iPhone Forensics, sans iPhone

Adam Crosby (adam@uptill3.com)

March 14, 2010
Background Info
iTunes Sync

What happens when you plug in...
OS Image

- The OS image for an iPhone is NOT backed up on each sync
- The image is only backed up when the phone software is upgrade
- The image only changes between major versions - no ‘patches’ for iPhone OS.
- Does not contain ANY user apps or data
Apps / User Data

- Initial sync with iTunes creates an archive on the computer of all of the user apps/data from the phone
  - The files contained in the archive are all obfuscated
  - Virtually none of the files or data are encrypted (!)
  - Files are updated in place with every sync of the phone
  - This archive is used to recover the phone if needed
Obfuscated Information
That’s a lot of files...

And none of those names are meaningful.

The Archive Is Located At: ~/Library/Application Support/MobileSync/Backup/<phone Id>/
What kind of Data?

A whole bunch of different kinds
iPhone uses 3 main ‘standard’ plist files

- Info.plist, Manifest.plist, and Status.plist
- Info.plist is by far the most interesting (more later)

- These are plaintext XML documents
- Easily read, not always easy to decipher
- May contain substantial Base64 encoded <data> chunks
- Open with text editor, or plistlib in Python 2.5+ (yay!)

Info.plist example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Build Version</key>
    <string>7D11</string>
    <key>Device Name</key>
    <string>Adam's Iphone</string>
    <key>Display Name</key>
    <string>Adam's Iphone</string>
    <key>GUID</key>
  </dict>
</plist>
```
‘mdinfo’ --> Apple bplist files

- Special kind of plist - ‘binary packed’
- Luckily, Apple provides an easy to use OS X utility called `plutil`
  ```
  plutil -convert xml1 <filename>
  ```
- Converts the bplist file into a standard plist **in place** (so copy it out of the live archive folder before digging in.
- Many of the bplist files have Base64 encoded `<data>` property values that are themselves bplists
  ```
  The SMS db bplist file is like this.
  ```
  ```
  Just unencode it and run it through `plutil` again to get the embedded plist data out
  ```
‘mddata’ --> The goods

- The mddata files are simply renamed copies of whatever the original source file described by the mdinfo file was. There are images, databases, text chunks, and application specific stuff (such as MOBI ebooks).

- No formatting changes - you can open the PNGs/JPGs as-is and look at the pictures

- Any other tools (such as sqlite3) can open files of the correct type in place as well

- ‘file *.mddata’ in the archive will give a great listing of each file and it’s actual data type
File Names
Crazy file names (learn to love SHA1)

- iPhone OS storage is broken into something called ‘Domains’
- Each file has a file name, path, and a particular storage domain
  - Two primary domains are ‘HomeDomain’ and ‘MediaDomain’
- The filenames of the mddata and mdinfo files are a SHA1 hash of the full path of the files on-handset location and the domain it is located in.
- Filenames do not appear to change, but worst case scenario is iterating over mdinfo files to find the actual file name you need
printf('MediaDomain-Library/SMS/Parts/98/02/15874-4.jpg') | shasum 39cf2a2aaa3dd7d72243e3d638217ccc15ad2575
So what’s in there?
Info.plist - All about the phone

- This is an unencoded, standard Apple Property List file (XML text)
- It contains a wealth of useful information about the phone:
  - ICC-ID - “Integrated Circuit Card ID”, the hardware serial number of the installed SIM card
  - IMEI - “International Mobile Equipment Identity”, the hardware serial number of the handset’s baseband proc [*]
  - Phone Number - Duh.
  - Serial Number - The iPhone’s serial number (this shows in iTunes)
  - Product Type - What kind of iPhone (‘iPhone2,1’ = 8GB 3GS, ‘iPhone1,2’ = 8GB 3G, etc.)
  - Product Version - What iPhone OS revision (3.1.3, 3.1.2, etc.)
  - Data of Last backup - Duh.
  - iPhone preferences plist (base64 encoded embedded plist)
  - Misc. other stuff (encoded / binary application specific data)
A note about IMEI and ICC-ID

- ICC-ID is burned into the SIM
- IMEI is burned into the phone
- It is illegal (and technically difficult) to change either the ICC-ID or the IMEI
- The ICC-ID identifies which network/carrier sold the SIM
- The IMEI can be used to uniquely identify a given handset, and is used by carriers to disable stolen handsets, even if a new SIM is swapped in.
- Neither the IMEI nor the ICC-ID is tied to the subscriber account at the GSM protocol level (that would be the IMSI, which is NOT recorded anywhere but in the SIM)
- ATT or other carriers and Apple may be able to collaborate with LE to determine a subscribers identity via ICC-ID, IMEI and Apple ID, as the information does exist
SQLite - Apple loves it, so should you

- Open Source, file-based database engine
- Apple uses it extensively across their platforms and application ranges - most iPhone devs know it as ‘CoreData’
- Simple relational database, supports most of SQL-92
- `sqlite3` is the OS X built-in CLI to access SQLite database files
- In the CLI, the `.schema` command shows database and table schemas
3d0d7e5fb2ce288813306e4d4636395e047a3d28.mddata
(also known as HomeDomain-Library/SMS/sms.db)

- SQLite database
- Contains full SMS records (including full text) for phone, since owner originally created an iPhone account (my DB goes back to 2007)
- Updated in place during sync - deleted messages are removed, new messages are inserted
- Does NOT contain MMS info (although receipt of an MMS is indicated)
- Simple data structure for messages table
## SMS message table schema

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWID</td>
<td>INTEGER (PK)</td>
</tr>
<tr>
<td>address</td>
<td>text</td>
</tr>
<tr>
<td>date</td>
<td>integer</td>
</tr>
<tr>
<td>text</td>
<td>text</td>
</tr>
<tr>
<td>flags</td>
<td>integer</td>
</tr>
<tr>
<td>replace</td>
<td>integer</td>
</tr>
<tr>
<td>svc_center</td>
<td>text</td>
</tr>
<tr>
<td>group_id</td>
<td>integer</td>
</tr>
<tr>
<td>association_id</td>
<td>integer</td>
</tr>
<tr>
<td>height</td>
<td>integer</td>
</tr>
<tr>
<td>UIFlags</td>
<td>integer</td>
</tr>
<tr>
<td>version</td>
<td>integer</td>
</tr>
<tr>
<td>subject</td>
<td>text</td>
</tr>
<tr>
<td>country</td>
<td>text</td>
</tr>
<tr>
<td>headers</td>
<td>blob</td>
</tr>
<tr>
<td>recipients</td>
<td>blob</td>
</tr>
<tr>
<td>read</td>
<td>integer</td>
</tr>
</tbody>
</table>
Address is either source or destination phone number, in 11-digit intl. format (18885551212)

Time is in UNIX epoch format (# of seconds since 1/1/70 0:00 UTC)

Easy to convert - ‘date -r 1268603160’ yields “Sun Mar 14 17:46:00 EDT 2010”

Flags field is used to indicate a number of things (these are all trial/error, not documented):

‘2’ = Message received from ‘address’

‘3’ = Message sent from handset to ‘address’

‘33’ = Message send failure (SMS never sent)

‘35’ = Message send failure (SMS never sent) (I think this also indicates a retry)

‘129’ = Message deleted (but still appears as a row - no contents though)

‘131’ = Unknown - no address / etc.

‘Text’ field is sometimes a plist - this typically indicates an MMS was received. MMS text/image information is stored separately (in mddata files outside of the SQLite db).

The other fields appear unused or static (or not of great interest...)
3rd Party Application example

SQLite database (hey, everybody uses it - thanks CoreData!)

Contains Facebook Friends List and a tiny bit of extra info

Facebook UID

www.facebook.com/profile.php?id=<fbid>

Direct URL to facebook profile picture (no login needed!)

Friends phone numbers, as listed in their profile
# Facebook App table schema

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>letter</td>
<td>char(1)</td>
</tr>
<tr>
<td>uid</td>
<td>bigint</td>
</tr>
<tr>
<td>first_name</td>
<td>char(32)</td>
</tr>
<tr>
<td>last_name</td>
<td>char(32)</td>
</tr>
<tr>
<td>name</td>
<td>char(64)</td>
</tr>
<tr>
<td>pic_square</td>
<td>char(200)</td>
</tr>
<tr>
<td>phone</td>
<td>char(25)</td>
</tr>
<tr>
<td>cell</td>
<td>char(25)</td>
</tr>
</tbody>
</table>
What else is there?
## Other identified databases

<table>
<thead>
<tr>
<th>Filename</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>992df473bbb9e132f4b3b6e4d33f72171e97bc7a.mddata</td>
<td>Voicemail List</td>
</tr>
<tr>
<td>ff1324e6b949111b2fb449ecddb50c89c3699a78.mddata</td>
<td>Call log</td>
</tr>
<tr>
<td>3d0d7e5fb2ce288813306e4d4636395e047a3d28.mddata</td>
<td>SMS Data</td>
</tr>
<tr>
<td>740b7eaf93d6ea5d305e88bb349c8e9643f48c3b.mddata</td>
<td>Notes’ App data</td>
</tr>
<tr>
<td>31bb7ba8914766d4ba40d6dfb6113c8b614be442.mddata</td>
<td>Sync’d Address book</td>
</tr>
<tr>
<td>6639cb6a02f32e0203851f25465fffb89ca8ae3fa.mddata</td>
<td>Facebook App data</td>
</tr>
<tr>
<td>970922f2258c5a5a6d449f85b186315a1b9614e9.mddata</td>
<td>Flightstats Info</td>
</tr>
<tr>
<td>5ad81c93601ac423bc635c7936963ae13177147b.mddata</td>
<td>Daily Burn food logger</td>
</tr>
</tbody>
</table>
Other items to look for

- App developers assume these files are on the phone, away from prying eyes (or interested users...)
- Passwords and other account details stored in plaintext
- At least Tweetie and Dailyburn store passwords in plaintext
  - (See 87f665a66ead44fd5592fa427c4def228098fdda.mddata for Tweetie’s twitter account information in bplist format)
- Cookies for some apps embedded browsers (see: Facebook) are stored in bplist files (for easier session hijacking, yay!)
What’s next?

* Time to start trolling through the app store for ‘service’ apps - GMail, GReader, MySpace, etc. to look for more apps that store their user creds in plaintext

* Scripting out data extraction for interesting stuff (like finding out hot numbers - who is called / calls the most, and do they have a Facebook picture?)

* Integrating a scrape of the address book with the other data tables to give ‘friendly names’ to each phone number

* Release existing tools open source via 757Labs

Saturday, March 20, 2010
Thanks / Credits

✦ 757 Labs for hosting!

✦ ‘dsp’ on LiveJournal for finally figuring out how the sha1’s are generated for file names

✦ Apple, for making CoreData so attractive for developers. SQLite makes trolling through data easy :)

✦ Damon Durand, for some initial work on the meaning of SMS flags
¿Preguntas?