FOX-IT

DFRWS 2011 Challenge Final Report

Investigation results

22 July 2011

Ivo Pooters
pooters@fox-it.com

Pascal Arends
Arends@fox-it.com

Steffen Moorrees
Moorrees@fox-it.com

This document describes the investigation results for the DFRWS 2011 challenge.

1 Related Documents

DFRWS 2011 - timeline.png	Time line
Sqlite_carving_extractAndroidData.pdf	Document on *extractAndroidData*
yaffs2_oh_carving_extractObjectHeaders.pdf	Document on *extractObjectHeaders*
\Toolkit_reports\case1\extractAndroidData\extractAndroidData_mtdblock6.img.html	*extractAndroidData* report for case 1
\Toolkit_reports\case2\analyzeAndroidData\analyzeAndroidData.html	*analyzeAndroidData* report for case 2

Contents

2 Goals

2.1 Case 1: Donald Norby

The following goals were set by us for this case:

- What applications are installed on the device?
- Are traces present on the device that indicate a relation between Norby and a group called Kryptix?
- With whom did Norby communicate in the last days before his death?
- What did Norby communicate about?
- Are traces present that indicate if Norby was planning suicide?
- Are traces present that give need to a wider investigation?

2.2 Case 2: Yob Taog

The following goals were set by us for this case:

- What applications are installed on the device?
- With whom did Yob communicate?
- What did Yob communicate about?
- Are traces present that indicate the intellectual documents were distributed (un)intentionally through the device or otherwise?
- Are traces present that give need to a wider investigation?

3 Summary and conclusions

This chapter is the summary/conclusion of the findings from the investigation on Case 1. and Case 2. The goals described in chapter 1. are listed with their respective answers. The goals from case 1. and case 2. are put in a logical order. The reader is advised to consult the timeline parallel with this section for an overview of the key events. The timeline is included in a separate PNG file with the results.

Case 1: What applications are installed on the device?

We have investigated the device of Donald Norby for traces of installed applications. Normally there would be a *packages.xml* file that lists all applications installed on the device. Because of technical difficulties with the supplied images we had to use carving techniques to recover the content of the *packages.xml* file. After getting insight into the content of the *packages.xml* file we could conclude that there was one application installed by the user. This installed application was *com.twitter.android*. For the complete list of installed applications the reader is referred to the *extractAndroidData* report included with the results.

Case 2: What applications are installed on the device?

We have investigated the device of Yob Taog for traces of installed applications. On the user data partition there is a file called *packages.xml* this file lists all applications that are installed on the device. The complete list of all installed applications is listed in the *analyzeAndroidData* included with the results.

Beside a couple of applications that probably got installed by Yob we found two malicious applications that were installed before Yob purchased the device. These two applications are being discussed later on in section 7.14.

Case 1: With whom did Norby communicate in the last days before his death?

During the investigation we found traces that indicate the device of Donald Norby was used to place a call to the telephone number 4124623802. Open source investigation shows that this telephone number belongs to the Verizon Wireless store in the Waterfront shopping centre in Pittsburgh. With the device of Donald Norby this number was called two times on 5/4/2011 on 8:04 PM (UTC-4) and 11:18 PM (UTC-4).

The device of Donald Norby was also used to call the number 4439264768. According to the contact information list found on the device this number belongs to Mr E. The number of Mr E is called twice on 5/4/2011 7:31 PM and 8:38 PM (UTC-4). Mr E. is called once to the device of Norby. This call was made on 5/8/2011 2:46 PM(UTC-4). Beside the calls Between Norby and Mr E they have exchanged multiple text messages. In total Norby has sent five text messages to Mr E., and received three text messages from Mr E.

The contact information of Mr E. also contained an e-mail address. This e-mail address is *mre@hushmail.com*. On the device of Norby we found several e-mails that were exchanged between the e-mail addresses *norby441@gmail.com* and *mre@hushmail.com*. In total six e-mails were exchanged between the two e-mail addresses. Because of technical difficulties we could not verify if

all the e-mails where actually sent. In total we recovered eleven e-mails. Three of these e-mails were sent by Google services.

Case 1: What did Norby communicate about?

On the device of Norby we recovered multiple text messages between Norby and Mr E. In Table 1 all exchanged text messages are listed.

Id	Number	Date/time (UTC)	Read	Status	Content
6	4439264768	5/5/2011 1:09 AM	1	out	Got the perfect Guy, plan is already in motion
7	4439264768	5/5/2011 1:12 AM	1	in	Break a leg
63	4439264768	5/6/2011 6:30 PM	1	out	the implementation seems to be working ok, no gold yet though
155	4439264768	5/8/2011 6:05 PM	1	out	Got some results, I think we need to up the fee, say double?
156	4439264768	5/8/2011 6:16 PM	1	in	You are joking, right? You can't seriously think about changing the deal now.
157	4439264768	5/8/2011 6:22 PM	1	out	I just sent you a sample, I think you'll be pleased...
158	4439264768	5/8/2011 6:30 PM	1	in	You are serious then. I can see the information is valuable but I am displeased with you breaking the deal.
159	4439264768	5/8/2011 6:56 PM	1	out	I knew you'd like them, ill be at the agreed spot, in about 25 min for the exchange

Table 1. SMS communication with Mr E.

We recovered multiple e-mail communications between the e-mail addresses *norby441@gmail.com* and *mre@hushmail.com*. These e-mails are listed in Table 2 Some of the records could not be completely recovered because of the Sqlite overflow page construction. In these cases the unknown column values are filled with the string ***overflow***.

Id	From	To, cc, bcc	DateSentMs (UTC)	Subject	Body
2	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:08 PM	sample	this is just a taste, much more where this came from. N.
1	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:08 PM	sample	this is just a taste, much more where this came from. N.
3	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:20 PM	sample	this is just a taste, much more where this came from. N.
4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:32 PM	showing i'm serious	This information is obviously very valuable. I 'd like to keep our relationship, but these will fetch aÂ
4	"norb k" <norby441@gmail.	"Mr E" <mre@hushmail.	5/8/2011 6:32 PM	showing i'm serious	This information is obviously very

	com>	com>			valuable. I 'd like to keep our relationship, but others would be willing to pay more. Â Here a
4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:34 PM	**overflow**	**overflow**
4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:34 PM	**overflow**	**overflow**
5	"" <mre@hushmail.com>	"norb k" <norby441@gmail.com>	5/8/2011 6:43 PM	Re: showing i'm serious	-----BEGIN PGP SIGNED MESSAGE----- Hash: SHA1 I certainly don't want you giving these files to someone else. Expect a call from me shortly. On Sun, 08 May 2011 14:34:46 -0400 norb k <<a href="mailto:norby441

Table 2. E-mail communication with Mr E.

On the device of Norby we recovered 299 text messages that are coming from Yob's Number. Based upon the content of those text messages it looks like the most of these messages are status messages. A couple of the text messages are forwarded messages from Yob's device. Down below in Table 3 are some examples of the recovered status messages. For a complete listing of all the text messages and all forwarded messages see the *analyzeAndroidData* report included with the results.

Phone number	Timestamp (UTC)	Type	Message
4124393388	05/05/2011 12:50:22 AM	In	ksmsvzwsms://message/Service Started
4124393388	05/05/2011 12:50:32 AM	In	ksmsvzwsms://message/May 4, 2011 8:50:12 PM EDT
4124393388	05/05/2011 12:53:31 AM	In	ksmsvzwsms://message/pkg uploaded!
4124393388	05/06/2011 04:52:12 PM	In	ksmsvzwsms://message/CallIn: 5854561283 May 6, 2011 12:51:31 PM EDT

Table 3. Examples of status messages.

Case 2: With whom did Yob communicate?

During the investigation we found traces that Job was called on 5/6/2011 12:51 PM (UTC-4) by the number 5854561283. With open source investigation we could not identify this number. This telephone conversation lasted 20 seconds. On 5/6/2011 Yob received two calls from Adrian (7607058888) with a total duration of 237 seconds. Yob called Adrian on 5/6/2011 once but they properly did not speak to each other; the call duration was 0 seconds.

Yob exchanged multiple e-mail and text messages with Luke Lancer, Sandra Peif, Reg Wetham and two system administrators from Swiftlogic

On the device of Donald Norby we found traces of multiple text messages from Yob's device. We could not recover the exact messages from Yob's device. Based upon the content of the messages it seems these where automatically generated messages.

Case 2: What did Yob communicate about?

Most of the communications that we found were of personal nature. But there were some messages that were business related. These where e-mail messages about an IT-outage and that Yob needed some PDF files for a presentation. The most relevant messages are listed in Table 4. For the exact content of the exchanged messages the reader is referred to the *analyzeAndroidData* report included.

Whom	**Timestamp (UTC)**	**Message (snippit)**	**Attachments**
From: yobtaog@gmail.com To: swiftlogic@consultant.com swiftlogicllc@consultant.com swiftlogicinc@consultant.com	05/06/2011 07:35:05 PM	helpdesk, I was unaware of the server outage starting today and need some files to work on this weekend, there is a very big meeting on Monday. Can you please email me sheets from project 2228, I need the ones that I've most recently modified…	
From: swiftlogic@consultant.com To: yobtaog@gmail.com	05/07/2011 11:11:04 PM	Re: File request: Mr Taog- My apologies, we kind of have our hands full down here with the main...	2228-11.pdf 2228-12.pdf 2228-15.pdf
From: swiftlogic@consultant.com ToL yobtaog@gmail.com	05/07/2011 12:40:49 PM	Re: File request: Mr Taog- It looks like Tim found your files, but he just went out for breakfast...	2201-4.pdf 2201-7.pdf 2201-8.pdf 2201-9.pdf 2228-7.pdf 2228-10.pdf
From: yobtaog@gmail.com To: regwetham@yahoo.com	05/07/2011 07:04:55 PM	Tonight: hey, I got the docs i needed from the helpdesk. Are you still planning on heading to that cigar bar tonight? yob	

Table 4. Business related Communications

On the device of Donald Norby we found traces of multiple text messages from Yob's device. We could not recover the exact messages from Yob's device. Based upon the content of the messages it seems these where automatically generated messages.

Phone number	**Timestamp (UTC)**	**Type**	**Message**
4124393388	05/05/2011 12:50:22 AM	In	ksmsvzwsms://message/Service Started
4124393388	05/05/2011 12:50:32 AM	In	ksmsvzwsms://message/May 4, 2011 8:50:12 PM EDT
4124393388	05/05/2011 12:53:31 AM	In	ksmsvzwsms://message/pkg uploaded!
4124393388	05/06/2011 04:52:12 PM	In	ksmsvzwsms://message/CallIn: 5854561283 May 6, 2011 12:51:31 PM EDT
4124393388	05/08/2011 03:10:38 AM	In	ksms**FORWARDED** SMS from 6245 at 20110507T190532America/New_York(6,126,-

			14400,1,1304809532) :shandra@cheerful.com (Re: Or you can walk down) Walking down now. Hop
4124393388	05/07/2011 11:40:09 PM	In	ksms**FORWARDED** SMS from 6245 at 20110507T135649America/New_York(6,126,-14400,1,1304791009) :shandra@cheerful.com (Save me!) If Luke asks, I'm going out with you dinner, OK? I just can't face Mr. Smooth tonight. Shandra

Table 5, example of automatically generated messages

The two forwarded messages look like messages Yob received but have been forwarded to Norby's device. For a complete list of received text messages, the reader is referred to the *extractAndroidData* report included.

Case 2: Are traces present that indicate the intellectual documents are spread (un)intentionally with the device?

During the investigation on Yob's device we have found no evidence that indicate that Yob intentionally spread the intellectual documents of Swiftlogic. We have found traces on Yob's device that a malicious application is installed. We found traces that make it very likely that this malicious application is responsible for spreading the Nine pdf files which very likely contain intellectual property of Swiftlogic. This malicious application is installed on 5/4/2011 20:49 (UTC-4).

Below follows a reconstruction of the relevant events.

In the morning of 5/4/2011 11:50 AM (UTC-4) Yob left a message on his Facebook profile stating that he had called his favorite Verizon store at the Pittsburgh Waterfront. He asked them if they have android phones in stock. And he says "I will be heading out there a little later to pick up that bad boy." We found traces on Norby's device that Norby searched for Swiftlogic and Yob Taog. These searches took place on 5/4/2011 between 8:06 PM (UTC-4) and 8:16 PM (UTC-4). With Norby's device is called to the Verizon store at 5/4/2011 on 8:04 PM (UTC-4) and 11:18 PM (UTC-4). With Norby's device was send a text message to Mr E. that he has found the perfect guy. This text message was sent 5/4/2011 9:09 PM (UTC-4). On Yob's twitter account he posted a message around 5/4/2011 9:40 PM (UTC-4) that he bought the device.

At 5/6/2011 2.30 PM (UTC-4) with Norby's device a text message is sent to Mr E. "the implementation seems to be working ok, no gold yet though" This leads us to the assumption that it is possible that Norby persuaded or ordered someone in the Verizion store to install the malicious application on Yob's device.

The operation of the malicious application is that with some events it sends a text message to 4124393389. These events are: incoming and outgoing calls, sending and receiving a text message and if the malicious application uploads files to a web server (50.56.29.109 port 1001). The malicious application has the capability to upload files that are present on the SD card to a web server.

Yob received nine PDF files per e-mail from his system administrators because of the IT outage. He receives these files on 5/6/2011 between 11.11 PM (UTC-4) and 5/7/2011 12.40 PM (UTC-4). Norby receives a text message at 5/8/2011 12.12 PM (UTC-4) from Yob's device with the message

“ksmsvzwsms://message/pkg uploaded!”. At 5/8/11 between 1:59 PM (UTC-4) and 2:02 PM (UTC-4) Nine PDF files are downloaded onto Norby’s device. These PDF files where very likely downloaded from the url *http://50.56.29.109/ss*. The downloaded files found on Norby’s device are equal to the nine pdf files that Yob received per e-mail from the system administrators.

After the files got downloaded with Norby’s device he sends a text message on 5/8/2011 2:05 PM (UTC-4) to Mr E.. The content of this text message is “Got some results, I think we need to up the fee, say double?”. Later on Norby and Mr E. make plans to meet; this meeting would take place on 5/8/2011 around 3:20 PM (UTC-4). Analyzing the the communication between Norby and Mr E. we believe that the meeting was to exchange the PDF files for money. This leads us to the assumption that it is likely that Norby distributed the intellectual documents belonging to Swiftlogic.

Case 1: Are traces present that indicate if Norby was planning suicide?

We have found no traces indicating that Norby was planning a suicide. However we found traces that Norby was making a deal with Mr E. and that he proposed to change the agreed fee. See Table 6. Communication to Mr E.

Type	Timestamp (UTC)	Message
e-mail	5/8/2011 6:32 PM	This information is obviously very valuable.Id like to keep our relationship, but others would be willing to pay more. Here a
SMS	5/8/2011 6:05 PM	Got some results, I think we need to up the fee, say double?

Table 6. Communication to Mr E.

After Norby suggested to “up the fee” Mr E. is displeased with Norby breaking the deal. After Norby has announced he wanted to change the deal he had a meeting with Mr E. on 5/8/2011 around 3:20 PM (UTC-4). After that meeting there are no further activities on Norby’s device except some status messages and forwarded messages from Yob’s device. This leads us to the assumption that Mr E. will possible be one of the last persons or even the last person who saw Donald Norby alive.

Case 1: Are traces present on the device that indicate a relation between Norby and a group called Kryptix?

We examined Donald Norby’s device and used open source intel to find a possible relation between Donald Norby and the Kryptix group. After a comprehensive and complete investigation we didn’t find any connection between Donald Norby and the Kryptix group.

Within this case there are some loose ends which must be further investigated. The results of this follow-up investigation can possible lead to a connection between the group Kryptix and Donald Norby. These loose ends are being discussed by the Case 1, investigative question “Are traces present that give need to a wider investigation?

Case 1 and Case 2: Are traces present that give need to a wider investigation?

We did not find any trace that Norby planned a suicide. However we found traces that Norby possibly was involved in an illegal transfer of documents. These documents where most likely from Yob Taogs device. We found traces that Norby was meeting someone called Mr. E. to exchange these documents. Traces were found in Case 1. and Case 2. that Norby was possibly involved installing a malicious application on Yob Tag’s device. Before Yob bought his device Norby had

contact for a short time (91 seconds) with someone in the Verizon Wireless store in the Watergate shopping mall in Pittsburgh.

We advise to investigate with whom Norby had contact in the Verizon Wireless store. We believe that the person who Norby called in the Verizon Wireless store possibly installed the malicious application discovered on Yob's device. This person in the Verizon Wireless store could provide information on the origin of the software and persons involved in this scheme.

We have found no trace on the device of Norby about the identity of mr E. That is why we advise to investigate on the phone number and e-mail address of mr E.

We found traces that the nine PDF documents were at some point made available on a webserver with the IP-address *50.56.29.109*. As mentioned before these documents likely contain intellectual property from Swiftlogic. We advise to investigate further to whom the webserver "50.56.29.109" belongs to and who operated the web server. When we logged in to the web server we got the message "*no files have been uploaded in the past 48 hours. check the server logs first, then call Cyph3r if there are issues.*" It is possible that *Cyph3r* is a nickname for someone related to hosting and/or maintenance of this web page.

Investigation of the web server could provide traces about the identity of Cyph3r.

4 Evidentiary Issues

4.1 Case 1: Donald Norby

Was the device "rooted"?

According to the acquisition log, the device is rooted using SuperOneClick 1.7.0.0 tool. The relevant extract from the log is listed in Listing 1.

```
1:52 PM USB debugging was determined to not be enabled on the device via
"adb devices" returning no serial numbers
1:53 PM enabled USB debugging on the devices via settings -> Application
Settings -> Development -> USB debugging
1:53 PM verified USB connectivity with "adb devices."  the device is
identified as "040373BF0B01B01A device"
1:55 PM used "adb shell" to verify that adb on the device is running as
"shell" user and "su" is not present in the path.  The device is likely
not "rooted"
1:56 PM used SuperOneClick 1.7.0.0 with psneuter option to achieve a
temporary root shell on the device
1:58 PM reconnected to device with "adb shell" verified root access
```

Listing 1. Extract from acquisition log

Did the device have "adb" enabled?

According to the acquisition log, ADB (Advanced Debugging Port) was enabled by the investigator manually through the user interface on the phone.

Was the collection process sound? (and can you verify the recorded acquisition log)

The current date and time of the device were not reported at time of acquisition.

Acquisition of the SD card was done using the command:

```
dcfldd if=/dev/sdb of=./SDCard.img hash=md5 hashwindow=500MB
md5log=SDCardHashed.txt hashconv=after conv=noerror,sync
```

Acquisition of the internal YAFFS2 partitions was done using the command:

```
dd if=/dev/block/mtdblockX of=/sdcard/mtdblockX.img
```

The data from was copied by using dd on the mtdblock devices. However, reading from mtdblock devices using dd does not include data in spare areas. The result is images containing only the sequence of bytes in the 2048-byte pages, but not the Out Of Band (OOB) bytes. Andrew Hoog posted a message on this issue:

http://www.telesphoreo.org/pipermail/g1-hackers/2009-February/000767.html

It is very difficult to reconstruct the file system from this image.

The acquisition log appears complete and can be fully verified.

Indicate version information

According to the acquisition log, the device was a Motorola A855 (Droid).

Information about the OS version is stored on the system partition in *build.prop.* The setting line `ro.build.version.release=<version>` indicates the Android version. Since we cannot reconstruct the file system it is not possible to extract the *build.prop* file as such.

Performing a key-word search on the string *ro.build.version.release=* yields multiple results with different values. Both values *2.0.1* and *2.1-update1* were found. This is most likely caused by downgrades and/or upgrades of the OS. So it is not clear from this which is the current OS version.

We found a JPG file on the SD memory card (see section 6.3) tells us that the image was likely taken with Norby's device and at that moment (5/6/2011) was running Android 2.1-update1. This can be seen in the EXIF information of the file:

```
Exif.Image.Make:Motorola
Exif.Image.Model:Droid
…
Exif.Image.Software:2.1-update1
Exif.Image.DateTime:2011:05:06 14:43:3
```

Recover credentials, if possible

This is described in more detail in section 6.7.4.

List applications installed

This is described in more detail in section 6.2.

4.2 Case 2: Yob Taog

Was the device "rooted"?

According to the acquisition log, the device is rooted.
Event records in the acquisition log indicate the agent successfully switched to Root with "su".
In the log it is not explicitly stated if the agent rooted the device or that he acquired the device already rooted.

```
[user1@exam3 platform-tools]$ ./adb shell

# su

# nanddump /dev/mtd/mtd0 | transfer 9000
```

Did the device have "adb" enabled?

According to the acquisition log, ADB was enabled.
The acquisition log states that the agent used a "adb shell" to acquire the data of the device. In the log is not stated if the agent enabled ADB or that he got the device this way. We can also verify this setting in a settings file on the user data partition at
data/com.android.settings/shared_prefs/com.android.settings.preferences.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<boolean name="location_network" value="true" />
<boolean name="enable_adb" value="true" />
<boolean name="location_gps" value="true" />
</map>
```

Was the collection process sound? (and can you verify the recorded acquisition log)

The current date and time of the device were not reported.

The acquisition log of the SD card does not describe which method was used to acquire the SD card. Only MD5 hashes over sectors are stated in the log and the overall MD5 hash. Also the post acquisition hash is not recorded in the log. With the available acquisition log it is not possible to verify if the SD memory was acquired in a forensically sound manner.

Acquisition of the internal YAFFS2 partitions was performed using the command:

```
nanddump /dev/mtd/mtdX.
```

Nanddump is an executable from the mtd-utils[1] package that can be used to make a bit-by-bit copy of an MTD device. To our knowledge, it is a suitable tool to create a bit-by-bit copy.

The output of this command is piped to the agents examination machine using adb forward and netcat. The hash value outputted by *nanddump* is recorded in the log. The files are saved as *mtdx.dd* and then hashed with *sha1sum* on the agents examination machine.

The acquisition log is not verifiable. The log commands are only partially available, and the agent did not record in what state he seized the device.

Indicate version information

Information about the OS version is stored on the system partition in *build.prop.* The setting line `ro.build.version.release=<version>` indicates the Android version.

In the build.prop file it is stated that the current OS version is 2.0.1.

Recover credentials, if possible

We have recovered Twitter credentials, these are discussed in section 7.12.1.

We also recovered Gmail credentials, these are discussed in section 7.11.

List applications installed

This is described in more detail in section 7.3.

[1] ftp://ftp.infradead.org/pub/mtd-utils/

5 Background Information

5.1 Android and YAFFS2

Android uses the Linux MTD subsystem to address NAND memory. The internal NAND memory is partitioned by Android into MTD devices or partitions. The number of MTD partitions may vary and depend on the device model and Android version. As far as we know at least the following partitions exist on an Android device:

- Recovery partition
- System partition
- User data partition
- Cache partition
- Boot partition

The mtd partitions are numbered from 1 to n. Information about the use of the partitions can be acquired by examining the Linux mount table, size of the partition and looking at the */proc/partitions* file.

5.1.1 YAFFS2

Yet Another Flash File System 2 is a flash file system designed for NAND flash memory. It is the follow-up of YAFFS. In YAFFS everything that is stored in the file system is named an *object* and is uniquely identified by YAFFS by their *object ID.* An object can represent any of the following:

- File
- Directory
- Special (pipes, devices etc)
- Hard link
- Symbolic link

5.1.2 YAFFS2 NAND model

The memory in NAND flash is arranged in *pages*. A page is the unit of allocation and programming. In YAFFS the unit of allocation is a *chunk.* Typically a chunk will be equal to the underlying page.

A *block* is the unit of erasure. A block consists of many chunks, typically 32 to 128. Blocks can become corrupt of damaged. YAFFS implements a mechanism to detect bad blocks and mark these as bad.

YAFFS2 objectives to work with newer NAND types include:

- Zero overwrites. YAFFS2 never performs overwrites. Thus, no deletion markers or other markers are used.
- Sequential chunk writing within a block. Within a block YAFFS2 writes the chunk strictly sequential.

5.1.3 YAFFS file storage

YAFFS2 has a true log structure. This means, that chunks are written only sequentially and no chunk is ever written to twice.

Instead of writing data in locations specific to the files, the file system data is written in the form of a sequential log. The entries in the log are all one chunk in size and can hold one of two types of chunk:

- Data chunk: holds the actual file data.
- Object header: A descriptor for the object. This holds details (meta-data) of the object such as parent directory, object name, timestamps etc.

Each chunk has tag associated with it. The tags comprise the following relevant fields:

- Object Id: The identifier of the object the chunk belongs to.
- Chunk Id: Identifies where in the object this chunks belongs. A chunkId of zero indicates that this is an object header chunk. ChunkId == 1 indicates the first chunk.
- Byte count: The number of bytes used in this chunk. (only for data chunks)
- Sequence number: As each block is allocated, the file system's sequence number is incremented and each chunk in the block is marked with that sequence number. This provides a way to organize the chunks in chronological order. Shrink header marker: Used to mark object headers that are written to shrink the size of a

5.2 Android applications

Applications are written in Java and packed into an Android package (APK). This actually an archive file (zip/rar) with extension .apk. This file is used to install the application.

Upon installation the APK file is stored in */data/app/*.

Each application receives its own unique user ID. The files needed for the application are set to that UID. This UID is constant for that application for the lifetime of the app on that device. It may have another UID on a different device or after a re-install.

The UID's are determined at time of installation and registered in /data/system/packages.xml. The UID number starts at 10000 and is incremented by 1 each time an application is installed. [2]

The packages.xml registers the installed applications and looks something like this:

```
<?xml …>
<packages>
…
<package name="com.package.example.name"
codePath="/data/app/com.package.example.name.apk" system="False"
ts="1283990162000" version="25" userId="10066"
installer="com.google.android.feedback">
…
</packages>
```

The developer declares the components of the application in an *AndroidManifest.xml* file which is mandatory and packed into the APK package.[3] The manifest file is something of the form:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
```

[2] http://wiki.cyanogenmod.com/index.php?title=Fix_permissions
[3] http://developer.android.com/guide/topics/fundamentals.html

```
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

5.3 Relevant Android Files

5.3.1 Firmware/OS information

Information about the operating system and the firmware can be found in the following locations:

- On the system (ROM) partition in *build.prop*. This file contains information on the Android version like version number (2.01, 2.1, 3.0 etc), firmware date and API level.
- On the user data partition in */data/property* contains multiple files which are named *persist*.* which contain certain relevant system wide settings. The files *persist.service.adb.enable* and *persist.sys.timezone* contain the adb setting and timezone setting respectively.

Information about the operating system settings can be found in the found in the following locations:

- On the user data partition in *Com.android.provider.settings/databases/settings.db*.
- On the user data partition in *Com.google.android.provider.settings/databases/settings.db*.

5.3.2 Contact data and call history

Contact information and call history information is stored in an SQLite database on the user data partition *data/com.android.providers.contacts/databases/contacts2.db*. Listing 2 shows the relevant tables from this database.

Contacts
General contact information

Raw_contacts
More contact information, linked to contacts

Data
Contains all extra contact data like address, email address, phone number etc
Linked to raw_contact.
The mimetype id indicates what kind of data it is:
Phonenr
- Data1: display number
- Data2: Type of number (1=home, 2=mobile)
- Data4: number in LE format

Postaddress:
- Data1: display address
- Data2: type of address
- Data7: city
- Data9: Zip code
- Data4: street

Name:
- Data1: display name
- Data2: first name
- Data3: last name

Email:
- Data1: email address
- Data2: type of email address (1 = home, 2 = work)

Mimetypes
Contains the various mime types of the contact stored in the data table.

Contacts
Contains last contacted date and time

Calls
Contains last incoming and outgoing calls
Type 1: incoming
Type 2: outgoing
Type 3: missed

Listing 2. contacts2.db relevant tables

5.3.3 SMS/MMS

SMS messages and MMS messages are stored in an SQLite database file on the user data partition at *data/com.android.providers.telephony/databases/mmssms.db*. Listing 3 shows the relevant tables from this database.

Sms:
Type: 1 voor inkomende SMS, 2 voor uitgaande sms

Parts:
Contains parts of MMS messages.

PDU:
Contains MMS messages
M_type 130 and 135 when messages are sent?
Updated to m_type just before sending

Pending_msgs
The MMS messages that are waiting to be sent.

Threads
Threads of SMS or MMS messages

Addr
The addresses of the MMS receivers (and senders?).

Listing 3. mmssms.db relevant tables

5.3.4 Gmail

Gmail messages are stored in an SQLite database file on the user data partition at *data/com.google.android.providers.gmail/databases/mailstore.<username>@gmail.com.db*. For every Gmail account one such database file will be present in the same directory. Listing 4 shows the relevant tables from this database.

Attachments
Contains information about mail attachments.
Conversations
Contains the conversation information.

Messages
Contains the messages

Sync settings
Store some synchronizations settings like conversation age, clientId,

Listing 4. mailstore.<username>@gmail.com.db relevant tables

5.3.5 Browser data

Browser history, bookmarks and searches are stored in an SQLite database on the user data partition at *data/com.android.browser/databases/browser.db*.

Bookmarks
Contains both bookmarks and browser history.

Searches
Contains google searches??

Credentials are usually stored in SQLite databases named *webview.db*. Many of these databases can exist on a typical Android device. Applications on an Android device can use the WebView API to create a browser screen. For security reasons, such application do not share the browser credentials, cookies and cache with the Android web browser. Therefore each application using the webview API gets its own webview.db to store this information.

The browser webview.db is located on the user data partition at *data/com.android.browser/databases/webview.db*. Listing 5 shows the relevant tables from a webview database.

Passwords
Contains stored credentials used for web urls

Httpauth
Contains stored credentials used for http authentication at web urls

Cookies

Formdata and formurl
Contain memorized form values for automatic form filling

Listing 5. webview.db relevant tables

The browser's cached files are stored on the user data partition at *data/com.android.browser/cache/webviewcache/**. The index for these cached files is stored in an SQLite database on the user partition at *data/com.android.browser/databases/webviewcache.db*. Only one relevant table is present in this database, cache.

5.3.6 Location info

Location information is stored in multiple SQLite database files on the user data partition:

- *Data/com.google.android.location/files/cache.cell*. Cache of GSM cells connected to.
- *Data/com.google.android.location/files/cache.wifi.* Cache of WiFi points.
- *data/com.android.browser/app_geolocation/cachedPositions.db*. Cached browser positions.

- *data/com.google.android.apps.maps/databases/search_history.db*. History of searches in Google Maps.
- *data/com.google.android.apps.maps/databases/da_destination_history.* History of navigation destinations in Google Maps.

5.3.7 Google Talk

Google Talk messages are stored in an SQLite database on the user data partition at *data/com.google.android.providers.talk/databases/talk.db*.

5.3.8 Other applications

Every application in Android can store information in databases. Because of the sandbox model it is however trivial to locate user data generated by a certain application. Each application can only store data in its own directory or via special content providers on the SD card. The application directory is located on the user data partition at *data/<com.location.url.name>/*.

5.4 References

http://www.ssddfj.org/papers/SSDDFJ_V4_1_Lessard_Kessler.pdf

http://www.yaffs.net/

http://www.yaffs.net/files/yaffs.net/HowYaffsWorks.pdf

http://www.yaffs.net/files/yaffs.net/YaffsTuning.pdf

6 Investigation of Case 1

6.1 Relevant preliminary findings

6.1.1 YAFFS2 partitions

From the *proc/partitions* file on the system partition, it was observed that 8 YAFFS partitions exist on the device. These are listed in Table 7. The partitions that are most likely used to store user data are mtdblock5 (mounted on */cache*) and mtdblock6 (mounted on */data*).

Since user data is only stored on the user data partition, cache partition and SD memory card, we have focused our investigation on these devices.

Device	Mounted on	Size	Description
Mtdblock0	*/config*	1.536 KB	
Mtdblock1		384 KB	
Mtdblock2			
Mtdblock3		4.608 KB	Recovery
Mtdblock4	*/system*	143.744 KB	System
Mtdblock5	*/cache*	94.848 KB	Cache
Mtdblock6	*/data*	268.032 KB	Userdata
Mtdblock7		2.048 KB	Kernel panic

Table 7. Yaffs2 partitions acquired from Norby's device

6.1.2 Time and time zone information

The time zone information is stored on the user data partition in the directory */property* in a file named *persist.sys.timezone*. Since we cannot reconstruct the file system, it is not possible to read the file as such. We have developed a python program *extractObjectHeaders* to extract object headers with a certain file name from a YAFFS2 partition. This method is further described in the included document. *extractObjectHeaders* shows that only one object header exists on the user data partition with the specified file name *persist.sys.timezone* at decimal offset 1716224. The object header, according to the YAFFS2 algorithm, can be written before or after the changed content. We examined the chunks before and after the object header. The first chunk before the object header contained only the string *America/New York* (UTC–4 in DST).

Given the content and the position of the chunk, we find it very likely that the chunk belongs to the file *persist.sys.timezone*. Since only one object header with this file name exist, we can assume that it belongs to the most recent version of the file.

From the evidence collection report we could not confirm that the date and time settings were correct at the time of evidence collection. For the remainder of this report, we will work under the assumption that the date and time were set correctly.

6.2 Installed applications

We have investigated the device for installed applications. Background information on Android and applications installations can be found in section 5.2.

When a package is installed using the APK installer engine on Android, the application is registered in the */data/system/packages.xml* file. This is the place to look for installed applications. Since the file

system could not be reconstructed using the YAFFS2 meta-data, it was not possible to extract the *packages.xml* file as such.

We have used FTK 3.3 to carve the user data partition for elements of the *packages.xml* file. Any carve software that allows the user to configure the signatures can be used for this. We chose to carve for specific elements of the XML file, instead of the whole *packages.xml* file. Given the log structured nature of the file system, it is very likely that the *packages.xml* file will be highly fragmented and thus it would be difficult to carve for the whole file.

The *packages.xml* file format is described in section 5.2. Each installed package is registered as an XML element *package*. An example of a package element is depicted below:

```
<package name="com.package.example.name"
codePath="/data/app/com.package.example.name.apk" system="False"
ts="1283990162000" version="25" userId="10066"
installer="com.google.android.feedback">
…
</package>
```

So a package element can be identified by the header

```
<package
```

and footer

```
</package>
```

or

```
 />
```

Depending on whether the xml element has sub elements.

We created a custom carving signature for FTK 3.3 to carve for XML package elements of *packages.xml* files which consists of the header and a footer signature of the package element.

A small python program (*processCarvedPackageXML*) is created to parse the results from the carving run and output the registered packages in a HTML table. The algorithm of the program is summarized in Listing 6.

```
For each file in a specified folder:
  Attempt to parse the file as an xml element.
  If this succeeds:
    Parse out any <package … /> elements.
  For each package element:
     Parse the xml element attributes of the element.
     Create a hash from the name, codepath, flags,  version, uid and date/time.
     Store package under the hash in a hash table. Overwriting any previously stored package with
exact same attributes. (Deduplication)

 Output the package registries to a html file.
```

Listing 6. Algorithm outline of processCarvedPackageXML

For a complete list of the resulting packages the reader is referred to the included report generated by *processCarvedPackageXML*. It is very likely that the user only installed one application: *com.twitter.android*. The application is installed or last updated on 5/6/2011 2:27 PM (UTC-4). We

observed that only one other package is registered as being installed or last updated after 12/19/2010, *com.android.vending* (Android marketplace).

Given the fact that old *packages.xml* pages in the file system are not overwritten within a short time after being deleted, we regard it very likely that no other packages.xml registrations were present on the device than the ones we found.

Therefore, we find it very likely that at the time of acquisition no other applications were installed than the applications listed in the *processCarvedPackageXML* report.

6.3 SD-card analysis

We examined the SD memory card using Forensic Toolkit 3.3. The file system on the SD memory card is the FAT32 file system. The file system was examined for existing and deleted files and folder structure. Also, the SD card was carved for files of all relevant types.

One JPG file was found on the SD memory card in *DCIM/Camera/* named *2011-05-06 14.43.35.jpg*. This picture was likely created on 5/6/2011 at 2:43 PM (UTC-4). It is a picture from within what looks to be a car, driversea Figure 1.

Figure 1. Car picture

```
Exif.Image.Make:Motorola
Exif.Image.Model:Droid
…
Exif.Image.Software:2.1-update1
Exif.Image.DateTime:2011:05:06 14:43:32
```

The EXIF data and the location of the JPG indicate that the picture was very likely taken with Norby's device. No geographic location information was present in the EXIF data.

On the SD memory card in the folder *download/* nine PDF files were found containing schematics from SwiftLogic. The modified times do not differ more than one second from creation times, therefore they are not mentioned in the table. This finding indicates a link between Norby and Swiftlogic. The *download/* folder is created and/or used by Android's native browser to store file that are downloaded using the web browser.

Name	L-Size (bytes)	Created (UTC-4)	MD5
2201-4.pdf	29541	5/8/2011 1:59:56 PM	A145BA75735B5ACC9C43AA2759C9B126
2201-7.pdf	42881	5/8/2011 2:00:18 PM	4AA76F74ADA38E97D9D7113EF8E3C44E
2201-8.pdf	52359	5/8/2011 2:01:22 PM	0A36E386F6F0FED84E80850739C96174
2201-9.pdf	46343	5/8/2011 2:01:08 PM	3F3A48026B33E093FF841852F7AF20BC
2228-10.pdf	136844	5/8/2011 2:01:57 PM	0F899A47B55289FCF1D6DA9915183A69
2228-11.pdf	260786	5/8/2011 2:02:20 PM	44A39F3F3E57DC50F4EBC04D0F9ADB00
2228-12.pdf	48113	5/8/2011 2:02:33 PM	EB8FDF32EB18598F931D23E703D8A3BD
2228-15.pdf	47157	5/8/2011 2:02:54 PM	A7D03C5CA92A5913E6B929FE94FA96F2
2228-7.pdf	177047	5/8/2011 2:01:47 PM	197620727BB96ECBBC8AF62BB22107DE

Table 8. PDF files on SD memory card

6.4 Carving the cache partition

Using both FTK 3.3 and Photorec we carved the cache partition for all relevant files of the following categories:

- Archive files
- Image files
- Audio/video files
- Documents
- HTML and XML files

No relevant files were found.

6.5 Carving the user data partition

Using both FTK 3.3 and Photorec we carved the cache partition for all relevant files of the same categories as mentioned above.

We found multiple cached HTML pages originating from: http://50.56.29.109/ss**.** The pages show an open directory listing with PDF files. Accessing the web location, we observe that it is protected by http authentication.

The open dir listing in the cached pages also shows that 9 PDF files are listed with names equal to the PDF files found on the SD memory card in the *download* folder. Further, the open dir list shows last modified dates which are all on 5/8/2011 on 5:54 PM. The listing is depicted in figure 1. It is unclear in which time zone the server is located and if the time is shown in UTC time or local time.

Open source investigation shows that server is hosted by Slicehost and the IP-address was registered on 4/19/2011. No reliable information on the geographical location was found.

Cached HTML pages were also found of twitter pages among which of the twitter account *yob_taog* (http://twitter.com/#!/yob_taog), of Facebook searches, of LinkedIn searches and google searches. From the searches it shows the user appeared to show interest in swiftlogic, Yob Taog and Robert Warr.

This led us to secure the content of the Twitter, Facebook and LinkedIn pages of Yob Taog and SwiftLogic.

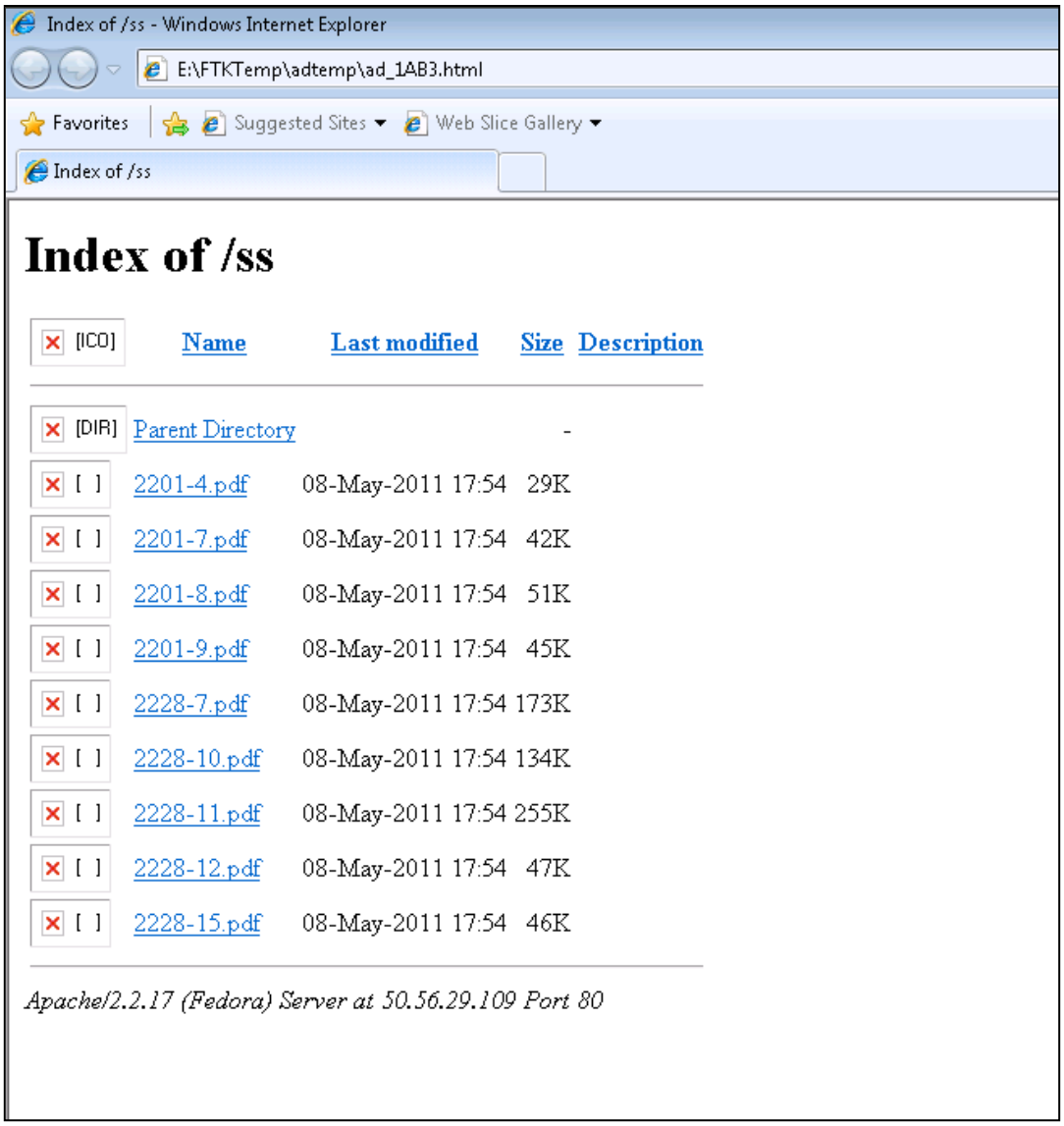

Figure 2. recovered html file of open dir listing

6.6 Location information

Android may store location information in the file */data/com.google.android.location/cache.cell* or *cache.wifi* files. These files hold information on the GSM cell or the WIFI access point the phone was connected to. When the Android phone is first initialized the user is asked if the user wants to send and use anonymized location information on the phone. If this is option not enabled, the phone will not store this location information.

We have scanned the YAFFS2 pages for object headers of the file *cache.cell* or *cache.wifi* using a self created program *extractObjectHeaders*[4]. No object headers with file name *cache.cell* or *cache.wifi* were found. This makes it very likely that this location information is not present on the phone.

[4] The reader is referred to the included document on *extractObjectHeaders* for a detailed description of the program.

6.7 Browser information

We created and used a python program named *extractAndroidData*[5] to extract browser history records, browser downloads, browser bookmark records, geo location information and browser credentials from the user data partition of the device. None of these traces reside on the other partition or on the SD memory card.

6.7.1 Geo location information

No browser geo location records were found on the user data partition.

6.7.2 Bookmarks

Bookmark records were recovered, but none of these appear relevant to this investigation.

6.7.3 Browser history

We observed from the history records that on 5/4/2011 at 8:06 PM (UTC-4) the phone user started a search on swiftlogic and employees of swiftlogic. The search appears to end at the facebook page of Yob Taog: *http://www.facebook.com/profile.php?id=100002336995096*.

Further, on 5/6/2011 at 2:27 PM (UTC-4), the user searches on twitter for *yob*_taog and visits the twitter page of Yob Taog: *http://mobile.twitter.com/yob_taog.*

On 5/8/2011 at 1:59 PM (UTC-4), the user visits the web page at *http://50.56.29.109/ss/*. Visiting this url using a web browser shows that the page is protected by http authentication. In section 6.5 we described a finding where an html page was recovered that very likely originated from this same url. Therefore we find it very likely that behind the http authentication at some moment a website with an open dir list was active.

We recovered in the browser history which files where downloaded with Norby's device. The relevant files are listed in Table 9.

ID	Download location	Saved location	Timestamp (UTC)
6	http://@50.56.29.109:80/ss/2201-4.pdf	/sdcard/download/	05/08/2011 05:59:54 PM
7	http://@50.56.29.109:80/ss/2201-7.pdf	/sdcard/download/	05/08/2011 06:00:17 PM
8	http://@50.56.29.109:80/ss/2201-9.pdf	/sdcard/download/	05/08/2011 06:01:06 PM
9	http://@50.56.29.109:80/ss/2201-8.pdf	/sdcard/download/	05/08/2011 06:01:21 PM
10	http://@50.56.29.109:80/ss/2228-7.pdf	/sdcard/download/	05/08/2011 06:01:45 PM
11	http://@50.56.29.109:80/ss/2228-10.pdf	/sdcard/download/	05/08/2011 06:01:55 PM
12	http://@50.56.29.109:80/ss/2228-11.pdf	/sdcard/download/	05/08/2011 06:02:17 PM
13	http://@50.56.29.109:80/ss/2228-12.pdf	/sdcard/download/	05/08/2011 06:02:31 PM
14	http://@50.56.29.109:80/ss/2228-15.pdf	/sdcard/download/	05/08/2011 06:02:51 PM

Table 9. Downloads using web browser

The relevant browser history records are listed in Table 10. For a complete listing of the recovered browser history records, the reader is referred to the *extractAndroidData* report included. For a description of Android and browser data, the user is referred to section 5.3.5.

6.7.4 Browser credentials

The *extractAndroidData* program carves records from the *password* and *httpauth* tables from *webview.db* sqlite databases. For information on this database, the reader is referred to section

[5] The reader is referred to the included document on *extractAndroidData* for a detailed description of the program.

5.3.5. Multiple *webview.db* databases may exist on the user data partition for storing web related information of various applications. When a user decides to store credentials for certain web pages, Android will create a record in the *password* table or the *httpauth* table of the browser *webview.db* database.

We recovered one set of credentials from the user data partition. These credentials were stored for host *50.56.29.109*. The username is *norby* and the password is *aaassspp*.

We have used these credentials to login to the web page at *http://50.56.29.109/ss/*[6]. The web page at this location shows the following html code:

```
<html>
<body>
no files have been uploaded in the past 48 hours.<br>
check the server logs first, then call Cyph3r if there are issues.
</body>
</html>
```

It is possible that Cyph3r is a nick name for someone related to hosting and/or maintenance of this web page.

[6] In the Netherlands a court order is required to log in to server that is not property of the subject. For this case, we make the assumption that this requirement is fulfilled and the agent is allowed to login.

date/time (utc)	title	url	Visits
5/5/2011 12:06 AM		**http://www.google.com/m/search?q=swiftlogic**&pbx=1&aq=f&oq=&aqi=g5-k0d0t0&fkt=4740&fsdt=15642&cqt=&rst=&htf=&his=&maction=&source=android-home&csll=&action=&ltoken=98e4b1a1c8dda	1
5/5/2011 12:06 AM		**http://www.google.com/m/search?**pbx=1&source=android-home&aq=f&oq=&aqi=-k0d0t0&fkt=1948&fsdt=7351&cqt=&rst=&htf=&his=&maction=&**q=swiftlogic+employee**	1
5/5/2011 12:07 AM	Swift Logic Technologies LLC, Catonsville, MD	**http://m.manta.com/c/mrlb6kb/swift-logic-technologies-llc?page=company**	1
5/5/2011 12:07 AM	Bob Warr profiles | LinkedIn	**http://www.linkedin.com/pub/dir/Bob/Warr**	1
5/5/2011 12:08 AM		**http://www.google.com/m/search?**pbx=1&source=android-home&aq=f&oq=&aqi=-k0d0t0&fkt=4027&fsdt=16213&cqt=&rst=&htf=&his=&maction=&q=**swiftlogic+linkedin**	1
5/5/2011 12:08 AM	Swift Logic | LinkedIn	**http://www.linkedin.com/pub/swift-logic/b/437/3a3**	1
5/5/2011 12:14 AM	www.swiftlogic.gr | Facebook	**http://www.facebook.com/apps/application.php?id=106785212675838**	1
5/5/2011 12:16 AM	Taog Yob | Facebook	**http://www.facebook.com/people/Taog-Yob/100002336995096**	1
5/5/2011 12:16 AM		**http://www.facebook.com/profile.php?id=100002336995096**	1
5/6/2011 6:27 PM	yob_taog - Twitter Search	**http://search.twitter.com/search?q=yob_taog**	1
5/6/2011 6:27 PM		**http://search.twitter.com/search?q=yob_taog**	1
5/6/2011 6:27 PM		**http://m.twitter.com/yob_taog**	1
5/6/2011 6:27 PM	Twitter	**http://mobile.twitter.com/yob_taog**	1
5/6/2011 6:27 PM		**http://mobile.twitter.com/yob_taog**	1
5/6/2011 6:28 PM	Twitpic - Share photos and videos on Twitter	http://twitpic.com/4tscf6	1
5/6/2011 6:28 PM	Twitpic - Share photos and videos on Twitter	http://twitpic.com/4tvmcu	1
5/8/2011 5:59 PM	Index of /ss	http://50.56.29.109/ss/	1
5/8/2011 6:28 PM	Index of /ss	http://50.56.29.109/ss/	2

Table 10. Recovered browser history records

6.8 Contact information

We used the *extractAndroidData*[7] program to extract contact information from the user data partition of the device. No contact information resides on the other partitions or on the SD memory card.

Three contacts were found stored in Norby's contact list. The contacts are listed in Table 11.

Contact name	Number(s)	Other info
Mr E	4439264768	Email: mre@hushmail.com
Taog	4124393388	
Mr e	4439264768	

Table 11. Recovered contact records from Norby's phone

Further we used the same script to extract call history records from the user data partition.

Number	Date/time (utc)	Duration (secs)	In/out	Name
4439264768	05/04/2011 11:31:08 PM	341	Out	Mr E
4124623802	05/05/2011 12:04:01 AM	91	Out	
4439264768	05/05/2011 12:38:17 AM	115	Out	Mr E
4124623802	05/05/2011 03:18:33 PM	84	Out	
4439264768	05/08/2011 06:46:24 PM	381	In	Mr E

Table 12. Recovered call history records from Norby's Phone

It is very likely that during the period 5/4/2011 to 5/8/2011 Norby was engaged in multiple telephone conversations with the number stored under the name of *mr E.* Further, Norby contacted the number *4124623802* twice with call durations of 91 seconds and 84 seconds respectively. Performing open source investigation on this number leads to a Verizon Wireless store in the Waterfront shopping centre in Pittsburgh.

http://magicyellow.com/Chains/Verizon_Wireless/Pittsburgh_PA.html

The shop is located at 163 E Bridge St Homestead, PA 15120. The shop, among others, sells smart phones.

6.9 Text messages

We used the *extractAndroidData* program to extract SMS communications (text messages) from the user data partition of the device. No text messages reside on the other partitions or SD memory card.

In total 318 text message records were recovered from the user data partition. 299 Text message records result from text messages received from the number *4124393388*. This number is stored with contact name Taog. If we deduplicate the records based on all values except the type of the message (read/unread), 150 text message records are left. We find it very likely that these are the text messages that have been received from *4124393388* between 5/5/2011 and 5/11/2011.

Judging from the content it appears that something automated was sending status text messages and forwarded communications from the number *4124393388.* Examples of these found text messages are

[7] The reader is referred to the included document on *extractAndroidData* for a detailed description of the program.

ksmsvzwsms://message/May 9, 2011 12:03:09 AM EDT

and

ksmsvzwsms://message/Service Started

An example of a forwarded message is

ksmsFORWARDED SMS from 6245 at 20110508T045142America/New_York(0,127,-14400,1,1304844702) :shandra@cheerful.com (Thanks) Thanks for being so gracious last night

All these text messages have the prefix *ksms*.

Also, text messages were exchanged with the contact stored as M*r E*. The text messages recovered from the device very likely communicated with the number *4439264768* (Mr E.) are listed in Table 13. The conversation appears to be about an implementation with the intention to provide result in some form. On 8/5/2011 0:13 AM (UTC-4), a draft message was created indicating that results were acquired. The message is not sent until 8/5/2011 2:05 PM (UTC-4). A discussion follows about the fee and at 2:56 PM Norby states he will attend at the arranged exchange location in 25 minutes.

Id	Number	Date/time (UTC)	Read	Status	Content
6	4439264768	5/5/2011 1:09 AM	1	out	Got the perfect Guy, plan is already in motion
7	4439264768	5/5/2011 1:12 AM	1	in	Break a leg
63		5/6/2011 6:23 PM	1	draft	the implementation seems to be working ok, no gold get though
63	4439264768	5/6/2011 6:30 PM	1	pending	the implementation seems to be working ok, no gold yet though
63	4439264768	5/6/2011 6:30 PM	1	out	the implementation seems to be working ok, no gold yet though
99		5/8/2011 1:47 AM	1	draft	software seems to be working, I was a little worried given the source and short timeline
155		5/8/2011 4:13 AM	1	draft	Got something for you, sample shortly
155		5/8/2011 5:31 PM	1	draft	Got something for you, sample shortly
155	4439264768	5/8/2011 6:05 PM	1	pending	Got some results, I think we need to up the fee, say double?
155	4439264768	5/8/2011 6:05 PM	1	out	Got some results, I think we need to up the fee, say double?
156	4439264768	5/8/2011 6:16 PM	0	in	You are joking, right? You can't seriously think about changing the deal now.
156	4439264768	5/8/2011 6:16 PM	1	in	You are joking, right? You can't seriously think about changing the deal now.
157	4439264768	5/8/2011 6:22 PM	1	pending	I just sent you a sample, I think you'll be pleased...
157	4439264768	5/8/2011 6:22 PM	1	out	I just sent you a sample, I think you'll be pleased...
158	4439264768	5/8/2011 6:30 PM	0	in	You are serious then. I can see the information is valuable but I am displeased with you breaking the deal.
158	4439264768	5/8/2011 6:30 PM	1	in	You are serious then. I can see the information is valuable but I am displeased with you breaking the deal.

159	4439264768	5/8/2011 6:56 PM	1	pending	I knew you'd like them, ill be at the agreed spot, in about 25 min for the exchange
159	4439264768	5/8/2011 6:56 PM	1	out	I knew you'd like them, ill be at the agreed spot, in about 25 min for the exchange

Table 13. Text messages exchanged with mr E

For a complete listing of the found text message records, the reader is referred to the report generated by *extractAndroidData* included.

6.10 Gmail communications

We used the *extractAndroidData* program to extract Gmail messages from the user data partition of the device. No Gmail communication resides on the other partitions or SD memory card.

In total 11 Gmail message records were recovered from the user data partition. The Gmail messages that do not originate from Google services are listed in Table 14. Some of the records could not be completely recovered because of the Sqlite overflow page construction[8]. In these cases the unknown column values are filled with the string ***overflow***.

It is not clear from the recovered information whether the Gmail messages have actually been sent. It is likely given the multiple copies of the emails that some of these messages are (automatically stored) draft versions. More reference tests are needed to conclude on these findings.

We find it very likely that emails were composed and possibly also sent from the email address *norby411@gmail.com* to *mre@hushmail.com*. The sender indicates that a sample is attached for review by the receiver and he hints to a higher price for the results.

Also we find it very likely that at least one email was received from mre@hushmail.com on 5/8/2011 on 2:43 PM (UTC-4). In this email the sender (very likely the contact mr E) expresses his worries about Norby giving 'the files' to someone else and says that Norby can expect a call from mr E shortly.

Id	From	To, cc, bcc	DateSentMs (UTC)	Subject	Body
2	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:08 PM	sample	this is just a taste, much more where this came from. N.
1	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:08 PM	sample	this is just a taste, much more where this came from. N.
3	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:20 PM	sample	this is just a taste, much more where this came from. N.

[8] This is the case when the records are too big to fit into a leaf page. We have not been able to recover data from overflow pages. For more information on this the reader is referred to the document on *extractAndroidData* included.

4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:32 PM	showing i'm serious	This information is obviously very valuable. I 'd like to keep our relationship, but these will fetch aÂ
4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:32 PM	showing i'm serious	This information is obviously very valuable. I 'd like to keep our relationship, but others would be willing to pay more. Â Here a
4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:34 PM	**overflow**	**overflow**
4	"norb k" <norby441@gmail.com>	"Mr E" <mre@hushmail.com>	5/8/2011 6:34 PM	**overflow**	**overflow**
5	"" <mre@hushmail.com>	"norb k" <norby441@gmail.com>	5/8/2011 6:43 PM	Re: showing i'm serious	-----BEGIN PGP SIGNED MESSAGE----- Hash: SHA1 I certainly don't want you giving these files to someone else. Expect a call from me shortly. On Sun, 08 May 2011 14:34:46 -0400 norb k <<a href="mailto:norby441

Table 14. Relevant Gmail communication

6.11 Other application data

We further used the *extractAndroidData* program to extract messages from the Google Talk database and Google Maps information messages from the user data partition of the device. We found no records of Google Talk chat messages or Google Maps searches or destinations.

6.12 Keyword search

We performed a keyword search using FTK 3.3 on the user data partition, cache partition and SD memory card. We defined keywords from the case premises and previous findings. The keywords used are listed in Table 15.

Keywords	Remark
50.56.29.109	IP-address of dir list
4124623802	Verizon store phone number
Kryptix	The name of organization
Cyph3r	See section 6.7.4
5854561283	Unknown Phone number on Taog's device (section 7.9)

Table 15. Keyword list

No relevant information was found other than which we report on in other sections of this report.

7 Investigation of Case 2

7.1 Relevant preliminary findings

7.1.1 YAFFS2 partitions

10 YAFFS partitions where acquired by the agent from the device. These are listed in table Table 16. The partitions that are most likely used to store user data are mtd7 (cache) and mtd8 (user data).

Since user data is only stored on the user data partition, cache partition and SD memory card, we have focused our investigation on these devices.

Device	Size	Partition type
Mtd0	660 KB	MBM
Mtd1	396 KB	CDT
Mtd2	396 KB	LBL
Mtd3	396 KB	MISC
Mtd4	3.696 KB	BOOT
Mtd5	4.752 KB	RECOVERY
Mtd6	148.236 KB	SYSTEM
Mtd7	97.812 KB	Cache
Mtd8	276.408 KB	Userdata
Mtd9	2.112 KB	Kernel panic

Table 16. YAFFS2 partitions acquired

7.1.2 Time and time zone information

The time zone information is stored on the user data partition in */property* in a file named *persist.sys.timezone*. From the file *persist.sys.timezone* we found that the last configured time zone is *America/New York* (UTC–4 in DST).

From the evidence collection report we could not confirm that the date and time settings were correct at the time of evidence collection. For the remainder of this report, we will work under the assumption that the date and time were set correctly.

7.2 Rebuilding the YAFFS2 file system

Before we began to examine the user data and cache partition we decided to extract the files out of the images. The files system on the images is the Yet Another Flash File system 2 (YAFFS2) file system. For more information on the YAFFS2 file system, the user is referred to section 5.1.

We first used the tool unyaffs2[9]. This tool is not working fine, it gives a lot of errors, only file listing is accurate. Another way is to rebuild a Linux kernel with YAFFS2 support. The next sections describe how to rebuild a Linux kernel with YAFFS2 support.

7.2.1 Rebuild the Kernel

We started with a 32bit Ubuntu 11.04 clean installation on a virtual machine.
Next, we downloaded the source YAFFS2 repository from
http://www.aleph1.co.uk/gitweb?p=yaffs2.git;a=summary.

In order to include the YAFFS2 support, additional packages are required. We downloaded some additional packages as follows:

[9] from yaffs2utils, http://code.google.com/p/yaffs2utils/

```
sudo apt-get -y install fakeroot build-essential crash kexec-tools
makedumpfile kernel-wedge git-core libncurses5 libncurses5-dev
libelf-dev libdw-dev binutils-dev

sudo apt-get -y install kernel-package linux-meta

sudo apt-get -y install kernel-package

mkdir -p  ~/dev/kernel
cd ~/dev/kernel
sudo apt-get build-dep --no-install-recommends linux-image-$(uname -
r)

apt-get source linux-image-$(uname  -r)
```

Next, we go into the yaffs2 source folder and patch the kernel using the *patch-ker.sh* script.

```
cd yaffs2
./patch-ker.sh l m /home/fox/dev/kernel/linux-2.6.32

cd ~/dev/kernel/linux-2.6.32
cp -vi /boot/config-$(uname -r) .config
```

Create menuconfig that prompts for all new config options:

```
make oldconfig
```

```
make-kpkg clean

nice fakeroot make-kpkg --initrd --append-to-version=-yaff --overlay-
dir=/home/fox/dev/kernel/linux-2.6.32 kernel-image kernel-headers
```

Now you have new packages that you can install in your Ubuntu install

```
sudo dpkg -I linux-*.deb
```

Reboot the machine and select to boot the alternate kernel with YAFFS2 support.

7.2.2 Mount the images

It is not possible to just mount the image as a mtdblock device.
We need a utility *nandsim* to emulate a NAND flash device, to write our images to.
It depends on the size of the original NAND flash what parameters need to be used to create a simulated NAND device. The parameters for the different NAND sizes are below:

```
modprobe nandsim first_id_byte=0x20 second_id_byte=0x33  (16MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x35  (32MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x36  (64MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x78  (128MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x71  (256MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0xa2 third_id_byte=0x00 fourth_id_byte=0x15
(64MiB, 2048 bytes page)
modprobe nandsim first_id_byte=0xec second_id_byte=0xa1 third_id_byte=0x00 fourth_id_byte=0x15
(128MiB, 2048 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0xaa third_id_byte=0x00 fourth_id_byte=0x15
(256MiB, 2048 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0xac third_id_byte=0x00 fourth_id_byte=0x15
(512MiB, 2048 bytes page)
modprobe nandsim first_id_byte=0xec second_id_byte=0xd3 third_id_byte=0x51 fourth_id_byte=0x95
(1GiB, 2048 bytes page)
```

```
Modprobe mtdblock
Modprobe yaffs
modprobe nandsim first_id_byte=0xec second_id_byte=0xbc
third_id_byte=0x00 fourth_id_byte=0x55 cache_file=/tmp/nandsim.bin
```

Now flash erase the simulated NAND device before writing the images back to the simulated NAND device. Don't forget the –r parameter if the image is made with *nanddump*. The images we have contain raw out of band bytes (oob).

```
flash_erase /dev/mtd0 0 4096
nandwrite –a –r /dev/mtd0 ~/DFRWS/mtd8.dd
```

Now we can mount the file system

```
Mount  /dev/mtdblock0 /mnt/case2/data

fox@server1104:/mnt/case2/data$ ls -l
total 27
drwxrwx--x 1      fox   fox     2048 2011-05-05 04:06 anr
drwxrwx--x 1      fox   fox     2048 2011-05-08 05:09 app
drwxrwx--x 1      fox   fox     2048 1970-01-01 01:02 app-private
drwx------ 1      fox   fox     2048 1970-01-01 01:02 backup
-rw-rw-rw- 1      root  root       8 2011-05-11 02:45 cc_data
drwxrwx--x 1      fox   fox     2048 2011-05-08 05:09 dalvik-cache
drwxrwx--x 1      fox   fox     2048 2011-05-08 05:09 data
drwxr-x--- 1      root  1007    2048 1970-01-01 01:02 dontpanic
drwxrwx--x 1      2000  2000    2048 1970-01-01 01:02 local
drwxrwx--- 1      root  root    2048 1970-01-01 01:02 lost+found
drwxrwx--t 1      fox   9998    2048 2011-05-11 02:45 misc
drwx------ 1      root  root    2048 2011-05-10 22:43 property
drwxrwxr-x 1      fox   fox     2048 2011-05-11 02:42 system
drwxr-xr-x 1      fox   fox     2048 2011-05-07 18:50 tombstones
```

And then we have a readable logical file system to do our investigation on.

7.3 Installed applications

We have investigated the device for installed applications. Background information on Android and applications installations can be found in section 5.2.

A python program was created and used to parse the registered packages in the *packages.xml* file. The algorithm of the program is summarized in Listing 7.

```
Look for a file named packages.xml
 Attempt to parse the file as an xml element.
If this succeeds:
  Parse out any <package … /> elements.
For each package element:
     Parse the xml element attributes of the element.
     Extract the name, codepath, flags,  version, uid and date/time.

 Output the package registries to a html file.
```

Listing 7. Outline analysis program

For a complete list of the resulting packages the reader is referred to the included report generated by the *analyzeAndroidData* program.

7.4 SD-card analysis

We examined the SD memory card using Forensic Toolkit 3.3. The file system on the SD memory card is the FAT32 file system. The file system was examined for existing and deleted files and folder structure. Also, the SD card was carved for files of all relevant types.

In the root of the file system 9 PDF files were found. These files are listed in Table 17 and were created on 5/7/2011 between 12:54 AM and 1:17 PM (UTC-4).

Name	L-Size (bytes)	Created (UTC-4)	MD5
2201-4.pdf	29541	5/7/2011 1:13:46 PM	A145BA75735B5ACC9C43AA2759C9B126
2201-7.pdf	42881	5/7/2011 1:14:20 PM	4AA76F74ADA38E97D9D7113EF8E3C44E
2201-8.pdf	52359	5/7/2011 1:14:48 PM	0A36E386F6F0FED84E80850739C96174
2201-9.pdf	46343	5/7/2011 1:16:10 PM	3F3A48026B33E093FF841852F7AF20BC
2228-10.pdf	136844	5/7/2011 1:17:14 PM	0F899A47B55289FCF1D6DA9915183A69
2228-11.pdf	260786	5/7/2011 12:54:02 PM	44A39F3F3E57DC50F4EBC04D0F9ADB00
2228-12.pdf	48113	5/7/2011 1:10:00 PM	EB8FDF32EB18598F931D23E703D8A3BD
2228-15.pdf	47157	5/7/2011 1:10:34 PM	A7D03C5CA92A5913E6B929FE94FA96F2
2228-7.pdf	177047	5/7/2011 1:16:58 PM	197620727BB96ECBBC8AF62BB22107DE

Table 17. PDF files on SDcard

On the SD memory card in *DCIM/Camera* deleted photo's are present which are likely taken by the camera on a Motorola droid, according to traces in the EXIF data. Multiple pictures are taken around and in a bar named 'fat heads' Figure 3. These are taken on 5/5/2011 between 7:05 PM (UTC-4) and 7:48 PM (UTC-4).

```
Exif.Image.Make:Motorola
Exif.Image.Model:Droid
Exif.Image.Software:2.0.1
```

Figure 3. Fat Heads

By open source investigation traces are found that it is most likely that these pictures are taken on the East Carson Street, Pittsburgh, Pennsylvania, United states.

On 5/6/2011 between 6:29 and 6:39 PM (UTC-4) multiple photos were taken using the Motorola droid somewhere in a garden, what looks like it could be a garden expo.

7.5 Carving the cache partition

Using both FTK 3.3 and Photorec we carved the cache partition for all relevant files of the following categories:

- Archive files
- Image files
- Audio/video files
- Documents
- HTML and XML files

No relevant files were found.

7.6 Carving the user data partition

Using both FTK 3.3 and Photorec we carved the user data partition for files of all relevant files of the same categories as mentioned above.

No relevant files were found which are not already found by other methods discussed further on.

7.7 Location information

Android may store location information in the file */data/com.google.android.location/cache.cell* or *cache.wifi* files. These files hold information on the GSM cell or the WIFI access point the phone was connected to. When the Android phone is first initialized the user is asked if the user wants to send and use anonymized location information on the phone. If this is option not enabled, the phone will not store this location information.

On the user data partition the files *cache.cell* and *cache.wifi* are present. Due to time constraints, we decided not to analyze these files.

The browser sometimes stores location information in a SQLite database. This database is located on the user data partition in the directory */data/com.android.browser/app_geolocation/cachedpositions.db.* The database holds the information displayed in Table 18.

latitude	longitude	altitude	accuracy	Timestamp (UTC-4)
40,50	-80,25	390,80	8,00	5/8/2011 6:39 PM

Table 18. Cached positions

These coordinates resolves to Pittsburgh International Airport (PIT), Imperial, PA 15126, United States.

7.8 Browser information

We examined the browser history, bookmarks and credentials on the user data partition. None of these traces reside on the other partition or on the SD memory card.

In the SQLite database *webview.db* on the user data partition we have found credentials for the twitter profile of Yob Taog. The database is located in the directory *data/com.android.browser/databases/webview.db*. The credentials are listed in Table 19.

Host	Username	Password
httpsmobile.twitter.com	yobtaog@gmail.com	aaassspp

Table 19. Twitter credentials

No further relevant browser information was found. The complete browser information is listed in the *analyzeAndroidData* report included.

7.9 Contact information

Contact information is stored in an SQLite database. The contacts database is located on the user data partition in the directory *data/com.android.providers.contacts/databases/contacts2.db.* The contact information found is listed in Table 20.

Name	E-mail	Phone number
Reg Wetham	regwetham@yahoo.com	
Shandra Pfeif	shandra@cheerful.com	
Luke Lancer	luk3lancer@gmail.com	
Hersolv Einskavich	hersolv@gmail.com	
Adrian		7607058888
Swiftlogicllc@consultant.com	swiftlogicllc@consultant.com	
Swiftlogicinc@consultant.com	swiftlogicinc@consultant.com	
Swift Logic	swiftlogic@consultant.com	

Table 20. Contact information

Call history can be found on the user data partition in an SQLite database in the location */data/com.android.providers.contacts/databases/contacts2.db*. All the information found in the call history is being displayed in table 5.

Name	Timestamp (UTC-4)	Duration	type	Name
5854561283	05/06/2011 12:51:31 PM	20	In	
7607058888	05/06/2011 1:04:08 PM	110	In	Adrian
7607058888	05/06/2011 1:17:53 PM	127	In	Adrian
7607058888	05/06/2011 3:25:22 PM	0	out	Adrian

Table 21. Call history

7.10 Text messages

SMS/MMS messages can be found on the user data partition in an SQLite database in the location */data/com.android.providers.telephony/databases/mmssms.db.* No text messages reside on the other partitions or SD memory card.

In total 17 SMS/MMS messages were found. Four messages have been sent and thirteen messages have been received. The received messages are all read. The first SMS/MMS messages is received 5/5/2011 9:34 PM (UTC-4) the last SMS/MMS message is received 5/10/2011 4:43 PM (UTC-4). Only for the case relevant messages are listed down below in Table 22. For a complete list of text messages the reader is referred to the *analyzeAndroidData* report include with the results.

From	Timestamp (UTC-4)	type	Message
sms.dynadel@gmail.com	05/06/2011 05:53:30 PM	In	Reminder, planned IT outage this weekend. This maintenance window will start at 3 PM today and continue for approx 48 hours.
sms.dynadel@gmail.com	05/06/2011 05:55:16 PM	In	This effects external services such as website, email, webmail, and the ftp server. Use the secondary email access and helpdesk # for emergencies

Table 22. SMS/MMS messages

These messages are relevant because of the IT outage Yob gets a couple of PDF files e-mailed that he needs for a presentation. Section 7.11 further elaborates on the Gmail messages.

7.11 Gmail communications

We have examined the user data partition for traces of Gmail communication. Gmail messages can be found on the user data partition in an SQLite database in the location *data/com.google.android.providers.gmail/databases/mailstore.yobtaog@gmail.com.db*. No Gmail

communication resides on the other partitions or SD memory card. Only the relevant communications to this case are being displayed.

In total forty e-mail messages were found in the database. The first message in the database is from 05/06/2011 3:35 PM (UTC-4), the last message in the database is from 05/10/2011 8:44 PM (UTC-4). For a complete list of communications the reader is referred to the *analyzeAndroidData* report included with the results.

Id	From	To	Sent (UTC)	Message (snippet)	attachment
42	yobtaog@gmail.com	swiftlogic@consultant.com swiftlogicllc@consultant.com swiftlogicinc@consultant.com	05/06/2011 07:35:05 PM	File request: helpdesk, I was unaware of the server outage starting today and need some fil...	
48	swiftlogic@consultant.com	yobtaog@gmail.com	05/06/2011 11:11:04 PM	Re: File request: Mr Taog- My apologies, we kind of have our hands full down here with the main...	2228-11.pdf 2228-12.pdf 2228-15.pdf
49	yobtaog@gmail.com	swiftlogic@consultant.com	5/07/2011 12:29:18 AM	Re: File request: Tim, Sheets 7 and 10 should have also been included in that timeframe... Also...	
50	swiftlogic@consultant.com	yobtaog@gmail.com	05/07/2011 12:40:49 PM	Re: File request: Mr Taog- It looks like Tim found your files, but he just went out for breakfa...	2201-4.pdf 2201-7.pdf 2201-8.pdf 2201-9.pdf 2228-7.pdf 2228-10.pdf
65	yobtaog@gmail.com	swiftlogic@consultant.com	05/08/2011 6:35:14 PM	More files please: Can you please send me all the sheets for project 2228? Thanks, -yob	
75	regwetham@yahoo.com	Yobtaog@gmail.com	05/10/2011 11:24:19 AM	How is Atlanta: I haven't heard back about your presentation and meetings. Hope all is going well and that clients like what you are showing them. It's some of SwiftLogic's best.	
80	yobtaog@gmail.com	regwetham@yahoo.com	05/10/2011 08:34:28 PM	Re: How is Atlanta: Went well! I'll have to give you the details when I get back. … I'm a little pissed off at the tech team at the consultant address, they never got back to me about some additional sheets I wanted for today - they just never responded.... iknow they can because they did last week.	
82	yobtaog@gmail.com	regwetham@yahoo.com	05/11/2011 8:44:45 AM	Re: How is Atlanta: I just tried to stop by work, my keycard doesn't work! How do you like that? ...	

Table 23. Gmail messages

It is likely Yob received the nine PDF files (table 6, message #48, #50) because of the planned IT outage. He likely needed them to prepare for a meeting after the weekend. This becomes clear of the discussion he has with his friend/colleague Reg (table 6, message #75, #80).

On the user data partition we have found a database that hold the credentials for the Gmail account of Yab Taog. The credentials are listed in Table 24. The database is located on the user data partition in the directory *data/com.android.email/databases/emailprovider.db*

address	port	login	password
imap.gmail.com	993	yobtaog@gmail.com	aaassspp
smtp.gmail.com	465	yobtaog@gmail.com	aaassspp

Table 24. Gmail credentails

7.12 Social media

7.12.1 Twitter

We found that the user installed an application called *com.seesmic* (section 7.3) which is an application used among others to post and read Twitter messages. We examined the Twitter messages. These messages are found on the user data partition in an SQLite database found at *data/com.seesmic/databases/twitter.db* and on the twitter profile of Yob (http://twitter.com/#!/yob_taog). Only the relevant messages are displayed below in Table 25.

Sender	Date/time (UTC-4)	Message
yob taog	5/4/2011 21:43 PM	Yay! Just picked up my new android smartphone! !!!
yob taog	5/6/2011 1:41 PM	Holy crap, just found out there is a planned maintenance outage starting this afternoon! Time to go tell the IT depth to reschedule...
yob taog	5/6/2011 1:49 PM	now i get the txt about the outage. great. thanks guys. http://wiep.net/talk/wp-content/uploads/2008/12/it-dept.jpg
yob taog	5/8/2011 6:32 PM	waiting for a plane at PIT, wish it was still a hub (even though I happen to have a direct flight today)

Table 25. Twitter messages

It's very likely that from the time 5/4/2011 9:43 PM (UTC-4) until the device was in the possession of Yob.

7.12.2 Facebook

In the browser history we found traces of use of a Facebook account which most likely is used by Yob (http://www.facebook.com/profile.php?id=100002336995096). On his facebook account several messages were found posted. Only the relevant messages are displayed in Table 26.

Timestamp (UTC-4)	Message
5/4/2011 11:50 AM	Well guys, I'm going to take the plunge. I have decided to get a BRAND NEW Motorola Droid. More Android goodness !!!! I have called my favorite Verizon store at the Pittsburgh Waterfront, and checked that *they have plenty in stock.* I will be heading out there a little later to pick up that bad boy. Oh yes.

Table 26. Facebook message

This message is relevant because in here Yob states where and when he is going to pick-up his new phone.

7.13 Keyword search

From the findings from the investigation on Norby's (case1.) device we created a keyword list to search the device of Yob. The findings in the Norby indicated that the two cases may be linked.

We defined a keyword from the findings we made on Norby's device because. We did this because Norby's device beholds PDF files that probably belong to Yob. The keywords are listed in Table 27.

Keywords	Remark
50.56.29.109	IP-address of dir list
4124623802	Verizon store Phone number

Table 27. Keyword list

The keyword originates from carved HTML pages out of Case 1 (section 6.5). These HTML pages very likely originated from: *http://50.56.29.109/ss*, the pages show an open directory listing with PDF files. The names of the PDF files in this listing are the same as the PDF filenames found on Yob's device.

The keyword search performed by us in FTK 3.3 revealed that the IP-address *50.56.29.109* is present on the phone of Yob. This keyword is found in the file:

/data/dalvik-cache/data@app@com.andriod.mm.apk@classes.dex.

This is a cache file from the *Dalvik VM*[10] that is likely used to store application code and data (variables, constants, etc) for the application named *com.andriod.mm*. This lead us to further examine the workings of this application. This is described in section 7.14.

7.14 Com.Andriod.MM and Com.VZW.smsprovider

7.14.1 Com.Andriod.MM

According to packages.xml the application *com.andriod.mm* was installed on 5/4/2011 on 20:49 PM (UTC-4).

This application is not listed in the Android market and later on in the investigation a string search on the IP-address *50.56.29.109* lead us to the file *data@app@com.andriod.mm.dex*. This file is located on the user data partition in the *data/dalvik-cache* folder. Apparently the string is somehow used by this application. Reverse engineering the APK file that was found on the user data partition in the folder *apps/com.andriod.mm* provided us with the following algorithm. We reverse engineered the APK as follows:

1. Copy the APK file from the user data partition and unzip the APK file.
2. Use dex2jar[11] to convert the dex file to a jar file
3. Use jd-gui[12] to decompile the java bytecode.
4. Use apktool[13] to properly decode the AndroidManifest.xml files

[10] This is the virtual machine layer on top of which all applications are executed. The Dalvik VM creates a sandboxed environment for an application and holds program code and data in cache.
[11] http://code.google.com/p/dex2jar/
[12] http://java.decompiler.free.fr/?q=jdgui

AndroidManifest.xml

The manifest file describes the permissions of an application and the triggers it responds to.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0"
package="com.andriod.mm"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="4" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission
android:name="android.permission.PROCESS_OUTGOING_CALLS" />
    <application android:label="@string/app_name"
android:icon="@drawable/icon" android:debuggable="true">
        <receiver android:label="@string/app_name"
android:name="com.andriod.mm.bootComp">
            <intent-filter>
                <action
android:name="android.intent.action.AIRPLANE_MODE_CHANGED" />
                <action
android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="android.intent.action.SCREEN_OFF" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.andriod.mm.callOut">
            <intent-filter>
                <action
android:name="android.intent.action.NEW_OUTGOING_CALL" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.andriod.mm.callIn">
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE"
/>
            </intent-filter>
        </receiver>
        <service android:name="com.andriod.mm.mediaMounter"
android:enabled="true" android:exported="true" />
    </application>
</manifest>
```

Permissions and triggers

According to the manifest file this application has permission to:

- Use the internet interface
- Receive an event when boot completes
- Read the phone state
- Process outgoing calls

The applications listens on the following events (triggers):

[13] http://code.google.com/p/android-apktool/

- Airplane mode changes, boot sequence completes and screen is turned off. The *bootcomp* class handles these events
- An outgoing call is placed. The *callout* class handles this event.
- The phone state changes (incoming call mostly). The *callIn* class handles this event.

Next, we explain in short the workings of the various Java classes of the application. The most relevant strings are made bold in the description.

Class bootComp

It activates when any of the events is received: boot completed, airplane mode changed or screen is turned off. The service class com.andriod.mm.mediaMounter is started.

Class callIn

It activates when an incoming call is received. Str = "CallIn: " + <incoming number> + <date/time> Send str by SMS using **com.vzw.smsProvider**(.ACTION_SEND intent).

Appearantly the application *com.andriod.mm* makes use of another application *com.vzw.smsProvider* which is not a standard Android application. The program *com.vzw.smsProvider* is explained in section 7.14.2.

Class callout

It activates when an outgoing call is placed. Str = "Callout: " + <outgoing number> + <datetime> Send str by SMS using com.vzw.smsProvider(.ACTION_SEND intent).

Class MediaMounter$1:

Extends *TimerTask* and appears to call *MediaMounter.doStuff(*) every hour.

Class MediaMounter

OnCreate(): SendMSG("**service started**"); doStuff()
OnStart: DoStuff()
Do Stuff(): Files = getFiles(external storage directory); Zip the files and write them to 'temp' sendFile('temp' file) if(sendFile is ok):

sendMSG("**pkg uploaded!**")
GetFiles(folder): Loop through folder and add files to array. Return the array.
SendFile(file): Create TCP socket to **50.56.29.109:10001**, Write the file to the socket outputstream If host is unreachable: sendMSG("connect failed (start server!)" + exception message); If host is unreachable: sendMSG("connect failed (start server!)" + exception message);
sendMSG(msg): prepend "**vzwsms://message/**" to msg and convert to URI. Use Android's intent infrastructure to call the ACTION_SEND of the application com.vzw.smsProvider. Provide URI.

To summarize the workings: The application reads files from the SD memory card and sends the file to a server on IP-address *50.56.29.109* through port 10001. Further it monitors incoming and outgoing calls and reports on this through *com.vzw.smsProvider*. The application is activated on boot, screen off/on and incoming calls.

We have performed a port scan on the server behind IP-address *50.56.29.109* using Zenmap[14]. The goal was to find if the server still listens on port 10001 or other ports. The results shows that the server at IP-address *50.56.29.109* is no longer listening on port 10001. The only ports listening on is port 80 (http) and port 21 (ftp). We attempted to login on ftp using the credentials found on Norby's device (section 6.7.4), but no communication with the FTP server succeeded.

7.14.2 Com.VZW.smsprovider

We performed the same steps to analyze the workings of this application as for the application *com.andriod.mm*.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0"
package="com.vzw.smsProvider"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="6" />
    <application android:label="@string/app_name"
android:icon="@drawable/icon" android:debuggable="true">
        <receiver android:name=".sendSMSRec">
            <intent-filter>
                <action android:name="com.vzw.smsProvider.ACTION_SEND" />
                <data android:scheme="vzwsms" />
```

[14] Zenmap is a GUI version of Nmap, a well-known port-scanning progam. The software can be downloaded from http://nmap.org/zenmap/.

```
            </intent-filter>
        </receiver>
        <receiver android:name="com.vzw.smsProvider.SMSRec">
            <intent-filter android:priority="100">
                <action
android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
        <service android:name=".smsServiceProvider" android:enabled="true"
/>
    </application>
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
</manifest>
```

Permissions and triggers

According to the manifest file this application has permission to:

- Send SMS messages
- Monitor received SMS messages

The applications listens on the following events (triggers):

- Receiving an SMS
- *com.vzw.smsProvider.ACTION_SEND*. A custom event which is used by *com.andriod.mm* to send messages.

Next, we explain in short the workings of the various Java classes of the application. The most relevant strings are made bold in the description.

sendSMSRec

Extends *BroadcastReceiver* intended to receive intents[15] from other applications.

```
onReceive(uri):
 Converts the URI provided by the intent to a string str
 smsLib.sendkSMS(str)
```

SMSRec

```
onReceive(intent):
 only runs when intent action is “android.provider.telephony.SMS_RECEIVED” (when a SMS is
received on the phone)
 get all the received messages and  for each message:
```

[15] Intents is a termed used by the Android API for signals that can be passed by other applications. It is a mechanisms for applications to communicate with each other.

```
   str = message + "FORWARDED SMS from " + originating address + " at " + current date/time + " : ").
   smsLib.sendkSMS(str)
```

smsLib

```
sendkSMS(string1):
  sendkSMS("14124393389", string1);
```

```
sendkSMS(string1, string2):
  prepend string2 with "ksms";
  use default local SMS manager to send the message to string1.
```

To summarize the workings: This applications on command sends a specific text message or all the received text messages to phone number *14124393389*. The messages are prepended with the string *'ksms'.* In section 6.9 it is described how many text messages were received on Norby's device according to this format. Therefore we find it most likely that this phone number was in use by Norby.