Use offense to inform defense.
Find flaws before the bad guys do.

Securely deploying Android devices

GIAC (GCIH) Gold Certification

Author: Angel Alonso-Parrizas, parrizas@gmail.com
Advisor: Antonios Atlasis

Accepted: September 22nd, 2011

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, are also growing. Devices based on Android has been adopted by companies and organizations due to their friendliness and the convenience they offer. However, not many companies have implemented security policies for using Android in a secure fashion, endangering the security of the company. During this paper it will be described how it is possible to improve the security of Android so it can be safely used in business where security is a high priority.

1. Introduction

Nowadays it is necessary for most companies to provide e-mail/Internet access to employees outside of the office, hence many business provide their staff with BlackBerrys, iPhones, Android or other smartphones with Internet connectivity. But, how can these comply with the enterprise'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 (Apple, 2010) which permit a security baseline to be defined for the handset, 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 (Google, 2011) that can implement some security policies in the mobiles; nonetheless, the security baseline that can be set through Google Apps is insufficient as some of the most 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 device is compromised by malware while connecting to the Internet? How to reduce the risk of this happening? How to detect it? How safe is it to plug in an external SD card if the business does not allow the plugin of any external USBs? Is it safe to allow the user to have bluetooth running? These are some examples of the kind of questions that should be considered when permitting the use of smartphones in an enterprise environment.

In this paper we will present a set of steps to improve the security of Android in different areas. This work is part of the final project (practicum) for the Master in Security and Forensic (MSSF) at School of computing in Dublin City University (DCU).

Angel Alonso-Parrizas, parrizas@gmail.com

2. Android: threats, architecture and some tools

2.1. Security threats in Android

Android is an Operating System (OS) based on Linux so 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. Nevertheless, 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 handset 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. However, this also results in a lack of some security layers of protection such as external firewalls, perimetrical antivirus or IDS/IPS which are found in many companies.

The article by Knings, Nickels and Schaub (2011) 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.

Applications in Android can be installed in different ways. The most popular one is the official Google repository, Market (https://market.android.com/). It is also feasible 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 (Wyatt, 2011). Some Spanish security researchers introduced malware in Android as PoC in order to demonstrate the lack of security in the Market (Jesus, 2011). Because this is the most critical part in the security chain, the end user must never be able to introduce any application and only authorized software should run in the device. 'The Current State of Mobile Device Security' report by Nachenberg (2011), a researcher from Symantec, 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. Those scenarios will not be considered and are out of the scope of this paper; for that reason, 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 GPS localization. The main problem is that versions of Android below 3.0 do not support encryption by default; as a result it is not possible to encrypt the device itself.

It must be taken into consideration if the SD card attached to the device is secure or not. Some of the points to keep in mind are that the information stored in the SD card is not encrypted and that the file system used is FAT. This means that all the applications with read permissions on the SD card have access to the whole data stored and consequently there is not any access list enforced through file permissions.

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 upload authorized software on the smart phone. Android can be accessed from the shell with the SDK toolkit (http://developer.android.com/sdk/index.html), but this gives the end user the possibility of also accessing the device through USB, and this is a security breach of the policies.

2.2. Android architecture: The security model

Android is composed of four layers, as can be seen in figure 1:

- Application: This is the layer for the application installed (i.e. 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. Also, in this layer, there are some libraries like SSL, SQLite or libc.
- Linux Kernel: it abstracts the hardware from the software.

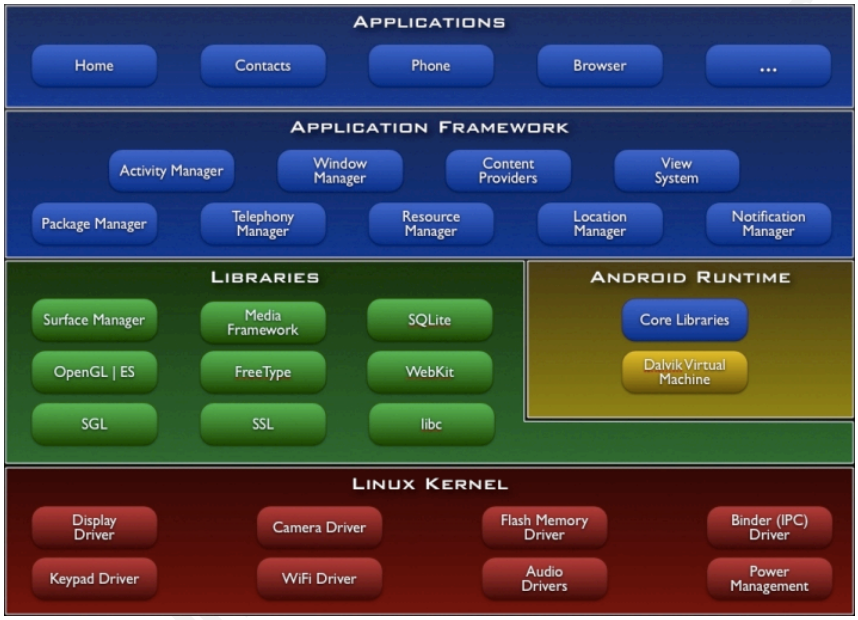

Figure 1: Android architecture

Android follows the same idea of user/permissions like a normal Linux system. This is very well explained in Android Developers guide, there are some key differences though. Android is not a multiuser system unlike traditional Unix/Linux systems where multiple external users are connected to the system. Instead, it uses the concept of UID and GID to assign permissions to each application and process and hence, there is isolation between processes/applications. Whenever a new application is installed, some specific permissions should be allowed. For example, if the application needs to access the GPS, it will request access to the GPS but 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 and this is extended to be able to perform actions. It is the developer who decides which permissions the applications needs through the file AndroidManifest.xml (full documentation in http://developer.android.com/guide/topics/manifest/manifest-intro.html), which is read at installation time. Android runs a Mandatory Access Control Model and there is a reference monitor to check and implement the policies described in the AndroidManifest.xml. The Android security architecture, by default, implements a deny policy. So there is no permission to perform any operations that would adversely impact other applications, the operating system or the user.

On account that there are different permissions for each process and application, it can be said that the security model is robust and from an OS point of view the isolation is performed efficiently and effectively. But the main problem is when the user installs a new application that asks for more permissions than 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 because it might be possible to create an application (e.g. malware) which asks for privileges to everything (but the user has to agree with those permissions).

A higher granularity might improve the security. For instance, the user should decide what privileges to permit for each application. With the current model all the permissions are granted or denied.

Mobile Security Application (Dwivedi, Clark, and Thiel, 2010, p. 16-47) is a good reference book which explains in detail Android architecture and other mobile technologies like iPhone.

In summary, the model is quite robust and it is well constructed. Nevertheless, as in many other cases, the human factor is the problem. If a user allows applications to access everything or if a developer deploys an 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.

2.3. Tools for Android to enforce security

There are some tools that can improve the security of the Android. Google Labs can define and establish 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 (Google, n.d.). It also permits the smartphone to be located through GPS, the device to be locked remotely, to turn the alarm on or the mobile to be wiped remotely. Other kind of applications can be found on the Market, like Norton Mobile Security, AVG Antivirus, Lookout, or Autowipe.

These applications include antivirus/antimalware software, which scan the application installed and protect the smartphone while browsing, as well as applications that can lock and wipe of the phone and SD card remotely through SMS or via web. Other tools permit to backup the information. Furthermore, there are tools to wipe of the mobile if the SIM is changed or wipe of the mobile after a set number of failed logins in order to protect confidentiality of the user data in these cases.

3. Improving the security of Android

3.1. Objectives of this work

The main objectives of this study, aiming at improving the security of Android devices, were the following:

1. Implement a security channel of communication (VPN) with OpenVPN (OpenVPN Technologies, 2002) and enforce all the traffic through this tunnel. Implement filters on the incoming and outgoing traffic.
2. 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.
3. Disable the installation of software. Only authorized software must be run and the user will not be able to install any software.
4. Enforce a policy password within the company standards. The smartphone will be wiped after a set number of trials.

5. Enable remote secure functionalities. It must be possible to wipe, locate and lock the smartphone remotely from the Internet and sending an SMS.
6. Include Antivirus and Antimalware tools. It must be possible to scan the applications installed and protect the smartphone while browsing.
7. 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.

In the rest of this section it will be explained the lab setup and the set of tools and steps taken to harden the system.

3.2. The lab

The lab is composed of an HTC desire with a 3G card and WiFi connection. The main firmware/ROM is CyanogendMod 7.0.3 (Cyanogenmod, 2011) build based on Android 2.3.3. All the setup and tests will be done with Android SDK toolkit running on Ubuntu 11.04, Snow Leopard and Windows 7. A Virtual Private Server (VPS) hosted in The Netherlands is used as the endpoint and management server. Moreover, a set of different security tools like OpenVPN, iptables and a SSH server for Android named Dropbear (Johnston, 2011) are used.

3.3. Encrypting the network channel and implementing firewall policies

As it was mentioned previously, the basic idea is to send all the traffic from the Android device, either via the 3G (rmnet0) interface or the wireless one (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 establish the tunnel with the endpoint or he/she will not have access to the Internet. The way to enforce the network policy is through iptables in the Android smartphone and at the endpoint (VPS). An example of the architecture is in the figure below:

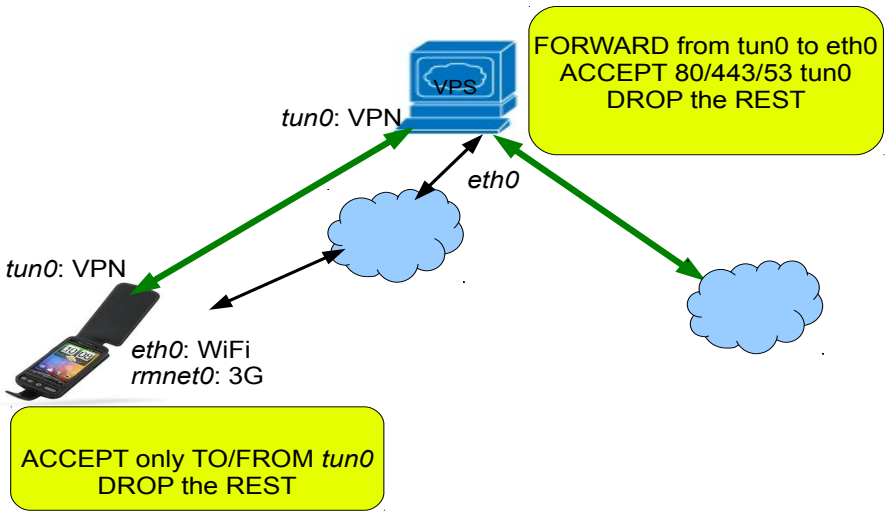

Figure 2: Network connectivity channel

3.3.1. Configuring OpenVPN at endpoint with certificates

The encryption in a VPN connection can be established in 2 ways, with digital certificates (public/private key pair) or with a shared key. The main reason for using digital certificates instead of a shared key is because it is more flexible and if any user loses the smartphone, the certificate can be revoked. PKI also escalates better with 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 new ones.

In the OpenVPN web page it is possible to find the manual (OpenVPN Technologies, 2002) to build the CA, the server certificate and the client certificate but the main steps are the following: First configure the 'vars' that will contain the parameters of the certificate (ie. country, email, etc). Then, initialize the PKI and build the CA. Build the server keys and certificate, and 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 explains in detail how to configure the server through the 'server.conf' file, nonetheless, some important decisions must be made. First of all, the port and the protocol to use. In our study, it was decided to choose 80/tcp instead of using the default 1194/udp, since this port usually is not filtered in firewalls (there might be some situations where web content filters are installed and the traffic is drop, but this port could be changed to other port such as 53/UDP). The second decision that we should make is about the algorithms to use. It was decided to use AES with CBC and a key of 256 bit (AES-256-CBC) as it is the default cipher and mode used by OpenVPN. Moreover, the compression was enabled and all the traffic is routed through the VPN; for this last reason, it is necessary to propagate a route '0.0.0.0/0.0.0.0'. Likewise, static IP was assigned per user to enhance security. Finally, 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.

3.3.2. Configuring OpenVPN in the smartphone with certificates

When the certificates for the clients are packaged, it is essential to protect them with a passphrase. The certificates will be copied to the SD card and imported to the smartphone. Then, it is needful to setup the VPN creating a new connection according to our previous selection of settings. In them, we should only add the IP of the VPN server and the redirect gateway, through which all the traffic will be routed.

3.3.3. Implementing firewall policies in Android

The objective is to apply a DENY default policy. In order to do that, only the IN/OUT traffic going through the interfaces eth0 and rmnet0 from/to the VPN's IP is ACCEPT and the rest is DROP. The traffic allowed is necessary to establish the VPN tunnel. Regarding the flows going through tun0, the virtual interface, all the OUT traffic is allowed to any IP to route the traffic to the endpoint, and the incoming traffic to port 22/tcp is allowed to permit SSH access.

With these rules it is guaranteed that when connecting through a public WiFi or 3G, the traffic will not go on clear, avoiding sniffing. Also, any other traffic from the local network or Internet will be denied.

3.3.4. 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 flows. It is also practicable to create granular specific rules per user, with a static IP assigned to each user in the OpenVPN server (e.g. allow sysadmin traffic to SSH). It might be viable to configure a proxy (e.g. squid) as HTTP/s content filtering. However, this is outside of the scope of this work.

In the case of this study some basic rules were set up, allowing traffic forwarding from tun0 to eth0 to route the traffic to the Internet. Also it is created a rule to NAT the traffic from tun0 to eth0, in order to have access to the Internet with the public IP of the endpoint. Additionally, a few set of rules are implemented to accept all incoming HTTP, HTTPS and DNS traffic in tun0 interface and deny the rest of the traffic.

With this set of policies, any connection from the device that isn't HTTP/s (e.g. Messenger, Skype, etc), will be denied.

3.4. Granting access through SSH authenticated with keys

The version of the ROM used, Cyaongenmod version 7.0.3, contains the ssh server binary, Dropbear and so, further compiling is not required. The manual that explains how to build, configure and run the server is at 'Connect to the device with SSH, (2011)" In the case of this work, it will not be used passwords but keys to access SSH. This is a more robust and secure way.

The set of steps to create this set up starts with the generation of the pair of SSH keys for each machine from which we want to connect from (in the case of this work, keys are created for the management server, the VPN tunnel). Then the public key has to be exported to the SD card of Android, included in the 'authorized_keys' file and the proper permissions has to be set to be readable by root. On the other hand, the SSH server keys has to be created and this is done with the 'dropbearkey' command in the smart phone. The last part is to run the dropbear process enforcing the authentication with keys and disabling passwords.

With this setup, it will only be possible to access the smartphone from SSH with the correct key. Moreover, as firewall rules have been enforced with iptables, the SSH port (22/tcp) is only reachable on the tun0 interface, the VPN interface. With the purpose of starting dropbear once the system is booting, it is mandatory to include the command in an init script (like in any UNIX/Linux system).

3.5. Disabling bluetooth

The easiest way to disable bluetooth is to just remove the permission in the bluetooth device. Bluetooth is managed by the kernel through the devices /dev/ttyHS0 and /dev/ttyMSM0 and 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.

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

As it has been explained in section 2.2, 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. To prevent users from installing any software outside of the standard build, it is essential to remove and disable Android Market, the application that permits to install from the Google repository, and the binary file that manages packages in Android from the shell, this is 'pm' (/system/bin/pm). Also, in order to disable the access to the shell through the USB, it is required to disable the binary/daemon /sbin/adbd.

To remove the Market application from the Android it is needed 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 feasible to run it. If this same idea is applied to the binary /system/adbd, it will not be viable to install packages from the shell or to access the Android by USB.

Disabling permission on binary files of the core system is done at booting time, as the /sbin directory is mounted with read only permissions and it is reset every time the system is booted. This step 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.

3.7. Disabling the SD card

Connecting an 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. 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' permissions. The first question is why use FAT instead of ext3 or ext4, but apart from that, is the 'noexec' permission enough to prevent running the applications? Mario Ballano, a security researcher from Symantec, published a very interesting article about how it is possible to use the SD card to hijack privileges from other applications in order to steal information or execute malicious code (Ballano, 2011) Moreover, in 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.

3.8. Disabling unnecessary binaries

As part of any hardening process, it is a good practice to remove compilers and unnecessary packages, set proper permissions to binaries (especially setuid/setgid), etc.

In the case of Android, the binaries with network access, editors and sniffers will be taken into considerartion. The affected binaries are the following: 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 (e.g. ping) the permissions will be setup only for root.

3.9. Remove unnecessary software

By default, Cyanogenmod doesn't contain many applications installed. For example, 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 that 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 which will be allowed that aren't part of the core system (like the settings application or clock/alarm) are a browser, a mail client, a calculator, a calendar, a camera application, contacts, gallery, messaging, phone and voice dialer. If there is any need to install an additional application (e.g. twitter for the Marketing department or SSH for sysadmins), the security administrator will install it. To do it, the security admin will upload the 'apk' 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 rollback in case in the future an application needs 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.

4. Additional security controls

As it was mentioned in section 2.3, there are some existing tools that can add security to Android. In this work some of them will be included.

4.1. GoogleApps for business: enforcing a password policy

It was not able to find an external application to enforce password policies in Android, nothwithstanding GoogleApps for business can be used to do this (the cost is 40$/year).

The policies are defined in the Google Apps web page and these are remotely synchronized /enforced automatically through the Internet. However, to do this it is needed to install in Android an application from the market, called 'device policy'. The set of functionalities that must be configured, are the following:

- 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 expires (10 minute).
- Number of invalid passwords before the device is wiped (10 times).
- Allow camera (yes).

4.2. 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 is stolen or lost. The set of functions are the localization of the device through GPS and Google Maps, lock the device, reset the password, turn on a noisy alarm and wipe the mobile remotely.

4.3. Autowipe

Autowipe is a tool that can wipe the phone and SD card.

We are interested in two functionalities provided by Autowipe (VesperaNovus, 2011), the remote wipe through SMS with a passphrase and the autowipe if the SIM card

is changed. On the contrary, the ability to wipe the SD card is not used because, the SD card is disabled for the reasons explained in section 3.7.

With this setup, if the phone is lost or stolen and someone changes the SIM card, the phone will be wiped. In addition to this, if the phone is lost and there is not Internet access, it is possible to send a text message from someone else's phone to wipe it.

4.4. Antivirus: AVG Mobile

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 (AVG, 2011) which can scan applications, files and media in real time. Moreover, it has an option to browse the web securely. There is a function to find/locate the phone via Google maps, backup/restore all the valuable apps and data and lock/wipe the device to protect the privacy.

For the purpose of this work the only interesting feature is the AV engine and the secure browsing. The other functionalities have been covered with other tools.

5. Steps to harden Android from scratch

Once the tools and the instructions how the system will be hardened has been defined, it is necessary to describe the steps to follow in order to do that.

5.1. The boot process in Android

Android boot system is well explained in 'The Android boot process' (Enea, 2009). 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. Nevertheless, in Cyaongenmod firmware, it is possible to create a 'userinit.sh' script (in /data/local) that will execute when booting. This script will run the necessary commands and will call the rest of the scripts

5.2. userinit.sh

A copy of the script can be found in the appendix. 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 script 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 and disable the SD card.

The scripts iptables.sh and removesoftware.sh can be found in the appendix.

5.3. Putting all together

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

1. Install Cyanogenmod.
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 with 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 the 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.

6. Conclusions

During this project it 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, control the traffic, reducing the risk of installing software, removing the unnecessary software and configuring a central point to manage all the devices are some of the key points achieved during this project. In addition to this, 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 or the software permitted. The possibility of applying different firewall rules with granularity (e.g. per user) from a centralized system gives the platform a high grade of flexibility. The security admins can also upload software through SCP and install without having physical access to the device.

7. References

Apple. (2010). *iPhone in Business*. Retrieved from http://www.apple.com/iphone/business/integration/

AVG. (2011). *Antivirus-free* [software]. Retrieved from http://www.appbrain.com/app/anti-virus-free/com.antivirus

Ballano M., (2001, June 29). *Android Class Loading Hijacking* [web blog]. Retrieved from http://www.symantec.com/connect/blogs/android-class-loading-hijacking

Connect to the device with SSH. (n.d). *Howto: connect to device with SSH* [web blog] Retrieved from http://bit.ly/nJheuA

CyanogenMod. (May 2011). *CyanogenMod ROM* [software]. Retrieved from http://www.cyanogenmod.com/

Dwivedi, H, Clark, C, & Thiel, D. (2010). *Mobile application security*. McGraw-Hill Osborne Media.

Enea. (2009, June 11). *The Android boot process* [web blog]. Retrieved from http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Google. (2011). *Google Apps for Mobile*. Retrieved from http://www.google.com/apps/intl/en/business/mobile.html

Google (n.d). *Device Policy Administration for Android*. Retrieved from http://www.google.com/support/a/bin/answer.py?answer=1056433

Jesus Y. (2001, June 6). *The harsh reality of Android Market* [web blog]. Retrieved from http://bit.ly/o3PYme

Johnston M. (2011) *Dropbear: SSH for Android (version 0.52)* [software]. Retrieved from http://matt.ucc.asn.au/dropbear/

Knings B., Nickels J. and Schaub. (2011, May 30). *Catching AuthTokens in the Wild The Insecurity of Google's ClientLogin Protocol*. Retrieved from http://www.uni-ulm.de/en/in/mi/staff/koenings/catching-authtokens.html

Nachenberg C. (2011, June 28). *The Current State of Mobile Device Security* [web blog]. Retrieved from http://bit.ly/iZceu4

OpenVPN Technologies. (2002). *OpenVPN* [software]. Retrieved from http://openvpn.net/

Angel Alonso-Parrizas, parrizas@gmail.com

VesperaNovus. (2011). *Autowipe: delete remotely the device* [software]. Retrieved from http://bit.ly/mSy1Qq

Wyatt T. (2011, May 30) *DroidDreamLight, New Malware from the Developers of DroidDream* [web blog]. Retrieved from http://blog.mylookout.com/2011/05/security-alert-droiddreamlight-new-malware-from-the-developers-of-droiddream/

8. Appendix

8.1. Userinit.sh

```
#!/system/bin/sh
# Customize some parameters and lockout the SO
# July 2011

mount -o rw,remount /system
# run dropbear / SSH
/system/xbin/dropbear -g -s
# Disable Bluetooth
chmod 000 /dev/ttyMSM0
chmod 000 /dev/ttyHS0
# stop market if running
killall com.android.vending
# hardening TCP/IP stack for IPV4
sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=1 #ICMP broadcast
sysctl -w net.ipv4.conf.all.accept_redirects=0 # ICMP redirects ipv4
sysctl -w net.ipv6.conf.all.accept_redirects=0 #ICMP redirects ipv6
sysctl -w net.ipv4.conf.all.send_redirects=0 # ICMP redirects
sysctl -w net.ipv4.conf.all.accept_source_route=0 #source routing disable
sysctl -w net.ipv4.conf.all.forwarding=0 #Forwarding traffic
sysctl -w net.ipv4.conf.all.rp_filter=1
sysctl -w net.ipv4.conf.all.log_martians=1 #filter martians
sysctl -w net.ipv4.tcp_max_syn_backlog=1280 # TCP syn half-opened
sysctl -w net.ipv4.ip_forward=0
# Removing/ disabling unnecessary binaries. Some of them have access to Internet
rm -f /system/xbin/irsii
rm -f /system/xbin/nano
rm -f /system/xbin/nc
rm -f /system/xbin/netserver
rm -f /system/xbin/netperf
```

```
rm -f /system/xbin/opcontrol
chmod 740 /system/xbin/scp
chmod 740 /system/xbin/rsync
chmod 740 /system/xbin/sdptest
chmod 740 /system/xbin/ssh
chmod 740 /system/xbin/strace
chmod 000 /system/xbin/tcpdump
chmod 740 /system/xbin/vim
chmod 000 /system/bin/bluetoothd
chmod 750 /system/bin/iptables
chmod 750 /system/bin/ping
chmod -s /system/bin/ping
# Run Iptables
/data/local/iptables.sh
## This is the last step of the hardening
## This will be uncommented only when the system is configured
## This will lockout the system and will only give access through SSH
# Removing unnecessary software
# This must be uncomment at last step
# do a backup before
/data/local/removesoftware.sh
# disable the Packet Management binary
chmod -x /system/bin/pm
# Disable the adbd daemon
mount -o rw,remount -t rootfs rootfs /
chmod -x /sbin/adbd
mount -o ro,remount -t rootfs rootfs /
# disable the SD card
umount /mnt/sdcard/.android_secure
umount -l /mnt/sdcard
mount -o ro,remount /system
```

8.2. iptables.sh

```
#!/system/bin/sh

/system/bin/iptables -F

# Traffic to localhost allowed

/system/bin/iptables -A INPUT -i lo -j ACCEPT

/system/bin/iptables -A OUTPUT -o lo -j ACCEPT

# Default deny policy

/system/bin/iptables --policy INPUT DROP

/system/bin/iptables --policy OUTPUT DROP

/system/bin/iptables --policy FORWARD DROP

# All the established sessions are allowed

/system/bin/iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

/system/bin/iptables -A OUTPUT -m state --state ESTABLISHED,RELATED  -p 6 --
sport 22 -j ACCEPT

# Traffic from the tunnel is allowed

/system/bin/iptables -A INPUT -i tun0 -p 6 --dport 22 -j ACCEPT

/system/bin/iptables -A INPUT -i tun0 -p 1 -j ACCEPT

# Traffic going to the VPS is allowed

/system/bin/iptables -A OUTPUT -d 46.249.37.120/32 -j ACCEPT

# Traffic going through the tun0 interface is allowed

/system/bin/iptables -A OUTPUT -o tun0 -j ACCEPT
```

8.3. removesoftware.sh

```
#!/system/bin/sh
# Remove and disable unnecessary software
```

```
# July 2011
cd /system/app/
rm /system/app/AndroidTerm.apk
rm /system/app/Bluetooth.apk
rm /system/app/Development.apk
rm /system/app/FM.apk
rm /system/app/CarHomeGoogle.apk
rm /system/app/GoogleFeedback.apk
rm /system/app/GoogleQuickSearchBox.apk
rm /system/app/FileManager.apk
rm /system/app/HTMLViewer.apk
rm /system/app/MarketUpdater.apk
rm /system/app/Music.apk
rm /system/app/RomManager.apk
rm /system/app/SoundRecorder.apk
rm /system/app/Talk.apk
rm /system/app/Torch.apk
rm /system/app/Vending.apk
chmod 000 /data/data/com.android.bluetooth
chmod 000 /data/data/jackpal.androidterm2
chmod 000  /data/data/com.android.development
chmod 000 /data/data/com.android.fm
chmod 000 /data/data/com.android.htmlviewer
chmod 000  /data/data/com.android.music
chmod 000 /data/data/com.android.musicvis
chmod 000  /data/data/org.openintents.cmfilemanager
chmod 000 /data/data/com.android.soundrecorder
chmod 000 /data/data/com.google.android.carhome
chmod 000  /data/data/com.google.android.apps.books
chmod 000 /data/data/com.google.android.googlequicksearchbox
chmod 000 /data/data/com.android.vending
```

Angel Alonso-Parrizas, parrizas@gmail.com

```
chmod 000 /data/data/com.android.vending.updater
chmod 000  /data/data/com.koushikdutta.rommanager
```

Last Updated: November 13th, 2018

Upcoming SANS Penetration Testing

SANS San Francisco Fall 2018	San Francisco, CA	Nov 26, 2018 - Dec 01, 2018	Live Event
SANS Stockholm 2018	Stockholm, Sweden	Nov 26, 2018 - Dec 01, 2018	Live Event
SANS Austin 2018	Austin, TX	Nov 26, 2018 - Dec 01, 2018	Live Event
Austin 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Austin, TX	Nov 26, 2018 - Dec 01, 2018	vLive
Mentor Session AW - SEC560	Colorado Springs, CO	Nov 28, 2018 - Dec 07, 2018	Mentor
SANS Nashville 2018	Nashville, TN	Dec 03, 2018 - Dec 08, 2018	Live Event
SANS Santa Monica 2018	Santa Monica, CA	Dec 03, 2018 - Dec 08, 2018	Live Event
SANS Dublin 2018	Dublin, Ireland	Dec 03, 2018 - Dec 08, 2018	Live Event
Mentor Session AW - SEC504	St. Petersburg, FL	Dec 05, 2018 - Dec 14, 2018	Mentor
Community SANS Tampa SEC560	Tampa, FL	Dec 10, 2018 - Dec 15, 2018	Community SANS
SANS Frankfurt 2018	Frankfurt, Germany	Dec 10, 2018 - Dec 15, 2018	Live Event
SANS Cyber Defense Initiative 2018	Washington, DC	Dec 11, 2018 - Dec 18, 2018	Live Event
Mentor Session AW - SEC542	Oklahoma City, OK	Dec 12, 2018 - Jan 25, 2019	Mentor
Cyber Defense Initiative 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Cyber Defense Initiative 2018 - SEC560: Network Penetration Testing and Ethical Hacking	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Mentor Session AW - SEC504	Hong Kong, China	Dec 22, 2018 - Jan 26, 2019	Mentor
SANS Bangalore January 2019	Bangalore, India	Jan 07, 2019 - Jan 19, 2019	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	SEC504 - 201901,	Jan 08, 2019 - Feb 14, 2019	vLive
Mentor Session - SEC542	Denver, CO	Jan 10, 2019 - Mar 14, 2019	Mentor
SANS Threat Hunting London 2019	London, United Kingdom	Jan 14, 2019 - Jan 19, 2019	Live Event
Sonoma 2019 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Santa Rosa, CA	Jan 14, 2019 - Jan 19, 2019	vLive
SANS Amsterdam January 2019	Amsterdam, Netherlands	Jan 14, 2019 - Jan 19, 2019	Live Event
SANS Sonoma 2019	Santa Rosa, CA	Jan 14, 2019 - Jan 19, 2019	Live Event
Cyber Threat Intelligence Summit & Training 2019	Arlington, VA	Jan 21, 2019 - Jan 28, 2019	Live Event
SANS Miami 2019	Miami, FL	Jan 21, 2019 - Jan 26, 2019	Live Event
SANS Las Vegas 2019	Las Vegas, NV	Jan 28, 2019 - Feb 02, 2019	Live Event
SANS Security East 2019	New Orleans, LA	Feb 02, 2019 - Feb 09, 2019	Live Event
Community SANS Minneapolis SEC504	Minneapolis, MN	Feb 04, 2019 - Feb 09, 2019	Community SANS
Security East 2019 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	New Orleans, LA	Feb 04, 2019 - Feb 09, 2019	vLive
Security East 2019 - SEC542: Web App Penetration Testing and Ethical Hacking	New Orleans, LA	Feb 04, 2019 - Feb 09, 2019	vLive
Mentor Session - SEC560	Fredericksburg, VA	Feb 06, 2019 - Mar 20, 2019	Mentor