iPhone Forensics - On iOS 5

The goal of iPhone Forensics is extracting data and artefacts from iPhone without altering the information on the device.

iPhone forensics can be performed on the backups made by iTunes (escrow key attack) or directly on the live device. This article explains the technical procedure and challenges involved in extracting data from the live iPhone.

iPhone 4 GSM model with iOS 5 is used for forensics.

Researchers at Sogeti Labs have released open source forensic tools (with the support of iOS 5) to recover low level data from the iPhone. Below details outline their research and gives an overview on usage of iPhone forensic tools.

Steps involved in iPhone forensics:

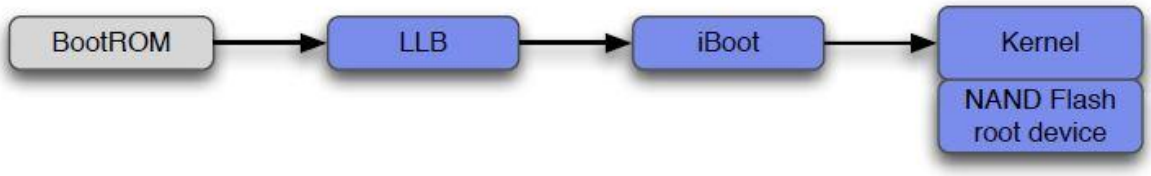
- Creating & Loading forensic toolkit on to the device without damaging the evidence
- Establishing a communication between the device and the computer
- Bypassing the iPhone passcode restrictions
- Reading the encrypted file system
- Recovering the deleted files

1. Creating & Loading forensic toolkit

Imagine a computer which is protected with OS level password - we can still access the hard disk data by booting a live CD or by removing the hard disk and connecting it to other machine. When we compare computers to the iPhone, it is an embedded device. So it is not easy to take out the chips (hard disk) and dump data in it. To perform iPhone forensics, we use Live CD approach. As the iPhone has only one serial port, we are going to load custom OS over USB to access hard disk of the device. But the problem here is, iPhone only loads the firmware which is signed by Apple.

In order to create and load forensic toolkit, first we need to understand iPhone functions at operating system level. iOS (previously known as iPhone OS) is the operating system that runs on all Apple devices like iPhone, iPod, Apple TV and iPad. iOS is a zip file (ships with .ipsw extension) that contains boot loaders, kernel, system software, shared libraries & built in applications.

When an iPhone boots up, it walks through a chain of trust which is a series of RSA signature checks among software components in a specific order as shown below.

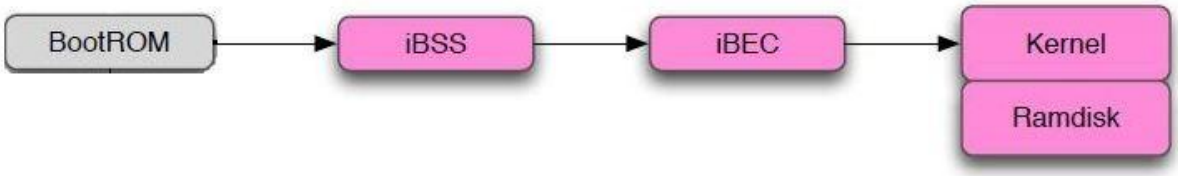


The BootRom is Read only memory (ROM) and it is the first stage of booting an iOS device. BootRom contains all the root certificates to signature check the next stage.

iPhone operates in 3 modes – Normal Mode, Recovery Mode, DFU mode

In Normal mode, BootRom start off some initialization stuff and loads the low level boot loader (LLB) by verifying its signature. LLB signature checks and loads the stage 2 boot loader (iBoot). iBoot signature checks the kernel & device tree and kernel signature checks all the user applications.

In DFU mode, iPhone follows the boot sequence with a series of signature checks as shown below. BootRom signature checks the second level boot loaders (iBSS, iBEC). Boot loader signature checks the kernel and kernel signature checks the Ramdisk.



During iOS update, Ramdisk gets loaded into RAM and it loads all other OS components. In Forensics, we will create a custom Ramdisk with all our forensic tool kit and load it on iPhone volatile memory. Signature checks implemented at various stages in the boot sequence does not allow to load our custom Ramdisk. To load our custom Ramdisk we have to bypass all these signature checks. In the chain of trust boot sequence, if we compromise one link, we can fully control all the links that follow it. The hacker community have found several vulnerabilities in BootRom using which we can flash our own boot loader and patch all other signature checks in all the subsequent stages. Apart from signature checks, every stage is also encrypted. These encryption keys can be grabbed from JailBreaking tools.

Building custom Ramdisk

First we will build a custom ram disk with all our forensic tools and patch the ram disk signature checks in kernel. Later, we use jailbreak tools to load our kernel by patching BootRom signature checks.

With the open forensic toolkit released by Sogeti Labs, we can build Ramdisk only on MAC OS X. The entire forensic toolkit contains python scripts, few binaries and few shell scripts.

In order to run the tools, first we need to install all the dependencies (Use the below listed commands from OS X terminal).

Download ldid, grant execute permissions and move it to /usr/bin directory.

```
curl -O http://networkpx.googlecode.com/files/ldid
chmod +x ldid
sudo mv ldid /usr/bin/
```

Download and install OSXFuse.

```
curl -O -L https://github.com/downloads/osxfuse/osxfuse/OSXFUSE-2.3.4.dmg
hdiutil mount OSXFUSE-2.3.4.dmg
sudo installer -pkg /Volumes/FUSE\ for\ OS\ X/Install\ OSXFUSE\ 2.3.pkg -target /
hdiutil eject /Volumes/FUSE\ for\ OS\ X/
```

Download & install python modules - pycrypto, M2crypto, construct and progressbar.

```
sudo ARCHFLAGS='-arch i386 -arch x86_64' easy_install pycrypto
sudo easy_install M2crypto construct progressbar
```

Download and install Mercurial (http://mercurial.selenic.com/) to check out the source code from the repository.

```
hg clone https://code.google.com/p/iphone-dataprotection/
cd iphone-dataprotection
make -C img3fs/
```

Download redsn0w to fetch encryption keys to decrypt Ramdisk and Kernel.

```
curl -O -L https://sites.google.com/a/iphone-dev.com/files/home/redsn0w_mac_0.9.9b8.zip
unzip redsn0w_mac_0.9.9b5.zip
cp redsn0w_mac_0.9.9b5/redsn0w.app/Contents/MacOS/Keys.plist .
```

To patch the signature checks in kernel, supply iOS 5 ipsw file to kernel_patcher.py

```
python python_scripts/kernel_patcher.py IOS5_IPSW_FOR_YOUR_DEVICE
```

The above python script creates a patched kernel and a shell script to create Ramdisk.

```
sh ./make_ramdisk_n88ap.sh
```

Running the shell script downloads the forensic tool kit and adds it to the Ramdisk. The Ramdisk image is just a plain HFS+ file system which is native to Macs, making it fairly simple to add files to it. All the steps mentioned above create a patched kernel and a custom Ramdisk with forensic tools. *Note:* I have created the patched kernel and a custom Ramdisk for iPhone 4. You can directly download these files and skip all the above steps.

Download Link - http://www.4shared.com/folder/dKmG68Im/iPhone_Forensics.html

Loading Forensic Toolkit

In order to load forensic toolkit, supply iOS 5 ipsw file, patched kernel and custom Ramdisk to redsn0w tool. Connect the device to computer using USB cable and run the below command. Follow the steps displayed by redsn0w to boot the device in DFU mode. In DFU mode, redsn0w exploits the BootRom vulnerability and loads patched kernel & custom Ramdisk on to the device.

```
./redsn0w_mac_0.9.9b5/redsn0w.app/Contents/MacOS/redsn0w -i
iOS5_IPSW_FOR_YOUR_DEVICE -r myramdisk.dmg -k kernelcache.release.n88.patched
```

If the process fails with the *No identifying data fetched* error, make sure that the host computer is connected to the internet. After redsn0w is done, the Ramdisk boots in verbose mode.

2. Establishing device to computer communication

Once booted with custom Ramdisk, networking capabilities (like WI-FI) are not enabled by default. So a different way is chosen to communicate with the device by following the approach that Apple took with iTunes. USBMUX is the protocol used by iTunes to talk to the booted iPhone and coordinate access to its iPhone services by other applications. USB multiplexing provides TCP like connectivity over a USB port using SSL. Over this channel iTunes uses AFC service to transfer files. But here we use this channel to establish a SSH connection and get a shell on the device.

SSH works on port 22. Tcprelay.py script redirects port 22 traffic to 2222 port.

```
python usbmuxd-python-client/tcprelay.py -t 22:2222 1999:1999
```

SSH is now accessible at localhost:2222.

```
ssh -p 2222 root@localhost
password: alpine
```

At this point, we get access to the file system. To make things even more complicated, every file is encrypted with its own unique encryption key tied to particular iOS device. Furthermore, data protection mechanism introduced with iOS 4 adds another layer of encryption that does not give access to the protected files & keychain items when the device is locked. Data protection is the combination of using hardware based encryption along with a software key. Every iPhone (>3gs) contains a special piece of hardware (AES processor) which handles the encryption with a set of hardcoded keys (UID, GID). OS running on the device cannot read the hardcoded keys but it can use the keys generate by UID (0x835 and 0x89B) for encryption and decryption. Software key is protected by a passcode and is also used to unlock the device every time the user wants to make use of the device. So in order to access the protected files, first we have to bypass the passcode.

3. Bypassing the iPhone passcode restrictions

Initially (< iOS 4), passcode is stored in a file which can be removed directly over SSH. Since the introduction of data protection (from iOS 4), passcode is used to encrypt protected files and keychain items on the device. So in order to decrypt the data, we have to supply the valid passcode.

Passcode validation is performed at two levels one at springboard and another one at kernel level. Bruteforce attack performed at springboard level locks the device, introduces delays and may lead to data wipe-out. However these protection mechanisms are not applicable at kernel level (AppleKeyStore method) and it leads to bruteforce attacks. To make brute force attacks less practical, passcode key derived from the user passcode is tied to hardware UID key. So the brute force can only be performed on the device and it is not possible to prepare pre compute values (like rainbow tables) offline.

```
python python_scripts/demo_bruteforce.py
```

Port 1999 opened with tcprelay.py is used by the bruteforce script. It connects to the custom restored_external daemon on the Ramdisk, collects basic device information (serial number, UDID, etc.), unique device keys (keys 0x835 and 0x89B), downloads the system keybag and tries to bruteforce the passcode (4 digits only).

Below table illustrates the time required to bruteforce different passcodes.

Passcode Complexity	Bruteforce time
4 digits	18 minutes
4 alphanumeric	51 hours
5 alphanumeric	8 years
8 alphanumeric	13,000 years

4. Reading the encrypted file system

Upon successful passcode brute force, the script automatically downloads the keychain. Keychain is a Sqllite database which stores sensitive data on your device. Keychain is encrypted with hardware key. Keychain also restrict which applications can access the stored data. Each application on your device has a unique *application-identifier* (also called as entitlements). The keychain service restricts which data an application can access based on this identifier. By default, applications can only access data associated with their own application-identifier. Later apple introduced keychain groups. Now applications which belong to same group can share the keychain items. There are two ways to access all the keychain items. One way is, by writing an application and making it as a member of all application groups. The other way is by writing an application and granting com.apple.keystore.access-keychain-keys entitlement.

Keychain database contents can be extracted using keychain_tool.py

```
python python_scripts/keychain_tool.py -d [UDID]/keychain-2.db [UDID]/[DATAVOLUMEID].plist
```

Execute the dump_data_partition shell script to dump the file system

```
./dump_data_partition.sh
```

The script reads the file system from the device and copies it to UDID directory as an image (.dmg) file. The image file can be opened using the modified HFSExplorer that will decrypt the files *on the fly*. To decrypt it permanently, emf_decrypter.py script can be used.

```
python python_scripts/emf_decrypter.py [UDID]/[data_DATE].dmg
```

It decrypts all files in the file system image. To view the decrypted files, mount the file system with below command.

```
Hdituil mount [UDID]/[data_DATE].dmg
```

As soon as the file system is decrypted, there are various files of interest available such as the mail database, the SMS database and location history, etc...

5. Recovering the deleted files

Deleting a file on iPhone, only deletes the file reference. So it is possible to recover the deleted files. To recover the deleted files run emf_undelete.py script.

```
python python_scripts/emf_undelete.py [UDID]/[data_DATE].dmg
```

With this technique it is possible to recover valuable data like call logs, deleted images, deleted SMS, deleted contacts, deleted voicemail and deleted emails.

Video
http://www.youtube.com/watch?v=2Fs6ee1yeq4&context=C32aee7aADOEgsToPDskLQueZ3j9YDdIXdGSYdCN26

References

1. iPhone data protection in depth by Jean-Baptiste Bédrune, Jean Sigwald
   http://esec-lab.sogeti.com/dotclear/public/publications/11-hitbamsterdam-iphonedataprotection.pdf
2. iPhone data protection tools
   http://code.google.com/p/iphone-dataprotection/
3. 'Handling iOS encryption in forensic investigation' by Jochem van Kerkwijk
   http://staff.science.uva.nl/~delaat/rp/2010-2011/p26/report.pdf
4. iPhone Forensics by Jonathan Zdziarski
   http://shop.oreilly.com/product/9780596153595.do
5. iPhone forensics white paper
   http://viaforensics.com/education/white-papers/iphone-forensics/
6. Keychain dumper
   http://labs.neohapsis.com/2011/02/28/researchers-steal-iphone-passwords-in-6-minutes-true-but-not-the-whole-story/
7. 25C3: Hacking the iPhone
   http://www.youtube.com/watch?v=1F7fHgj-e_o
8. iPhone wiki
   http://theiphonewiki.com

About Me:

Satish Bommisetty is an Information Security Professional with 5 years of experience in Penetration testing of web applications and mobile applications.

His blog is located at http://securitylearn.wordpress.com
Email: satishb3@hotmail.com