate and lock the smartphone remotely from the Internet and SMS.
- Implement Antivirus and Antimalware. It must be possible to scan the applications installed and protect the smartphone while browsing.
- Disable unnecessary services/devices. Services like bluetooth must be disabled as they are not necessary. SD card, if not necessary, might be disabled as well

III. Hardening the device

In this section I will explain the lab setup and the set of tools and steps taken to harden the system.

A. The lab

- HTC desire with a 3G card and WiFi connection.
- CyanogendMod 7.0.3 [17] as the main firmware/ROM. It is build based on Android 2.3.3.
- Android SDK toolkit [8] running on Ubuntu 11.04, Snow Leopard and Windows 7.
- Virtual Private Server (VPS) hosted in The Netherlands as the endpoint and management server.
- Different security tools like OpenVPN, iptables [19] and Dropbear (ssh server for Android) [18]

B. Encrypting the network channel and Implementing firewall policies

As I mentioned previously, the basic idea is to send all the traffic from the Android device, either the 3G (rmnet0) interface or the Wireless interface (eth0) through a VPN tunnel built with OpenVPN. In order to do this, there will be a VPN end-point, controlled by the company, running OpenVPN. The user of the smartphone will have to established the VPN with the endpoint or he/she will not have access. The way to enforce the network policy is through iptables in Android and at the endpoint. An example of the architecture is in the figure 2.

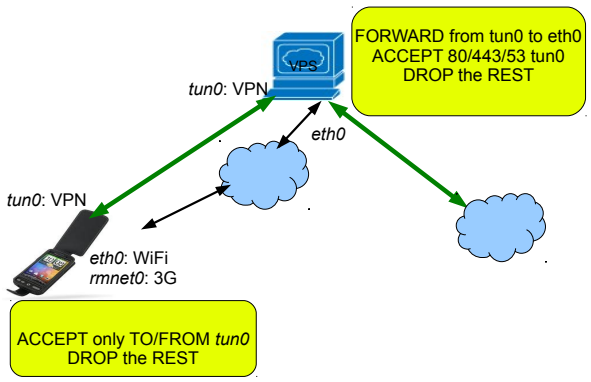

Fig. 2. Network connectivity through VPN

1) Configuring OpenVPN at endpoint with certificates: The encryption in a VPN connection can be established in 2 ways, with digital certificates (public key) or with a shared key. The main reason for using digital certificates instead of a shared key is because it is more flexible and more secure, and if any user loses the smartphone the certificate can be revoked. PKI also escalates better whit a higher number of users. Another interesting point is that if the company has its own CA this could be used to create the certificates for the server and for all the users instead of using a new ones. An interesting project about SmartCard API for Android is found in [20]. In the OpenVPN page it is possible to find the manual [21] to build the CA, the server certificate and the client certificate, but the main steps are:

- Configure the 'vars' that will contain the parameters of the certificate (ie. country, email, etc).
- Initialize the PKI and build the CA.
- Build the server keys and certificate.
- Build the client certificate/keys (one per each client).
- For each client, package its key/cert and the server cert in a '.p12' file. This is the standard used in Android to import the certificates

The OpenVPN manual [21] explains in detail how to configure the server through the 'server.conf' file, however some important decisions must be made:

- Which port and protocol to use: I decided to chose 80/tcp instead of using the default 1194/udp, since this port usually is not filtered in firewalls.
- Cipher Algorithms: I decided to use AES-256-CBC
- Compression: I decided to enable compression.
- Routes: as all the traffic will be routed through the VPN, it is necessary to propagate a route '0.0.0.0/0.0.0.0'.
- Static IP assigned per user. It is necessary to create a file with the virtual IP assigned for each user. This will help when creating specific firewalls rules based on the user/role.

2) Configuring OpenVPN in the smartphone with certificates: When the certificates for the clients are packaged, it is necessary to protect them with a passphrase. The certificates will be copied to the SD card and imported to the smartphone. Then, it is necessary to setup the VPN creating a new connection. The parameters to configure are:

- IP of the VPN server.
- Port and protocol: 80/tcp
- Redirect gateway: to route all the traffic through the VPN
- LZO compression: To compress the data
- Cipher Algorithm: AES-256-CBC
- Size of the cipher: 256 bit

3) Implementing firewall policies in Android: The objective is to apply a DENY default policy. The rules are set as follow:

- Interface lo: ACCEPT any IN/OUT
- Interface eth0/rmnet0: DENY IN/OUT. Exception: Traffic to the VPN IP (necessary to establish the tunnel and forward the traffic)
- Interface tun0: ACCEPT incoming SSH. ACCEPT OUT traffic to ANY to route the traffic through the VPN.

With these rules we will guarantee that when connecting through a public WiFi or 3G will not go on clear, avoid sniffing, and any traffic from the local network or Internet will be denied.

4) Implementing firewall policies at the endpoint: The security admin will enforce access control to the Internet through the endpoint. It is possible to create rules to allow traffic only to specific ports (like HTTP/S) and deny the rest of the traffic. It is also possible to create granular specific rules per user, through the static IP assigned to each user in the OpenVPN server (ie: allow sysadmin traffic to SSH).
It might be possible to configure a proxy (ie: squid) as HTTP/s content filtering. However, this is outside of the scope of this project. In the case of this project, the author will set the following rules:

- FORWARD the traffic from tun0 to eth0 to route the traffic to the Internet.
- NAT the traffic from tun0 to eth0 to have access to the Internet with the public IP of the endpoint.
- ACCEPT all incoming traffic to HTTP, HTTPS and DNS in tun0 interface.
- DENY the rest of the traffic.

With this set of policies we avoid any connection from the device that isn't HTTP/s (ie: Messenger, Skype, etc).

C. Granting access through SSH authenticated with keys

The version of the ROM used, Cyaongenmod [17], contains the ssh server binary, Dropbear [18], so further compiling it is not necessary. The manual that explains how to build configure and run the server is at [22]. In the case of this project the author will not use passwords but keys to access SSH. This is a more robust and secure way. The steps necessary to create this set up are:

- Create the pair of SSH key for each machine from which we want to connect from. In the case of the project, the author will create keys for the management server (VPN tunnel).
- Export the public key to the SD card of Android.
- Generate the SSH server keys with 'dropbearkey'.
- Configure the permissions of the files/directories for Dropbear.
- Import the management server keys from the SD card to the 'authorized_keys' file.
- Enforce dropbear to authenticate with keys and disable passwords when launching the process.

With this setup, it will only be possible to access the smartphone from SSH with the correct key. Besides, as we have enforced iptables rules, SSH port (22/tcp) is only opened on the tun0 interface, the VPN interface, as an additional security control.

If we want to start Dropbear once the system is booting, we have to include the command in an init script (like in any UNIX/Linux system).

D. Disabling bluetooth

The easiest way to disable bluetooth is just remove the permission in the bluetooth device. Bluetooth is managed by the kernel through the devices /dev/ttyHS0 and /dev/ttyMSM0. These devices have set read and write permissions (660) for the user and group bluetooth. If those permissions are removed the device will not be reachable.
This must be done when booting the system since the /dev directory is reset during reboot.

E. Avoiding the installation of software. Removing the access to the device through USB

As it has been explained before the greatest security issue in Android is the installation of software. Applications in android are packaged in 'apk' files and this can be installed in several ways: from the Internet through a repository like Market (or Appbrain) or through a USB connection. In order to prevent users from installing any software outside of the standard build, it is necessary to:

- Remove and disable Android Market, the application that permits to install from the Google repository
- Disable the binary file that manages packages in android from the shell, this is 'pm' (/system/bin/pm)
- Disable the access to the shell through the USB. This is done with the binary/daemon /sbin/adbd

To remove the Market application from the Android it is necessary to set the permissions of the container to 000. This will stop the possibility of running the Market. If the execution permission is removed from the 'pm' (/system/bin/pm) binary, it will not be possible to run it. If this same idea is applied to the binary /system/adbd it will not be possible to install packages from the shell or to access the android by USB. This has to be done when the system is booting, as the /sbin directory is mounted with read only permissions and it is reset every time the system is booted This has to be done at the end of the hardening process and after the SSH is configured, since once this step is applied there will no be access to the shell except through SSH.

F. Disabling the SD card

Connecting and external device (USB stick, SD card, etc) to a 'secure' system can be a big security risk whether it is a smartphone, a desktop or a server, etc. In the case of Android, the situation is the same, or even worse. By default android mounts the SD card as a FAT file system, with 'noexec', 'nodev'. The first question is why use FAT instead of ext3 or ext4, but

besides that, is the 'noexec' enough to prevent running the applications? Mario Ballano, a security researcher from Symantec, published a very interesting article [23] about how it is possible to use the SD card to hijack in order to steal information or execute malicious code. Besides the problem described by Mario, there is an additional issue: the lack of encryption on the SD card.

In order to disable the SD card, it is necessary to unmount it during the boot process:

- umount /mnt/sdcard/.android_secure
- umount -l /mnt/sdcard/

G. Disable unnecessary binaries

As part of any hardening process, it is good practice to remove compilers and unnecessary packages, set proper permissions to binaries (specially setuid/setgid), etc. In the case of Android, we will take into consideration the binaries with network access, editors and sniffers. The binaries affected are: irsii, nano, nc, netserver, netperf, opcontrol, scp, rsync, sdptest, ssh, strace, tcpdump, vim, bluetoothd, iptables, and ping.
Some of these will be just removed and for some others (ie: ping) the permissions will be setup only for root.

H. Remove unnecessary software. Remote installation of software

By default, Cyanogenmod doesn't contain many applications installed. For instance, the Market application is not installed.
It depends on the company policies and standards to decide which software can be installed and which software must be removed. Many companies have a software registry with the applications than are approved and this is applied to the standard Workstation or desktop build. The idea here is the same: define a set of standards applications and remove the rest. The main application that will be allowed that aren't part of the core system (like the settings application or clock/alarm) are: browser, mail client, calculator, calendar, camera, contacts, gallery, messaging, phone and voice dialer).
If there is any need to install an additional application (ie: twitter for the Marketing department or SSH for sysadmins), the security administrator will install it. To do it, the security admin will: upload the 'app' file with SCP through the VPN connection, change the permissions of the 'pm' binary to executable, install the application, rollback the permissions of 'pm' and remove the 'apk' file from the device. This model is very flexible and robust, because it is secure and any software can be installed remotely without any interaction from the user. In order to remove the unnecessary software, it is necessary to remove the 'apk' and the directory where the application is stored. The system application packages are in /system/app and the data is in /data/data. A good approach, to enable roollback if in future we want the application to be run, is to change the permissions of the directory to 000 instead of removing it, for example, for the bluetooth application might be chmod 000 /data/data/com.android.bluetooth.

IV. Additional security controls

As was mentioned at the beginning of the paper there are some existing tools that can add security to Android. The author will include some of these.

A. GoogleApps for Business: enforcing a password policy

The author was not able to find an external application to enforce password policies in Android, however GoogleApps for business can be used to do this (it cost 40$/year). The policies are defined and applied remotely and these will be synchronized automatically through the Internet. The application necessary to install in Android is 'device policy' and the functionalities are:

- Require the users to set passwords on the devices.
- Password complexity / strength. (At least one number, one letter, and one punctuation).
- Number of days before password expires (90 days).
- Number of historical password that are blocked (3 passwords).
- Automatically lock the device after a timeout (10 minute).
- Number of invalid passwords before the device is wiped (10 times).
- Allow camera (yes).
- Enable encryption on the device. (This is only available for version 3.0 of Android).

B. GoogleApps for Business: remote control of the device

Another set of features of the 'device policy' is the possibility to control the device remotely in case the device it is stolen or lost. The set of functions are:

- Locate the device through GPS and Google Maps.
- Lock the device.
- Reset the password.
- Turn on a noisy alarm.
- Wipe the mobile remotely.

C. Autowipe

Autowipe [15] is a tool that can wipe the phone and SD card. The SD card is not within the scope of this project because it is disabled. The author is interested in two functionalities: remote wipe through SMS with a passphrase and autowipe if the SIM card is changed. With this setup, if the phone is lost or stolen and someone change the SIM card, the phone will be wiped. In addition to this if we lose the phone and we do not have Internet at that moment, we can send a text message from someone elses phone to wipe it.

D. AntiVirus: AVG Mobile [13]

A very important layer of security in a defense-in-depth architecture is the Antivirus. The popular Antivirus company AVG has created a version for Android with Android which can perform the following functions:

- Scan applications, files and media in real time.
- Browse the web securely.
- Find/locate your lost or stolen phone via Google maps.
- Backup and restore all your valuable apps and data.
- Lock and wipe your device to protect your privacy.
- Kill tasks that slow your phone down.

For the purpose of this project we are only interested in the AV engine and secure browsing. The other functionalities have been covered with other tools.

V. Steps to harden the system from the scratch

Once the tools and ideas about how the system will be hardened, it is necessary to describe the steps to follow in order to do that.

A. The boot process in Android

Android boot system is well explained in 'The Android boot process' at [24]. The main issue with Android is that the init script (init.rc) is part of the Ramdisk and can't be modified. In order to modify it, it is necessary to rebuild the ramdisk. However, in Cyaongenmod firmware, it is possible to create a 'userinit.sh' script (in /data/local) that will execute when booting. The author will create the script to run the commands necessary to harden the system, like for example, iptables or dropbear (ssh).

B. userinit.sh

A copy of the script can be found in http://www.angelalonso.es/mssf/userinit.sh. Basically, the script contains the following commands:

- Run SSH (dropbear) at booting time.
- Disable the permissions of bluetooth device.
- Kill the market process if running.
- Harden the TCP/IP stack.
- Remove unnecessary binaries.
- Run the iptables script (iptables.sh) stored in /data/local.
- Run the software to remove unnecessary application (removesoftware.sh).
- Disable the 'pm' (package management) binary to avoid the installation of any software.
- Disable the 'adbd' daemon to avoid the installation of any software through USB and access to the shell through USB.
- Disable the SD card.

A copy of the iptables.sh and removesoftware.sh can be found in http://www.angelalonso.es/mssf/iptables.sh and http://www.angelalonso.es/mssf/removesoftware.sh

C. Putting it all together

The set of steps to have the system configure and run the system are as follow:

1) Install Cyanogenmod [17].
2) Install Google Apps.
3) Install the Google App Device Policy application.
4) Install the Antivirus application.
5) Install the Autowipe application.
6) Setup Google App Device Policy withe the enterprise account. A secure password to access the device will be introduced at this stage.
7) Setup the Antivirus: configure the update/auto-scan frequency, setup the real-time scanner and safe surfing.
8) Setup de Autowipe: enable SMS text wipe, choose a passphrase, enable the subscriber ID change and enable password protect to access the application.
9) Configure the VPN: import the certificate, setup the VPN parameters (IP, port, protocol, cipher algorithm, key size, LZO enable).
10) Configure SSH: generate the ssh keys, import the public key to authorized_keys.
11) Copy the iptables.sh, removesoftware.sh and userinit.sh script to /data/local. Change permissions to 700.
12) Reboot the system and the system and it will be hardened.

VI. Conclusions

The author has been able to improve the security of Android in different areas making the platform usable

in business where a high level of security is required. Implementing a security channel, reducing the risk of installing software, removing the unnecessary services and configuring a central point to manage all the devices are some of the key points achieved during this project. In addition the use of some existing applications like Google Apps or Antivirus increased the layers of security. Some scripts have also been created to lockout the operating system and enhance the security. These scripts can be adapted to each specific situation or business in order to align the configuration to the company polices, such as the kind of traffic allowed.

The possibility of applying different firewall rules with granularity (ie: per user) from a centralized system gives the platform a high grade of flexibility. The security admins can also upload software through SCP, and it is possible to install additional software if necessary without having physical access to the device.

VII. Future work

In future versions of Android (3.0 or above) encryption of the filesystem will be supported by default. This is a must in this kind of technology as mobility is key point. It will be necessary to integrate it in this hardened version. In this version of Android it might be possible to enable the SD card if it can be encrypted.

It might be good idea to deploy some kind of GUI to enforce the security policies, instead of doing it through scripts or configuration files. This GUI might also set up all the parameters for the external application such as Antivirus or autowipe in an automatic manner, so no intervention of the security admin will be necessary.

Other area that could be improved are the possibility of creating a password when booting the system, similar to that in a standard bios, so the system will not boot unless the password is correct. However, this will require to modification to the source code of Android.

References

[1] iPhone OS Enterprise Deployment Guide, [30/05/2011] http://www.apple.com/iphone/business/integration/

[2] Google Apps for Android [30/05/2011] http://www.google.com/apps/intl/en/business/mobile.html

[3] Catching AuthTokens in the Wild The Insecurity of Google's ClientLogin Protocol [30/05/2011] http://www.uni-ulm.de/en/in/mi/staff/koenings/catching-authtokens.html

[4] Firesheep [01/06/2011] http://codebutler.com/firesheep

[5] DroidDreamLight, New Malware from the Developers of DroidDream [30/05/2011] http://blog.mylookout.com/2011/05/security-alert-droiddreamlight-new-malware-from-the-developers-of-droiddream/

[6] The harsh reality of Android Market [16/06/2011] http://bit.ly/o3PYme

[7] The Current State of Mobile Device Security, Carey Nachenberg, [28/06/2011] http://bit.ly/iZceu4

[8] The Android SDK [20/05/2011] http://developer.android.com/sdk/index.html

[9] Mobile Security Application (book) [2010] Himanshu Dwivedi, Chris Clark, David Thiel. McGraw-Hill

[10] Android user ID and permissions [14/06/2011] http://developer.android.com/guide/topics/security/security.html

[11] Proposal of a model to improve the Security Model, [14/06/2011] Mir.Nauman. Security Engineering Research Group, Institute of management sciences, Peshawar http://imsciences.edu.pk/serg/2010/07/androidsecurityasurveyso-farsogood/

[12] Norton Mobile Applications [1/07/2011] http://community.norton.com/t5/Norton-Mobile-Apps-Public-Beta/bd-p/norton_mobile_pb

[13] Antivirus Free [1/07/2011] http://www.appbrain.com/app/anti-virus-free/com.antivirus

[14] Lookout application [1/07/2011] http://www.androidtapp.com/lookout-mobile-security/

[15] Autowipe: delete remotely the device [1/07/2011] http://bit.ly/mSy1Qq

[16] OpenVPN [22/6/2011] http://openvpn.net/

[17] CyanogenMod ROM [28/6/2011] http://www.cyanogenmod.com/

[18] Dropbear: SSH for Android [30/6/2011] http://matt.ucc.asn.au/dropbear/

[19] Iptables: firewall for Linux [26/6/2011] http://www.netfilter.org/

[20] Secure Element Evaluation Kit for the Android platform - the 'SmartCard API' [15/06/2011] http://code.google.com/p/seek-for-android/

[21] OpenVPN manual [22/6/2011] http://openvpn.net/index.php/open-source/documentation/howto.html

[22] Installation of SSH in Cyanogenmod [30/6/2011] http://bit.ly/nJheuA

[23] Android Class Loading Hijacking. Mario Ballano. Symantec [30/6/2011] http://www.symantec.com/connect/blogs/android-class-loading-hijacking

[24] The Android boot process from power on [28/5/2011] http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Acknowledgements

I would like to thank to my supervisor Renaat Verbruggen for his assistance. I would like to show my gratitude to Mss. Marie Celeste Woods who helped me reviewing the paper. Special thanks should be given to my parents and family for their love and support since always.