iPhone/iPod Touch Forensics Manual

Jonathan A. Zdziarski
32 West Dr., Bedford NH 03110
jonathan@zdziarski.com

Copyright © 2008 by Jonathan Zdziarski, All Rights Reserved
Document Rev. 13; June 2, 2008
Device Firmware 1.0.2 – 1.1.4

ACKNOWLEDGEMENTS

Many thanks to Forensic Agent David C. Graham for his validation work and Windows platform testing/troubleshooting, to Youssef Francis and Pepjin Oomen for accommodating my change requests to adapt iLiberty+ for forensic purposes, to Arnaldo Viegas de Lima for Windows platform troubleshooting and support, and to the iPhone Dev Team for ongoing research in legal, ethical techniques for accessing the iPhone/iPod touch platforms.

UPDATES

Periodic updates of this document are provided free of charge to public law enforcement personnel. To subscribe to receive future updates, send an email to the author from a verifiable public law enforcement account.

Table of Contents

Introduction

With the iPhone quickly becoming the market leader in mobile devices, the need to effectively perform forensic analysis of these devices has surfaced. Unlike most other smart phones, the iPhone incorporates desktop-like features in an easy-to-use mobile package. As a result of its wide spectrum of available features, many are likely to use it as a primary device for various forms of data and communication. While limited portions of data can be viewed using the direct GUI interfaces in the iPhone's software, much more hidden and deleted data is available by examining the raw disk image, which may provide for more thorough evidence gathering. Some of the data available includes:

- Keyboard caches containing usernames, passwords, search terms, and historical fragments of typed communication. Even after deleted, many keyboard caches can be easily recovered, even after several weeks.
- "Last state" screenshots automatically taken as an application is quit, suspended or terminated (used for aesthetic effects)
- Deleted images from the user's photo library and browsing cache
- Deleted address book entries, and other personal data
- Exhaustive call history, beyond that displayed
- Map tile images from Google® Maps application, and longitude/latitude coordinates of previous map searches (including location lookups)
- Browser cache and deleted browser objects
- Cached and deleted email messages, SMS messages, and corresponding time stamps and source/destination.
- Cached and deleted voicemail recordings
- Pairing records establishing trusted relationships between the device and one or more desktop computers

… most data survives even a full restore from iTunes!

Because the device is designed to provide for more than adequate storage needs, and because much of the content installed on the device remains static (such as music), the integrity of this data can be preserved for long periods of time. As the device uses a solid-state flash memory, it is designed to minimize writes, and sometimes even appears to spread out writes across the flash, thus leaving data intact for long periods of time.

This manual is designed as an aide for lawful, warranted forensic analysis to recover this and other data from what is an otherwise closed device, using publicly available third-party tools and customized proprietary tools packaged into a toolkit. It is by no means a complete forensic manual, but intended to cover the details that are specific to the iPhone. The technical notes in this manual should be combined with best practices in forensic investigation including handling of digital evidence, cross-contamination, and process disclosure.

What You'll Need

- A desktop/notebook machine running either Mac OS X Leopard or Windows XP. The tools used are also compatible with Tiger and Vista, but are not as widely tested. File paths for desktop trace have been provided for Mac OS X, Windows XP and Windows Vista. Examples in this document are also provided for both Mac and Windows operating systems. Due to the nature of the iPhone and its native HFS+ file system, however, it is by far easiest to analyze such a device using a Leopard-based Mac.

- An iPhone USB dock connector or cable. This will be required to install the forensics recovery toolkit into a nondestructive location on the device and to keep the device charged during the recovery process.

- Working WiFi on your desktop machine and an access point which both the iPhone and the desktop can connect to (preferably securely). In the absence of an isolated access point, links to instructions for creating ad-hoc networks are included. In most cases, disk copy can be performed over an SSH tunnel to further secure the data while in transit.

- An implementation of SSH (Secure Shell) on your desktop, including `ssh` and `scp` tools.

- iTunes from Apple. Version 7.6 was used for this manual, but other versions are likely to work as well. Source code examples will require a copy of the iTunes version 7.4.2 mobile device framework.

- Adequate disk space on the desktop machine to contain copies of the iPhone's media partition and extracted content. The minimum recommended space is three times the device's advertised capacity.

- General knowledge of UNIX and computer forensic methodology. Procedure is not covered here, only technical details, and so regardless of the quality of data recovered, it will be inadmissible if not properly preserved and documented.

Contacting Me

Sworn law enforcement officers and forensic investigators are welcome to contact me with any questions or suggested improvements at *jonathan@zdziarski.com*. I am also equipped to handle examination requests if needed.

About the iPhone

The iPhone is a mobile device designed and marketed by Apple Inc. Different models may vary, however the following components are most commonly used:

- Application Processor (CPU): Samsung/ARM S5L8900B01 512Mbit SRAM
- EDGE Baseband Processor: Infineon PMB8876 S-Gold 2
- GSM RF Transceiver: Infineon M1817A11
- MLC NAND Flash Memory: Samsung 65-nm 8/16GB (K9MCG08U5M), 4GB (K9HBG08U1M)
- GSM/EDGE Power Amplifier: Skyworks SKY77340-13
- WLAN Device chip: Marvell 90-nm 88W8686
- I/O Controller Chip: Broadcom BCM5973A
- Wireless NOR Flash Memory: Intel PF38F1030W0YTQ2 (32Mbytes NOR + 16Mbytes SRAM)
- Audio Codec Processor: Wolfson WM8758
- Bluetooth Device Chip: CSR BlueCore 4 ROM
- Touchscreen Processor: Philips LPC2221/02992

The iPhone runs a mobile build of Mac OS X 10.5 (Leopard), which has many similarities to its desktop counterpart. The primary differences are the architecture, user interface frameworks, and its use of a secure (although now exploited) kernel, designed to prevent tampering. The kernel itself is mapped into the file system, but believed to actually reside in a different location in the NOR.

In an effort to unlock the device and develop third-party software, the iPhone has become the subject of many hacker groups and developers. Many techniques have been found to access its operating system and lower-level components as a result of this. This has led to a significant software development community and the development of many tools, some of which will be used in this manual. Also used will be a custom forensics recovery toolkit for the iPhone consisting of OpenSSH, a basic UNIX world, and disk and network copy tools built for the iPhone's ARM architecture using an open source cross-compiler.

Determining the Firmware Version

To determine the version of operating firmware installed on the device, tap on the settings icon, then select *General* > *About*. The version number will be displayed with a build number in parenthesis. Before proceeding, ensure that the firmware version of the device falls within the range of versions supported by this document.

Disk Layout

By default, the iPhone is configured with two disk partitions. A system (root) partition approximately 300MB in size is used to house the operating system and preloaded applications, while the remaining available space is assigned to a user "media" partition mounted at `/private/var`. This scheme was used to allow iTunes to perform easy upgrades of the operating firmware without erasing user data.

The system partition is mounted as read-only by default, meaning it will remain in a factory state unless intentionally modified. As a result of this design, all user information (such as keyboard cache, contacts, browser data, and other user information) is stored on the separate media partition. The device nodes are as follows:

```
Block Devices:
  brw-r-----   1 root  operator   14,   0 Apr  7 07:46 /dev/disk0    Entire Disk
  brw-r-----   1 root  operator   14,   1 Apr  7 07:46 /dev/disk0s1  System (/)
  brw-r-----   1 root  operator   14,   2 Apr  7 07:46 /dev/disk0s2  Media (/private/var)

Raw Devices:
  crw-r-----   1 root  operator   14,   0 Apr  7 07:46 /dev/rdisk0   Entire Disk
  crw-r-----   1 root  operator   14,   1 Apr  7 07:46 /dev/rdisk0s1 System (/)
  crw-r-----   1 root  operator   14,   2 Apr  7 07:46 /dev/rdisk0s2 Media (/private/var)
```

The techniques used in this manual will use tools to mount the system (root) partition as read-write and install a recovery toolkit payload to gain access to the iPhone's operating system. Because the system partition is not designed to store user data, this operation is considered to be safe for conducting forensic analysis, as it leaves the media partition (including free space and deleted files) intact.

Communication

The iPhone can communicate across several different mediums, including serial (via AFC protocol), 802.11 WiFi, and Bluetooth. Due to Bluetooth limitations at the operating system level, the two preferred methods are via AFC and WiFi.

The AFC protocol (Apple File Connection) is the protocol used by iTunes to copy files to/from the device and to send firmware-level commands. This takes place over the device's USB dock connector, and uses a private *MobileDevice* framework installed with iTunes. Third party tools have been written to use this framework to perform ad-hoc operations using this protocol. By default, the environment that AFC is permitted to access on the iPhone is restricted to its `/var/mobile/Media` folder (`/var/root/Media` for software versions <= 1.1.2). This prevents iTunes, as well as third-party tools, from accessing lower-level areas of the operating system without modification. This is commonly referred to as a *chroot jail*. The term *jailbreaking* originated from the very first iPhone modifications to break out of this restricted jail, allowing AFC to be used to read and write files anywhere on the device. The AFC protocol will be used by some of the tools outlined in this manual to place the device into recovery mode and install the recovery toolkit on the system partition.

Because the AFC protocol does *not* support reading from raw devices, it will not be used for the actual disk image recovery process. Instead, once access has been made to the iPhone, raw disk imaging will be conducted over WiFi. This allows network tools, such as OpenSSH, to be installed on the device, allowing the examiner to gain shell access directly and perform various functions.

The AFC protocol can, in fact, be tricked into transferring raw device data across the USB, however the procedure is very ad-hoc and therefore questionable. As a result, it would be difficult for a jury to understand. Transferring files over a secure wireless connection is a much more widely understood technique, and provides for a more credible level of integrity in raw device recovery as it uses standard protocols and proper integrity checks.

To ensure evidentiary integrity, it is recommended that the disk imaging process be conducted across a WPA-keyed encrypted access point, even when using the file copy example occurring over an encrypted tunnel. Transfer across a WEP-keyed access point could potentially raise questions about WEP's initialization vector vulnerability, allowing a malicious third party to deduce the network key, thereby making tampering a possibility, albeit unlikely.

Power-On Device Modifications (Disclosure)

By default, the iPhone's file system is mounted with the `noatime` option, even if it is not specified in */etc/fstab*. This option prevents access times from being updated when a file is accessed on the device, meaning that files will only be *touched* when they are opened for writing.

Until a method is devised that would allow the iPhone's memory chip to be physically dumped or mounted on another device, the iPhone must be powered on and booted into its operating system to recover data. Furthermore, the forensic toolkit requires that the device be rebooted after the toolkit payload is installed. Just like a desktop operating system, the iPhone's Leopard operating system performs minor writes to certain files upon booting. Most writes are performed to replace or reset existing configuration files, which do not generally add any utilized space to the file system. Some writes, however, append a very minor amount of data to files. Overall, the writes to the file system are minimal, but documented here as they are changes involving the media partition.

Filename	**Est. Magnitude of Change**
/private/var/log/lastlog	28 bytes overwritten
/private/var/mobile/Library/Preferences/com.apple.voicemail.plist	1275 bytes overwritten
/private/var/preferences/csidata	121 bytes overwritten
/private/var/run/configd.pid	3 bytes overwritten
/private/var/run/resolv.conf	76 bytes overwritten
/private/var/root/Library/Lockdown/data_ark.plist	3252 bytes overwritten
/private/var/tmp/MediaCache/diskcacherepository.plist	320 bytes overwritten
/private/var/log/wtmp	144 bytes appended
/private/var/mobile/Library/Voicemail/_subscribed	Modification time updated
/private/var/mobile/Library/Voicemail/voicemail.db	7168 bytes overwritten
/private/var/preferences/SystemConfiguration/...	
NetworkInterface.plist	783 bytes overwritten
com.apple.AutoWake.plist	730 bytes overwritten
com.apple.network.identification.plist	1305 bytes overwritten
com.apple.wifi.plist	2284 bytes overwritten
preferences.plist	4380 bytes overwritten

On iPhone firmware versions <= 1.1.2, the *mobile* directory is replaced with *root*

In addition to the files above, the following files may be written to or recreated by logging into the device, causing the following changes to take place:

Filename	Est. Magnitude of Change
/private/var/run/utmp	468 new bytes written/replaced
/private/var/run/utmpx	1256 new bytes written/repalced

Upgrading the iPhone Firmware

Apple provides periodic firmware updates, which update the operating system, radio baseband, and possibly other device firmware of the iPhone. Thus far, these updates have not resulted in the loss of any live user data, but do frequently rename files and may occasionally write new ones to the media partition. It is therefore advisable not to update the iPhone's firmware for forensic purposes, except as a last resort. This will only be required if the device is running an older version of the firmware than is supported by this manual (1.0.0 or 1.0.1), and if no other suitable techniques are available to access these older firmware versions in a nondestructive manner.

To upgrade the iPhone firmware to the latest version, use the `update` button available in iTunes. If the most recent version of device firmware is not supported in this manual, the closest supported version may be downloaded manually and installed by using `Option+Click` (Mac) or `Shift+Click` (Windows) when using the `update` button. This will allow the examiner to select the desired firmware file to upgrade to.

This manual covers a wide range of iPhone software versions. Do not upgrade the iPhone's firmware unless absolutely necessary. If an upgrade is required, use the closest supported version to the currently installed version.

The following supported iPhone firmware updates can be downloaded from Apple's cache servers:

1.0.2 http://appldnld.apple.com.edgesuite.net/content.info.apple.com/iPhone/061-3823.20070821.vormd/iPhone1,1_1.0.2_1C28_Restore.ipsw

1.1.1 http://appldnld.apple.com.edgesuite.net/content.info.apple.com/iPhone/061-3883.20070927.In76t/iPhone1,1_1.1.1_3A109a_Restore.ipsw

1.1.2 http://appldnld.apple.com.edgesuite.net/content.info.apple.com/iPhone/061-4037.20071107.5Bghn/iPhone1,1_1.1.2_3B48b_Restore.ipsw

1.1.3 http://appldnld.apple.com.edgesuite.net/content.info.apple.com/iPhone/061-4061.20080115.4Fvn7/iPhone1,1_1.1.3_4A93_Restore.ipsw

1.1.4 http://appldnld.apple.com.edgesuite.net/content.info.apple.com/iPhone/061-4313.20080226.Sw39i/iPhone1,1_1.1.4_4A102_Restore.ipsw

See Apple's iTunes documentation for more information about updating the iPhone firmware.

Restore Mode and Integrity of Evidence

User data is typically preserved when the "update" option is used via iTunes, however a suspect may attempt to destroy the file system by using the iTunes *restore* function. When restored, the file system is destroyed, however the NAND itself is not initialized and so much of the data should still be recoverable following the recovery steps outlined in this manual. In addition to this, the device's previous configuration may have been intentionally restored after the firmware has been re-flahed, providing a full "live" backup of basic data. A backup of the device's files may also have been written to the desktop machine that performed a sync or restore. See the section titled *Desktop Trace* for more information.

Simply placing a device into restore mode does not destroy the file system. A fleeing suspect may attempt such a feat if they are aware of incriminating evidence on their device by holding down the `Home` and `Power` buttons on the iPhone until a *Connect to iTunes* message and/or icon is displayed. The forensic examiner might also accidentally enter the device into restore mode if a mistake is made during the recovery process, or if an unforeseen interruption occurs. DO NOT PANIC. When the device is placed into restore mode, all data still remains intact. In restore mode, the device can in fact be made to boot back into the operating system without a loss of data, provided the user has initiated the recovery process via iTunes to re-image the device. If the iPhone is simply sitting in restore mode and has not been re-imaged from iTunes, it will need to be booted back into the operating system to perform forensic recovery. This can be done from the iLiberty+ tool discussed in the next section. Some versions of iPhone firmware have been reported to kick themselves out of recovery mode within ten minutes of sitting idle, while connected to the dock.

When a full restore of an iPhone is performed (using iTunes' *Restore* option), the file system is destroyed, but the NAND is not re-initialized. This means that much of the data previously stored on the device should still be recoverable.

Cross-Contamination of Evidence and Syncing

Before performing any of the steps in this manual, be sure to disable all automatic syncing with iPhone/iPod Touch devices, and **never** attempt to sync a suspect's device manually. Otherwise, the examiner runs the risk of cross-contaminating a device with music, photos, and other data already synced to the desktop machine. It is also advisable to conduct all forensic recovery using a desktop machine with a fresh, write-protected installation of the operating system, iTunes, and the tools discussed in this manual. Ideally, the examiner should consider building a bootable CD or virtual machine to perform the recovery steps outlined throughout the rest of this manual, or using an external storage device freshly zeroed and formatted for each investigation.

Accessing the Device

Installing the Forensic Toolkit

The first step in performing forensic examination of the iPhone is to gain access to the operating system. This is required in order to obtain a raw disk image of the media partition for analysis. To gain access, a forensic-friendly *jailbreaking* tool will be used, which does not itself write to the user partition of the device.

The iLiberty+ program is a free tool designed by Youssef Francis and Pepijn Oomen for unlocking and installation of various payloads onto the iPhone/iPod Touch. iLiberty+ implements the low-level technique described in the *Technical Procedure* section of this manual to boot an unsigned RAM disk. Once loaded, this RAM disk then mounts the iPhone's root file system and will be used to install a recovery toolkit which includes OpenSSH, a basic UNIX world, netcat, and the *dd* disk copy/imaging tool. These will be used to recover and transmit a disk image of the device's media partition across a network connection.

For a low-level explanation of the *technical procedures* used by this tool, see the Technical Procedure section at the end of this manual.

Step 1: Download and Install iLiberty+

Download iLiberty+ from *http://theiphoneproject.org/downloads/*. Archives of the versions used in this manual may also be downloaded from *http://www.zdziarski.com/forensic-toolkit/iLiberty+/*.

Release candidates of iLiberty v1.6 may be downloaded from the forensic toolkit link above. Version 1.6 corrects some problems with inadvertent (yet minor) writing to the user data partition in version 1.51 and compatibility with certain older versions of iPhone software. It also supports an integrated passcode bypass tool, which works as effectively in most cases as the manual process outlined at the end of this chapter.

⇒ **Mac OS X** (iLiberty+ v1.51)

Drag iLiberty+.app into the /Applications folder on your desktop.

iLiberty+ v1.51 ONLY

iLiberty+ v1.51 installs the NullRiver software installer as a default payload. This performs two minor writes to the user media partition, but these writes can be avoided altogether. Download and install the custom jailbreak payload available at *http://www.zdziarski.com/forensic-toolkit/iLiberty+/iLiberty+.bin*. Use this to overwrite the existing default payload installed inside the application located at */Applications/iLiberty+.app/Contents/Resources/iLiberty+.bin* on your Mac OS X desktop. This new payload skips the NullRiver software installer, which is not needed.

⇒ **Windows** (iLiberty+ v1.3.0.113)

Run the iLiberty+ installer application. The application will be installed in *C:\Program Files\iLiberty\*, and icons will be added to the desktop and/or start menu.

Step 2: Dock the iPhone and Launch iTunes

Connect the iPhone to a dock connector and connect it to your desktop's USB port. This will keep the device charged as well as provide the connection needed to install the toolkit. Launch iTunes, and ensure the device is recognized. You should see the iPhone appear in the sidebar under the *Devices* section.

If the device was seized while in restore mode, use the *Exit Recovery* option from iLiberty+'s advanced menu (Mac OS X) or the *Jump Out of Recovery Mode* option from the Other Tools tab (Windows) to boot the device back into the operating system. Other tools, such as iBrickr, iNdependence, and iPHUC (iPHone Utility Client) can also be used to boot the iPhone if the iLiberty+ method fails. This will avoid reformatting the iPhone by restoring in iTunes. In some cases, the iPhone will kick itself out of recovery mode after approximately ten minutes, provided it remains powered on and connected to iTunes through the USB dock cable.

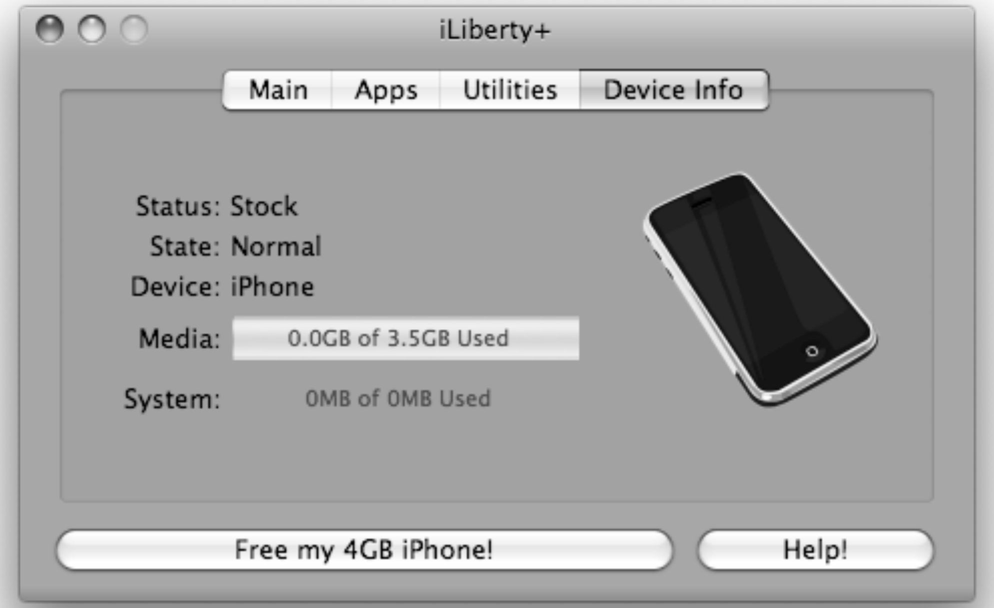
Step 3: Launch iLiberty+ and Ensure Connectivity

Launch iLiberty+. The iPhone should be detected upon launch.

During the jailbreaking process, iTunes may notify you that it has detected a device in recovery mode, and ask you if you would like to restore. This is normal, as iTunes is oblivious to the fact that the device is being worked on by another application. **Never instruct iTunes to perform a restore, or you will damage evidence!** If necessary, you may cancel this request or simply ignore it.

⇒ **Mac OS X**

Click on the *Device Info* tab to view information about the device's system and media partitions.



Status: Stock
State: Normal
Device: iPhone
Media: 0.0GB of 3.5GB Used
System: 0MB of 0MB Used

Fig. 1 iLiberty Status (Mac OS X)

⇒ **Windows**

Verify that iLiberty+ is reporting the status of the iPhone at the bottom right of the status bar. The status should read *Normal Mode*.

Fig. 2 iLiberty Status (Windows)

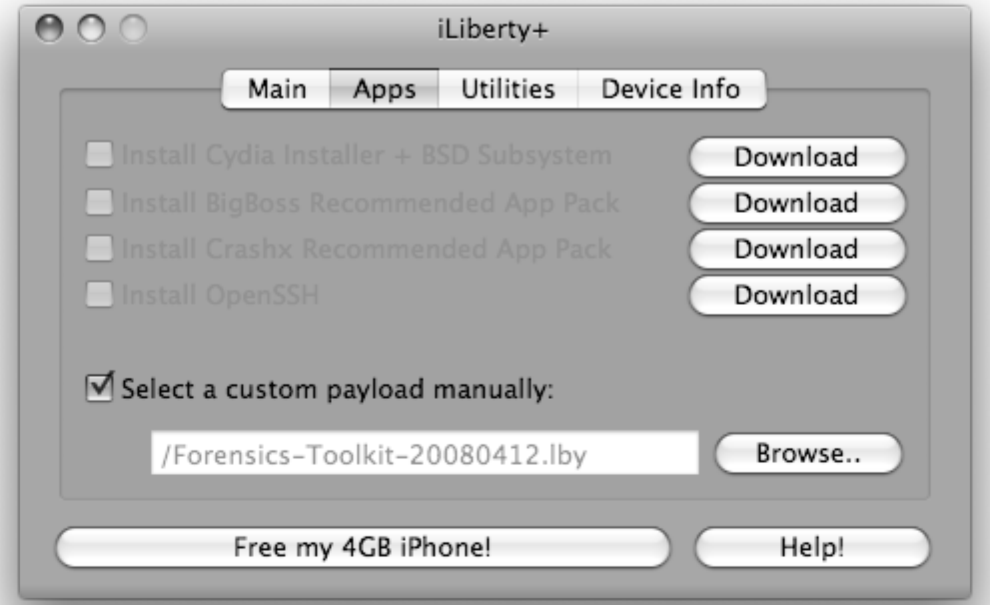
Step 4: Configure for Forensic-Toolkit Payload

Once the device has been recognized, the forensics toolkit payload should be downloaded and activated for the target device. Releases of the toolkit are stored in a repository located at *http://www.zdziarski.com/forensic-toolkit/Payloads/*. Each version of the toolkit is distributed in both an *.lby* format (for Mac OS X) and *.zip* file format (for Windows). The two archives are identical, and simply use a different file extension. Download the appropriate file extension for your operating system.

⇒ **Mac OS X**

Click on the `Apps` tab in iLiberty+. Make sure all payload checkboxes are unchecked. Check the checkbox labeled *Select a custom payload manually*. Download the latest (or desired) version of the Forensic-Toolkit payload *.lby* file from *http://www.zdziarski.com/forensic-toolkit/Payloads/* and then click `Browse`. Locate the file and click `Open`. It should now be selected and displayed in the custom payload field.

Some web browsers will automatically rename the *.lby* file to have a *.zip* file extension. If you are unable to select your payload in iLiberty+, check and ensure that the file extension is correct, and rename it back to *.lby* if necessary.



Select a custom payload manually: [✓]

/Forensics-Toolkit-20080412.lby

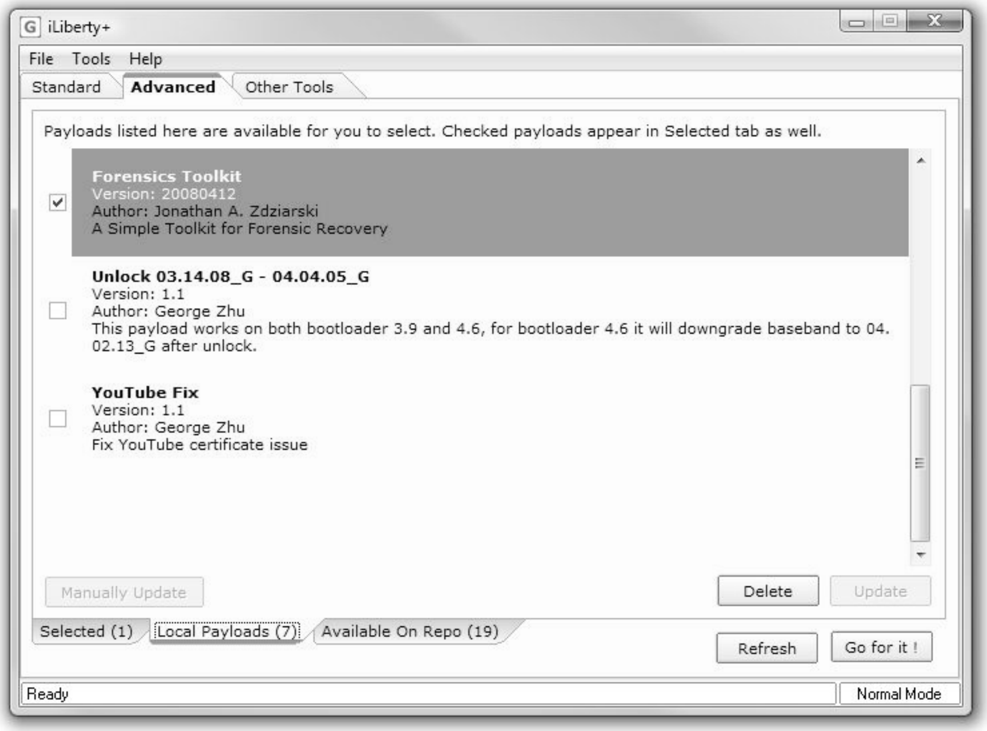
Fig. 3 Forensic-Toolkit Payload Selection (Mac OS X)

⇒ **Windows**

Download the latest (or desired) version of the Forensic-Toolkit payload *.zip* file from *http://www.zdziarski.com/forensic-toolkit/Payloads/*. Extract the contents of the archive. This should output two files: *90Forensics.sh* and *forensics-toolkit-(VERSION).zip*. Copy or move these two files into *C:\Program Files\iLiberty\payload\*.

Now click the `Advanced` tab. Click the bottom tab titled *Local Payloads*. Scroll to the *Forensics Toolkit* payload and click its checkbox. The payload should then appear under the `Selected` tab, which means it is now activated for installation.

If the toolkit does not appear on the list of local payloads, try clicking the `Refresh` button or restart iLiberty+.



Payloads listed here are available for you to select. Checked payloads appear in Selected tab as well.

✓ **Forensics Toolkit**
Version: 20080412
Author: Jonathan A. Zdziarski
A Simple Toolkit for Forensic Recovery

Unlock 03.14.08_G - 04.04.05_G
Version: 1.1
Author: George Zhu
This payload works on both bootloader 3.9 and 4.6, for bootloader 4.6 it will downgrade baseband to 04.02.13_G after unlock.

YouTube Fix
Version: 1.1
Author: George Zhu
Fix YouTube certificate issue

Fig. 4 Forensic-Toolkit Payload Selection (Windows)

Step 5: Execute the Payload

After verifying that the Forensics Toolkit payload has been activated, click the bottom button labeled `Free my` *`(CAPACITY)`* `iPhone` (Mac OS X) or `Go for it!` (Windows).

⇒ **Mac OS X**

A window will appear informing you of the tool's progress. The device should boot into the text-based screen described below to install the toolkit payload.

⇒ **Windows**

Before the jailbreak process commences, you will be asked to unplug the iPhone from its USB connection and then reconnect it. Unplug the device, and wait until it disappears from iTunes. Reconnect the device, and wait until it appears again in iTunes. Only after this, click the `OK` button. A progress window will appear, but may vanish as the device enters recovery mode. The process is still running in the background, however, and you should see status text such as *Booting Ramdisk* in the status bar of the iLiberty+ application. The device itself should, after a short time, boot into the text-based screen described below to install the toolkit payload.

In some rare cases, the device will either get stuck in recovery mode or fail to enter recovery mode at all:

⇒ If the device becomes stuck in recovery mode, follow the steps in step 2 to boot the device back into the operating system. This will safely boot the device without any loss of data.

⇒ If the device fails to enter recovery mode (appearing to do nothing), manually force it into recovery by holding down the `Power` and `Home` buttons until the device hard-powers itself off, back on, and finally displays the recovery screen (do not let up on the buttons until you see the *Connect to iTunes* text and/or icon). In iLiberty+, click the `Manual Boot` option on the *Other Tools* tab to boot the device manually. The device will boot out of recovery and install the forensic toolkit payload. Should this fail, repeat steps 2-5 once more.

During the jailbreaking process, the device will go through what will appear to be various text-based diagnostic and configuration screens. Note any errors, should they occur. Once the process has completed, the device should briefly display *Forensics Toolkit Installation Successful*, and will then reboot back into its operational state.

The device should now be ready to accept an SSH connection. See the next section to configure WiFi and access the device.

Configuring WiFi and SSH

Now that the forensic toolkit has been installed on the device, the toolkit's OpenSSH daemon should now be listening for incoming connections, and a UNIX world has been installed on the system partition to enable basic file and disk operations. In order to access the device, WiFi must first be configured to connect to the same network as the desktop machine.

Your access point must not enable an "AP Isolation" feature, which prevents devices on the network from communicating with other local devices. If your access point is configured in this fashion, you must either disable this feature, or revert to using an ad-hoc network. Otherwise, the desktop machine and the device will not be able to communicate.

Ad-Hoc Networks

If no access point is available, or if insecure devices are not permitted by policy to connect to local access points, the desktop can be configured to serve as its own access point. Both machines will require a static IP address.

⇒ For instructions on configuring your Windows XP machine to run an ad-hoc network, visit *http://www.microsoft.com/windowsxp/using/networking/setup/adhoc.mspx*

⇒ For instructions on configuring Mac OS X Leopard to run an ad-hoc network, visit *http://zdziarski.com/papers/tethering.txt*

The iPhone has difficulty connecting to ad-hoc networks that are using encryption. If using an ad-hoc network, it may need to be created as open. It has been reported that repeated attempts to connect to an encrypted ad-hoc network sometimes succeed. To ensure the evidence has not been tampered with while in transit, however, it may be appropriate to use a separate, local physical access point that has been secured with WPA. Ask your supervisor which technique is most appropriate if you are uncertain. One may also need to consult the district attorney to ensure the desired technique will be admissible. When using an unencrypted network, it is strongly advisable to perform the file copy over an encrypted tunnel - an example will be provided. If you are still uncertain, consider purchasing a WPA-capable wireless access point dedicated for forensic recovery.

Configuring Wireless (Device)

1. To configure wireless access on the iPhone, tap on the settings icon. A list of options will appear.

2. Tap the Wi-Fi option, second down from the top. This will transition to a window where WiFi can be configured. If WiFi is turned off, tap the switch at the top to turn it on.

3. A list of available WiFi networks should appear in the section labeled, *Choose a Network*. Tap on the network your desktop is presently connected to. A wait indicator will be displayed while the iPhone joins the network.

4. Once joined, tap the blue disclosure arrow to the right of the selected WiFi network. This will allow you to view and/or change the iPhone's IP address and other settings on the network.

5. Take note of the IP address of the iPhone, as you'll need it later. Use the `ping` utility on the desktop to ensure that the device is reachable. If it is not, one or both of the devices may be misconfigured, or the access point may enforce AP isolation.

SSH into the iPhone

Once the iPhone is active on the network, you may now connect to it via ssh from your desktop:

```
$ ssh –l root x.x.x.x
```

Where ***x.x.x.x*** is replaced by the IP address of the iPhone. If you are unable to connect, try *ping*ing the device to ensure you have network connectivity.

The forensics toolkit automatically resets the password for the `root` user to `alpine`.

Once you have successfully logged into the device, you're now ready to image the media partition. Proceed to the next section.

Installation Record (Disclosure)

The payload installed by iLiberty+ installs a forensic toolkit onto the otherwise read-only portion of the device, resulting in no destruction to user-level data stored on the device's media partition. At the time of payload installation, the following files are written to the system (root) partition.

File size may vary depending on the application and payload versions used. Some files are deleted after toolkit installation.

```
/usr/libexec/ipluspwns (basepack)
-rwxr-xr-x  1 root wheel   25212 Mar 27 08:59 chmod*
-rwxr-xr-x  1 root wheel   38320 Mar 27 08:59 echo*
-rwxr-xr-x  1 root wheel   23292 Mar 27 08:59 iPipe*
-rwxr-xr-x  1 root wheel   14352 Mar 27 08:59 mv*
-rwxr-xr-x  1 root wheel   13760 Mar 27 08:59 reboot*
-rwxr-xr-x  1 root wheel   19128 Mar 27 08:59 rm*
-rwxr-xr-x  1 root wheel 1298880 Mar 27 08:59 sh*
-rwxr-xr-x  1 root wheel   39036 Mar 27 08:59 sleep*
-rwxr-xr-x  1 root wheel   14916 Mar 27 08:59 umount*
-rwxr-xr-x  1 root wheel  141528 Mar 27 08:59 unzip*


/bin (basepack)
-rwxr-xr-x  1 root  wheel   134152 Mar 27 08:59 awk
-rwxr-xr-x  1 root  wheel    23368 Mar 27 08:59 blcheck
-rwxr-xr-x  1 root  wheel    14368 Mar 27 08:59 cat
```

```
-rwxr-xr-x  1 root  wheel    25212 Mar 27 08:59 chmod
-rwxr-xr-x  1 root  wheel    80660 Mar 27 08:59 chown
-rwxr-xr-x  1 root  wheel    19644 Mar 27 08:59 cp
-rwxr-xr-x  1 root  wheel    18972 Mar 27 08:59 cut
-rwxr-xr-x  1 root  wheel    33288 Mar 27 08:59 dd
-rwxr-xr-x  1 root  wheel     9212 Mar 27 08:59 dirname
-rw-r--r--  1 root  wheel     2971 Apr  1 20:25 functions.inc
-rwxr-xr-x  1 root  wheel   158708 Mar 27 08:59 grep
-rwxr-xr-x  1 root  wheel    18056 Mar 31 14:03 iEdit
-rwxr-xr-x  1 root  wheel    20776 Mar 27 08:59 igsm
-rwxr-xr-x  1 root  wheel    13492 Mar 31 14:03 ln
-rwxr-xr-x  1 root  wheel    41028 Mar 27 08:59 ls
-rwxr-xr-x  1 root  wheel    13348 Mar 31 14:03 mkdir
-rwxr-xr-x  1 root  wheel    24244 Mar 27 08:59 plutil
-rwxr-xr-x  1 root  wheel    13760 Mar 27 08:59 reboot
-rwxr-xr-x  1 root  wheel    19172 Mar 27 08:59 rm
-rwxr-xr-x  1 root  wheel    42888 Mar 27 08:59 sed
-rwxr-xr-x  1 root  wheel  1298880 Mar 27 08:59 sh
-rwxr-xr-x  1 root  wheel     9392 Mar 27 08:59 sleep
-rwxr-xr-x  1 root  wheel   260244 Mar 27 08:59 tar
-rwxr-xr-x  1 root  wheel   141528 Mar 27 08:59 unzip
```

/bin (payload)

```
-rwxr-xr-x  1 root  wheel  591364 Mar 16 09:23 bash
-rwxr-xr-x  1 root  wheel   45804 Feb 29 04:55 cat
-rwxr-xr-x  1 root  wheel   74456 Feb 29 04:55 chgrp
-rwxr-xr-x  1 root  wheel   65632 Feb 29 04:55 chmod
-rwxr-xr-x  1 root  wheel   74724 Feb 29 04:55 chown
-rwxr-xr-x  1 root  wheel  159704 Feb 29 04:55 cp
-rwxr-xr-x  1 root  wheel   33288 Apr  7 10:25 dd
-rwxr-xr-x  1 root  wheel  119948 Mar 27 07:48 grep
-rwxr-xr-x  1 root  wheel  115848 Feb 29 04:55 ln
-rwxr-xr-x  1 root  wheel  146360 Feb 29 04:55 ls
-rwxr-xr-x  1 root  wheel   44452 Feb 29 04:55 mkdir
-rwxr-xr-x  1 root  wheel   45900 Feb 29 04:55 mknod
-rwxr-xr-x  1 root  wheel  169368 Feb 29 04:55 mv
-rwxr-xr-x  1 root  wheel   39292 Feb 29 04:55 pwd
-rwxr-xr-x  1 root  wheel   13760 Apr  8 00:35 reboot
-rwxr-xr-x  1 root  wheel  142636 Feb 29 04:55 rm
lrwxr-xr-x  1 root  wheel       4 Apr  8 00:18 sh -> bash
-rwxr-xr-x  1 root  wheel   17004 Feb 27 18:50 sync
```

/etc (payload)

```
-rw-r--r--  1 root  wheel  1418 Jun 12  2006 ssh_config
-rw-r--r--  1 root  wheel  3230 Aug 25  2007 sshd_config
```

/sbin (payload)

```
-rwxr-xr-x  1 root  wheel  185008 Apr  8 00:34 fsck_hfs
-rwxr-xr-x  1 root  wheel   18052 May  7 12:12 md5
-rwxr-xr-x  1 root  wheel   19236 Apr  8 00:34 mount_hfs
-rwxr-xr-x  1 root  wheel   46300 Apr  8 00:35 newfs_hfs
-rwxr-xr-x  1 root  wheel  191976 May  7 12:22 ping
-rwxr-xr-x  1 root  wheel   14916 Apr  8 00:37 umount
```

/usr/bin (payload)

```
-rwsr-xr-x  1 root  wheel   31712 Feb 27 18:50 login
-rwxr-xr-x  1 root  wheel   29520 Apr  8 00:34 nc
-rwxr-xr-x  1 root  wheel   56284 Aug 23  2007 scp
-rwxr-xr-x  1 root  wheel   88876 Aug 23  2007 sftp
-rwxr-xr-x  1 root  wheel  340340 Aug 23  2007 ssh
-rwxr-xr-x  1 root  wheel  103960 Aug 23  2007 ssh-add
-rwxr-xr-x  1 root  wheel   87336 Aug 23  2007 ssh-agent
-rwxr-xr-x  1 root  wheel  134264 Aug 23  2007 ssh-keygen
-rwxr-xr-x  1 root  wheel  198048 Aug 23  2007 ssh-keyscan
```

/usr/lib (payload)

```
lrwxr-xr-x  1 root  wheel      18 Apr  8 00:18 libcurses.dylib ->
                                               libncurses.5.dylib
-r-xr-xr-x  1 root  wheel   35392 Jan  3 20:31 libhistory.5.2.dylib
lrwxr-xr-x  1 root  wheel      20 Apr  8 00:18 libhistory.5.dylib ->
```

```
                                                 libhistory.5.2.dylib
lrwxr-xr-x  1 root  wheel      20 Apr  8 00:18 libhistory.dylib ->
                                                 libhistory.5.2.dylib
-rw-r--r--  1 root  wheel   60780 Jan 14 21:44 libintl.8.0.2.dylib
lrwxr-xr-x  1 root  wheel      19 Apr  8 00:18 libintl.8.dylib ->
                                                 libintl.8.0.2.dylib
lrwxr-xr-x  1 root  wheel      19 Apr  8 00:18 libintl.dylib ->
                                                 libintl.8.0.2.dylib
-rw-r--r--  1 root  wheel     801 Jan 14 21:44 libintl.la
-rwxr-xr-x  1 root  wheel  105156 Feb 23 06:30 libncurses++.a
-rwxr-xr-x  1 root  wheel  379360 Feb 23 06:30 libncurses.5.dylib
lrwxr-xr-x  1 root  wheel      18 Apr  8 00:18 libncurses.dylib ->
                                                 libncurses.5.dylib
-r-xr-xr-x  1 root  wheel  239308 Jan  3 20:31 libreadline.5.2.dylib
lrwxr-xr-x  1 root  wheel      21 Apr  8 00:18 libreadline.5.dylib ->
                                                 libreadline.5.2.dylib
lrwxr-xr-x  1 root  wheel      21 Apr  8 00:18 libreadline.dylib ->
                                                 libreadline.5.2.dylib
-rwxr-xr-x  1 root  wheel  247684 Jan  4 05:35 libresolv.dylib
lrwxr-xr-x  1 root  wheel      17 Apr  8 00:18 terminfo -> ../share/terminfo


/usr/libexec (payload)
-rwxr-xr-x  1 root  wheel   59372 Aug 23  2007 sftp-server
-rwxr-xr-x  1 root  wheel  200664 Aug 23  2007 ssh-keysign
-rwxr-xr-x  1 root  wheel   35280 Aug 23  2007 ssh-rand-helper
-r-xr-xr-x  1 root  wheel     425 Dec 20  2006 sshd-keygen-wrapper

/usr/sbin (payload)
-rwxr-xr-x  1 root  wheel   32784 Apr  8 00:36 fdisk
-rwxr-xr-x  1 root  wheel  414512 Aug 23  2007 sshd

/Library/LaunchDaemons (payload)
-rw-r--r--  1 root  wheel  828 Feb  4  2006 com.openssh.sshd.plist
```

Circumventing Passcode Protection

There are two types of locks used on the iPhone: a SIM lock and an OS-level passcode. The SIM lock can be bypassed by simply removing / replacing the SIM. Devices protected with a passcode, however, must have it circumvented before the forensic toolkit can be installed.

To disable the passcode, raw commands will need to be issued to the iPhone to load a custom passcode-circumvention RAM disk. This requires the use of an open source tool called the iPhone Utility Client (humorously named iPHUC). Before circumventing the passcode, follow the steps below to set up a working version of the iPhone Utility Client on your desktop machine:

⇒ **Mac OS X**

Download the iPHUC_Universal.tar.gz archive from http://www.zdziarski.com/forensic-*toolkit/Bypass_Passcode/*. Extract the contents of the archive and copy the shared library to the appropriate location in */usr/local/lib*:

```
$ tar -zxvf iPHUC_Universal.tar.gz
$ sudo mkdir -p /usr/local/lib
$ sudo mv libMobileDevice742.dylib /usr/local/lib
```

⇒ **Windows**

Download the Windows version of iPHUC. This can be found here: *http://code.google.com/p/iphucwin32/*. Follow the instructions in the archive to prepare an environment using the correct readline and iTunes Mobile Device dynamic libraries.

Step 1: Download the Passcode Bypass RAM disk

This is a customized iPhone RAM disk designed to disable the passcode by deleting its configuration file. The URL is *http://www.zdziarski.com/forensic-toolkit/Bypass_Passcode/Bypass_Passcode.bin*.

In the event that the Bypass_Passcode.bin technique fails, a Bypass_Everything.bin image has also been provided. This image moves the entire preferences folder to /private/var on the file system, bypassing all previously set preferences including passcode. Some particularly finicky devices require use of the Move_Preferences.bin RAM disk instead.

Step 2: Use iPHUC to Enter Recovery Mode

Launch iPHUC from a terminal prompt and instruct the iPhone to enter recovery mode using the `enterrecovery` command. The command should return 0. You must then `exit` the iPHUC command line interface. Commands to be entered are emboldened below.

```
$ ./iPHUC
CFRunLoop: Waiting for iPhone.
notification: iPhone attached.
AMDeviceStartService 'com.apple.afc': 0
(iPHUC) /: enterrecovery
AMDeviceEnterRecovery: 0
(iPHUC) /: exit
Nothing left to do. Exiting.
```

If you are unable to issue this command through iPHUC, cleanly power the device down by holding in the button until the “Slide to Power off” slider comes up. Slide this to power off the device. Once powered down, press the power button, then immediately release. When you see the device power on, press and hold both power and home buttons until the device again power cycles and the restore logo is displayed. This ensures the device was cleanly dismounted, which is required in order to bypass the passcode.

Wait for the iPhone to enter recovery mode. It should display the iTunes icon on the screen. If, after repeated attempts, it does not, check the dock connector to ensure it is secured properly. If all else fails, force the iPhone into recovery mode by holding down the Home and Power buttons simultaneously until the iTunes icon displays on the screen.

Step 3. Upload and Boot the Passcode Bypass RAM Disk

Make sure you have excited iPHUC and then re-launch it to access the recovery options. Issue the commands below to upload and boot the passcode circumvention tool. Be sure to escape the spaces as shown.

```
$ ./iPHUC
(iPHUC Recovery) #: filecopytophone Bypass_Passcode.bin
filecopytophone: 0
(iPHUC Recovery) #: cmd setenv\ boot-args\ rd=md0\ -x\ -s\ pmd0=0x9340000.0xA00000
(iPHUC Recovery) #: cmd saveenv
(iPHUC Recovery) #: cmd bootx
(iPHUC Recovery) #: exit
```

At this stage, the iPhone should boot into verbose mode. The passcode tool will be invoked and move the SpringBoard configuration file located at */private/var/mobile/Library/Preferences/com.apple.springboard.plist*. This property list contains the user’s passcode preferences, which default to “no passcode”. This file is moved to */private/var* for later examination, if so desired. This preserves the original preferences file, but causes the iPhone to, upon reboot, default to having no passcode set.

If successful, you should see the text "Passcode Bypassed” or similar text appear at the bottom of the screen briefly and then the iPhone will reboot back into normal mode. The device should no longer require a passcode.

Should the device’s passcode fail to be circumvented, retry steps 3 and 4, but instead of issuing the `bootx` command, use `fsboot` instead. This may work on older versions of iPhone firmware.

If you see errors concerning mount_hfs, this suggests that the device was not properly shut down. Try powering the device off properly using the “Slide to Power off” method and then, on power on, force the device into recovery mode.

Performing Forensic Recovery

Once the device can be accessed via ssh, it is now ready for recovery.

Recovering the Media Partition

The first step in performing examination is to recover a raw disk image of the media partition. To do this, you will require two UNIX tools: *dd* and *nc*. The *dd* tool is used to copy the raw drive image, while the *nc* tool is used to send the data across the WiFi network to the desktop machine. Both of these tools must be installed on both the desktop and the iPhone. The Forensic-Toolkit payload automatically installs the iPhone builds of these tools, leaving the desktop portion up to the examiner.

The file copy over netcat is insecure unless forwarded through an SSH tunnel. In both cases, for evidentiary integrity, it is recommended that this copy be conducted over a private, encrypted wireless network.

⇒ Mac OS X Leopard includes these tools by default. Open a terminal window by opening the applications folder, opening the utilities folder, and double clicking on the *Terminal* application. Execute '`which dd nc`' to ensure both are visible to your current path.

⇒ Windows versions of these tools may be downloaded at *http://www.chrysocome.net/dd* and http://www.vulnwatch.org/netcat/. An archive is also available at *http://www.zdziarski.com/forensic-toolkit/Archive/*.

Mounting Read-Only

Before transmitting the media partition to the desktop machine, it may be appropriate to remount the partition read-only, and generate an md5 checksum of the raw disk on the device. To do this, connect to the iPhone using ssh and issue the commands below, replacing ***x.x.x.x*** with the IP address of the device:

```
$ ssh -l root x.x.x.x
# cd /
# umount -f /private/var
# mount -o ro /private/var
# md5 /dev/rdisk0s2
```

While the user partition is mounted as read-only, the user interface (via the touch screen) may not be used, except to touch an inactive portion of the screen to keep the backlight active. If, at any time, the operating system layer becomes non-responsive, rebooting the device will cause the user partition to be remounted back in read-write mode. This will allow the operating system to write to the partition, however, and so should this occur, another md5 checksum will need to be made on the device.

Unencrypted Recovery of the Media Partition

You're now ready to perform recovery of the media partition. To do this, you'll need to run separate commands from the desktop and the iPhone to transmit the contents across the network.

On the desktop, instruct the netcat tool to listen on a local port (in this example, 7000). The information sent to the desktop will be piped to the disk copy utility to write to disk:

⇒ **Mac OS X**

```
$ nc -l 7000 | dd of=./rdisk0s2 bs=4096
```

Some versions of netcat built for Mac OS X use the arguments `-l -p 7000`

⇒ **Windows**

```
$ nc -L -p 7000 | dd of=./rdisk0s2 bs=4096
```

Now connect to the iPhone using `ssh` and perform a disk dump. Below, ***x.x.x.x*** represents the IP address of the iPhone, and ***y.y.y.y*** represents the IP address of the desktop machine.

```
$ ssh -l root x.x.x.x
# /bin/dd if=/dev/rdisk0s2 bs=4096 | nc y.y.y.y 7000
```

The raw partition should transfer over the network, and this should be reflected in the size of the file on the local desktop increasing. This operation may take a few hours, depending on the capacity of the iPhone. Only the media portion of the device's storage will be sent, so the actual file size will be less than the advertised capacity. When the file reaches its maximum size, it may be necessary to cancel the operation on the iPhone's side by issuing a `control-c`.

If the operation fails prematurely, ensure that the iPhone is connected to the dock connector and is charging; the iPhone automatically shuts down its WiFi when on battery as it enters sleep mode. If necessary, also set the *Auto-Lock* feature to *never* in the iPhone's general settings to keep the display awake and unlocked.

If the operation fails entirely, check with your systems administrator to ensure that it is not being hindered by firewall policies, and check the desktop machine to ensure its firewall is configured to allow access on the desired port (in this example, 7000).

Once complete, the disk image can be fingerprinted or a checksum created, and checked into a digital vault. It is assumed that all further file operations will be performed on a copy of the disk image.

Never perform examination of an original disk image, but only a copy. Some tools have been known to slightly alter the disk image in the course of their operation, thereby altering the checksum. It is also likely to be altered if mounted as a file system.

Now that the media partition has been copied, the iPhone itself may be analyzed by hand to obtain any information available through the standard interfaces. The next section will cover forensic analysis of the media partition for data that is otherwise unavailable from the GUI.

Encrypted Recovery of the Media Partition

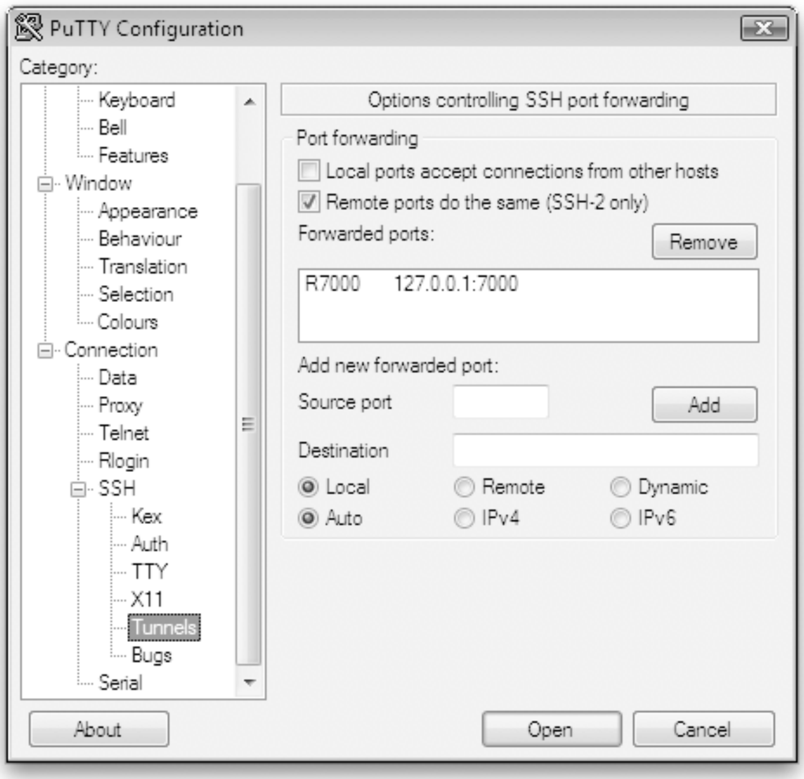
Using a technique similar to the above technique, the disk image can be transmitted across an encrypted SSH tunnel by creating a remote forwarding port to the iPhone. This helps prevent tampering and ensures that the data traveling across the wireless network is encrypted on an application level.

In some cases, certain combinations of the ssh client and server can result in packet size errors. In the event this occurs, try using a different version of ssh on the desktop machine, or revert to using the unencrypted netcat technique described in the last section – it is recommended, however, that the unencrypted technique be performed over an encrypted wireless access point.

When connecting to the iPhone via ssh, add parameters to both compress and remote port-forward data on a given port, where ***x.x.x.x*** represents the IP address of the iPhone:

```
$ ssh -l root -C -R 7000:127.0.0.1:7000 x.x.x.x
```

If using a GUI tool, such as PuTTY, instead of a command line tool, configure a remote port forward as shown in Fig. 5.



Options controlling SSH port forwarding

Port forwarding

[ ] Local ports accept connections from other hosts

[✓] Remote ports do the same (SSH-2 only)

Forwarded ports:

```
R7000     127.0.0.1:7000
```

Add new forwarded port:

Source port

Destination

[✓ Local] Remote Dynamic

[✓ Auto] IPv4 IPv6

Fig. 5 Remote Port Forwarding Configuration in PuTTY

On the desktop, instruct the netcat tool to listen on a local port (in this example, 7000). The information sent to the desktop will be piped to the disk copy utility to write to disk:

⇒ **Mac OS X**

```
$ nc -l 7000 > rdisk0s2
```

⇒ **Windows**

```
$ nc -L -p 7000 > rdisk0s2
```

On the iPhone, perform a disk dump. Instead of using the IP address of the desktop machine, use `127.0.0.1`, which feeds the data through the iPhone's loopback interface, and ultimately back through the reverse tunnel to the desktop.

```
# cat /dev/rdisk0s2 | nc 127.0.0.1 7000
```

The raw partition should transfer over the encrypted SSH tunnel, and this should be reflected in the size of the file on the local desktop increasing. This operation may take a few hours, depending on the capacity of the iPhone. Only the media portion of the device's storage will be sent, so the actual file size will be less than the advertised capacity. When the file reaches its maximum size, it may be necessary to cancel the operation on the iPhone's side by issuing a `control-c`.

If the operation fails prematurely, ensure that the iPhone is connected to the dock connector and is charging; the iPhone automatically shuts down its WiFi when on battery as it enters sleep mode. If necessary, also set the *Auto-Lock* feature to *never* in the iPhone's general settings to keep the display awake and unlocked.

If the operation fails entirely, check with your systems administrator to ensure that it is not being hindered by firewall policies, and check the desktop machine to ensure its firewall is configured to allow access on the desired port (in this example, 7000).

Once complete, the disk image can be fingerprinted or a checksum created, and checked into a digital vault. It is assumed that all further file operations will be performed on a copy of the disk image.

Never perform examination of an original disk image, but only a copy. Some tools have been known to slightly alter the disk image in the course of their operation, thereby altering the checksum. It is also likely to be altered if mounted as a file system.

File Recovery Using Foremost /Scalpel

The *Foremost* tool is a free forensics tool developed by Special Agents Kris Kendall and Jesse Kornblum of the U.S. Air Force Office of Special Investigations. Foremost can be downloaded from *http://foremost.sourceforge.net/* and compiled/installed on most desktop operating systems. Mac OS systems may either build from sources or install using MacPorts:

```
$ sudo port install foremost
```

The *Scalpel* tool is based on Foremost and performs much faster analysis, using an identical configuration file. Scalpel is available at *http://www.digitalforensicssolutions.com/Scalpel/*.

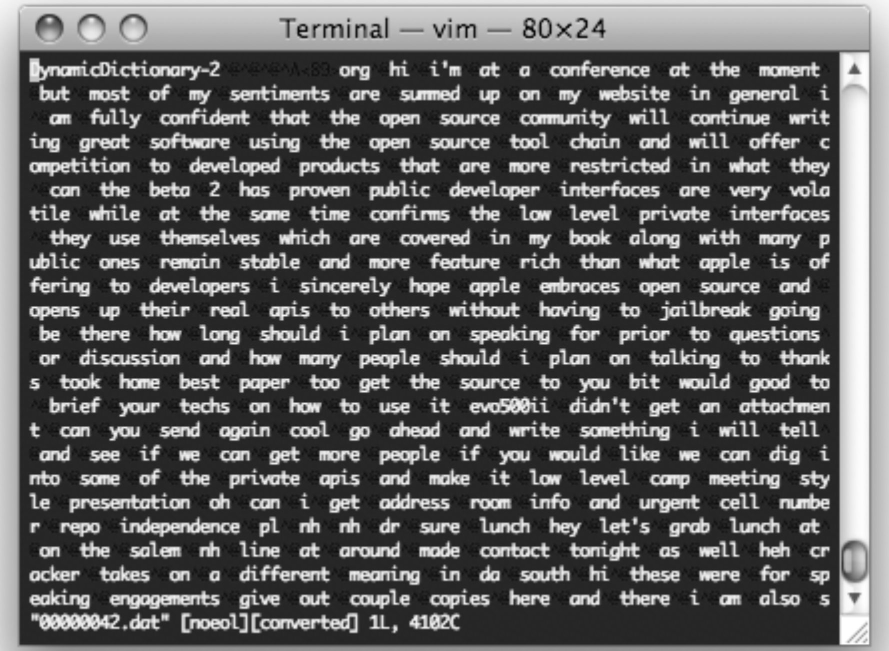
Both tools recover files by scanning for specific headers, footers, and internal data structures of a file. This process is commonly referred to data carving. It is ideal for extracting deleted files from raw disk images, such as the one created in the last section.

Configuring Foremost for iPhone Recovery

The Foremost tool uses a *foremost.conf* file for configuration. Scalpel uses an identical configuration, traditionally named *scalpel.conf*. Either sample configuration file allows the examiner to specify what types of files they would like to extract from the image. Additional files may also be defined in the configuration. The iPhone includes some proprietary file types, which may be of interest to the forensic examiner:

```
        dat        y    16384  DynamicDictionary
```

Dynamic dictionary files are keyboard caches used for learning specific spellings of words used frequently by the iPhone’s user. Whenever a user enters text – whether a username, some passwords, website URL, chat message, email message, or other form of input – many of the words are stored in the keyboard cache. Adding the line above to the configuration file will search for deleted and/or existing caches. An example of such a file is shown below, containing fragments from multiple emails sent across a day’s time period. Also included are various Google® search words (“evo500ii”) and other user input.



```
DynamicDictionary-2          org hi i'm at a conference at the moment
 but most of my sentiments are summed up on my website in general i
  am fully confident that the open source community will continue writ
ing great software using the open source tool chain and will offer c
ompetition to developed products that are more restricted in what they
  can the beta 2 has proven public developer interfaces are very vola
tile while at the same time confirms the low level private interfaces
  they use themselves which are covered in my book along with many p
ublic ones remain stable and more feature rich than what apple is of
fering to developers i sincerely hope apple embraces open source and
opens up their real apis to others without having to jailbreak going
 be there how long should i plan on speaking for prior to questions
 or discussion and how many people should i plan on talking to thank
s took home best paper too get the source to you bit would good to
  brief your techs on how to use it evo500ii didn't get an attachmen
t can you send again cool go ahead and write something i will tell
 and see if we can get more people if you would like we can dig i
nto some of the private apis and make it low level camp meeting sty
le presentation oh can i get address room info and urgent cell numbe
r repo independence pl nh nh dr sure lunch hey let's grab lunch at
 on the salem nh line at around made contact tonight as well heh cr
acker takes on a different meaning in da south hi these were for sp
eaking engagements give out couple copies here and there i am also s
"00000042.dat" [noeol][converted] 1L, 4102C
```

Fig. 6 A deleted, two-week old dynamic keyboard cache

```
        amr         y      65535    #!AMR
```

The AMR codec is considered the standard speech codec by 3GPP. It yields high quality audio playback for voice content, and is used on the iPhone to store voicemail messages. To extract larger chunks of voicemail messages, adjust the file size specified above.

```
        plist       y      4096     <plist  </plist
```

A .plist file is a configuration file used heavily in the Mac OS world, including the iPhone. Many preloaded applications, as well as Apple’s operating system components, use .plist files to store anything from basic configuration data to history and cache information. By examining these files, the examiner can get an idea of what websites the suspect may have previously visited, even after deleting a cache. Other useful information may include location lookup caches (revealing maps the suspect has looked up), mail server information, and etc.

```
        sqlitedb    y      5000000  SQLite\x20format
```

The SQLite database format is used widely in the Mac OS X world, and is used to store calendar, addressbook, Google Maps® tiles, and other information on the iPhone. SQLite databases are generally “live” on the file system, however older, deleted databases may be recovered in the event that the device was restored recently.

```
        email       y      40960    From:
```

Scanning for email headers is an effective way to recover messages.

The following configuration terms are already present in *foremost.conf* but are commented out. These types, in particular, should be considered for inclusion depending on recovery needs.

```
            htm         n      50000    <html  </html>
```

In addition to scanning for email headers, HTML files are used to store email messages as well as web browser content. It is possible to recover deleted email messages by scanning for HTML.

```
            pdf         y      5000000   %PDF-  %EOF
            doc         y      12500000  \xd0\xcf\x11\xe0\xa1\xb1
```

Adobe PDF and Microsoft Word files can be stored locally when sent to the suspect via email or navigated to using the iPhone's Safari web browser.

```
            txt         y      100000  -----BEGIN
```

PGP encrypted messages are generally not of great use without a key, however they can frequently be found on disk to include unencrypted messages within the same thread, should any have been sent/received.

Finally, GIF, JPG, and PNG image formats are all used on the iPhone, and can be enabled for scanning by removing the comments preceding the corresponding lines in the configuration file.

```
            png         y      40960                         \x89PNG
```

This particular format of PNG is used to store small icons and Google Maps® tiles.

Scanning With Foremost/Scalpel

Once a valid configuration file has been created, Foremost/Scalpel can then be instructed to scan the image from a command-line:

```
        $ foremost –c foremost.conf rdisk0s2
        foremost version 0.69
        Written by Kris Kendall and Jesse Kornblum.
        Opening /usr/local/sandbox /rdisk0s2
        rdisk0s2:   0.9% |                      |  130.0 MB    11:07 ETA
```

The entire process may take several hours to complete using `foremost`, or less than an hour using `scalpel`. Potentially useful information will be recovered to a directory named *foremost-output* (or *scalpel-output*) within the current working directory. The tool will also create an *audit.txt* file within the output directory containing a manifest of the information recovered.

Finding Valid Images with ImageMagick

Recovery tools generally err on the side of generating too much data, rather than missing files it believes may be important. As a result, they extract much data that is otherwise unwanted. Finding valid images to examine can be a time consuming process, however a few simple recipes can greatly help reduce the amount of time needed.

The ImageMagick package contains a set of image processing utilities, one of which can be used to display information about images. The *identify* tool included with ImageMagick is perfect for sifting through the thousands of files created by Foremost to identify only the readable images. ImageMagick can be downloaded from *http://www.imagemagick.org/script/index.php*. Mac OS users may build from sources or use MacPorts to install the package:

```
        $ sudo port install imagemagick
```

Once installed, a simple bash script is written to test the validity of an image file. For the purposes of this example, name the file *test-script.sh*:

```
        #!/bin/bash
        mkdir invalid
        identify $1 || mv $1 ./invalid/
```

Alternatively, the script can be modified to delete any image files it does not believe are valid.

```
        #!/bin/bash
        identify $1 || rm –f $1
```

Some images may be corrupt, but still somewhat recognizable. These images may appear to the identify tool as invalid. It is therefore recommended that images only be moved, and not deleted, so that invalid images can be later reviewed by hand.

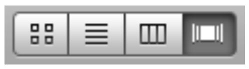
When calling ImageMagick's *identify* tool on a given file, a successful exit code will be returned if the image can be read. In the first example above, all valid images will be moved to a directory named *valid*, leaving the invalid images in the output directory. In the second example, invalid files are deleted, leaving valid images in the output directory. Either script can then be invoked for a given supported image type (jpg, gif, png, etc) using a simple find recipe:

```
$ mkdir valid
$ chmod 755 test-script.sh
$ find foremost-output –type f –name "*.jpg" \
  -exec ./test-script.sh {} \;
```

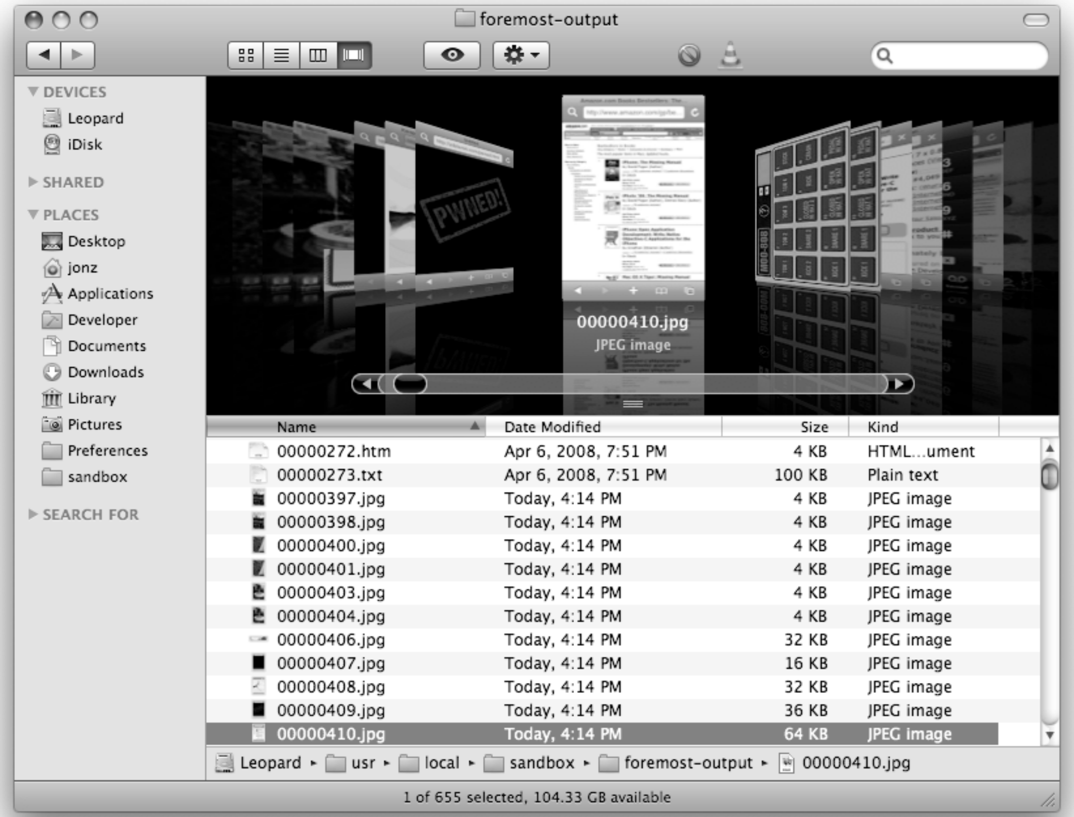
Graphical File Analysis

Both Mac OS X and Windows support preview browsers. Mac OS X, in particular, provides a very useful interface for browsing the contents of the Foremost output directory in a graphical manner.

Using Mac OS, browser to the *foremost-output* folder. At the top of the finder window, a series of buttons should be visible. Click the right-most icon, designated as Cover Flow:



The contents of the directory will now appear in a graphical representation, including previews of images, HTML, and other readable files. The entire directory can now be visually scrolled, saving time:



Name	Date Modified	Size	Kind
00000272.htm	Apr 6, 2008, 7:51 PM	4 KB	HTML...ument
00000273.txt	Apr 6, 2008, 7:51 PM	100 KB	Plain text
00000397.jpg	Today, 4:14 PM	4 KB	JPEG image
00000398.jpg	Today, 4:14 PM	4 KB	JPEG image
00000400.jpg	Today, 4:14 PM	4 KB	JPEG image
00000401.jpg	Today, 4:14 PM	4 KB	JPEG image
00000403.jpg	Today, 4:14 PM	4 KB	JPEG image
00000404.jpg	Today, 4:14 PM	4 KB	JPEG image
00000406.jpg	Today, 4:14 PM	32 KB	JPEG image
00000407.jpg	Today, 4:14 PM	16 KB	JPEG image
00000408.jpg	Today, 4:14 PM	32 KB	JPEG image
00000409.jpg	Today, 4:14 PM	36 KB	JPEG image
00000410.jpg	Today, 4:14 PM	64 KB	JPEG image

Leopard ▸ usr ▸ local ▸ sandbox ▸ foremost-output ▸ 00000410.jpg

1 of 655 selected, 104.33 GB available

Fig. 7 Cover-Flow View of Recovered Data (Mac OS X)

Many image files are likely to appear more than once, as they are sometimes rewritten when the iPhone syncs with its desktop. Album covers are likely to appear several times – once for each song.

Images of Interest

Recovery of image files should provide some of the following images of interest:

- Photos taken with the iPhone's camera
- Photos synced to the device from a desktop photo library
- Photos from the web browsing cache / history
- Multiple snapshots of running applications in the last state before they were exited/suspended, including:
    - Web browser "last page" visited (as shown in Fig. 7)
    - Contacts and dialer application screens
    - Google® Maps and YouTube "last viewed" captures

Many other images will also be recovered which are not necessarily useful, such as album covers (one cover per song), operating system images, and other stock images. Paths to “live” versions of these various images on the file system will be provided in the next section.

Electronic Discovery

In addition to performing forensic recovery, the “live” file system at the time it was imaged can also be browsed. This will allow the examiner to recover the most recent information on the device.

⇒ **Mac OS X**

The *rdisk0s2* disk image can be renamed to have a *.dmg* extension, and then directly mounted from the finder:

```
$ mv rdisk0s2 rdisk0s2.dmg
$ hdid rdisk0s2.dmg
```

⇒ **Windows**

The rdisk0s2 disk image can be converted to an ISO image using the *dmg2iso* tool available at http://vu1tur.eu.org/tools/. Many commercial forensic tools are also able to read and manage HFS+ file systems.

```
$ rename rdisk0s2 rdisk0s2.dmg
$ perl dmg2iso.pl rdisk0s2.dmg rdisk0s2.iso
```

SQLite

The device makes use of database files in SQLite v3 format to store much information. SQLite is a public domain database utility and can be downloaded at http://www.sqlite.org. SQLite can be used on the command line to easily access the individual files and issue SQL queries. Seek a guide on the structured query language for more information about SQL. The most basic commands needed by the examiner are described below.

Opening a database

To open a SQLite database, install SQLite and then invoke the `sqlite3` tool from the command line. The tool will dump you to an input prompt, allowing queries to be issued:

```
$ sqlite3 filename
SQLite version 3.4.0
Enter ".help" for instructions
sqlite>
```

Querying the database

Once opened, the following commands may be issued against the open database. For more information about these and other commands, see the SQLite document at http://www.sqlite.org.

`.tables`

Lists all of the tables available within the database

`.schema` ***table-name***

Displays the SQL statement used to construct the table specified. This displays every column in the table and its data type. The following example queries the schema for the `mailboxes` table inside the `Envelope Index` database used to store mail on the device:

```
sqlite> .schema mailboxes
CREATE TABLE mailboxes (ROWID INTEGER PRIMARY KEY,
                        url UNIQUE,
                        total_count INTEGER DEFAULT 0,
                        unread_count INTEGER DEFAULT 0,
                        deleted_count INTEGER DEFAULT 0);
```

`.dump` ***table-name***

Dumps the entire contents of a table into SQL insert statements. Binary data is outputted as long hexadecimal sequences which can later be converted to individual bytes with simple scripting.

`.output` ***filename***

Redirects output to a file on disk. Useful when dumping data.

`.headers on`

Turns display headers on so that the column title will be displayed in output from a `SELECT` statement

`SELECT * FROM` ***table-name;***

Returns all fields and columns stored within the table specified. If the display headers were turned on, the first row returned will contain the column names. Be sure to end the statement with a semicolon (`;`). The following example queries the actual records from the mailboxes table, displaying the existence of an IMAP mailbox located at imap.domain.com. This mailbox contains 3 total messages, all of which have been read, and 0 deleted messages.

```
sqlite> SELECT * FROM mailboxes;
1|imap://username@imap.domain.com/INBOX|3|0|0
2|local:///Outbox|0|0|0
```

`.exit`

Exits the SQLite command shell.

Property Lists

Property lists are XML manifests used to describe various configurations, state, or other information. Property lists can be formatted in either ASCII or binary format. When formatted for ASCII, the file can be easily read using any standard text editor. When formatted for binary, the file must be opened by an application capable of reading or converting the format to ASCII.

⇒ **Mac OS X**

Mac OS X comes standard with a tool named Property List Editor. This can be invoked by simply double-clicking on a file ending with a `.plist` extension

⇒ **Windows**

- An online tool is available to convert property lists to ASCII format. The tool can be found at *http://140.124.181.188/~khchung/cgi-bin/plutil.cgi*.
- The property list converter is open source and available on Apple's website, where it may be downloaded and compiled. The source code can be found at *http://www.opensource.apple.com/darwinsource/10.4/CF-368/Parsing.subproj/CFBinaryPList.c*. An Apple developer account will be required and is free to register.

Important Files

Though each case may call for different evidence, the following files are generally useful for most types of examination.

Database files may still contain deleted records in de-allocated portions of the file. Be sure to examine the files using a hex editor or other analysis tool to discover any additional information.

/mobile/Library/AddressBook/AddressBook.sqlitedb

/mobile/Library/AddressBook/AddressBookImages.sqlitedb

SQLite database containing address book entries and images, respectfully

/mobile/Library/Caches/MapTiles/MapTiles.sqlitedb

SQLite database containing Google® Maps tile cache. Contains image data of previously displayed map tiles for the *Maps* application. Each record contains an X, Y coordinate on a virtual plane at a given zoom level and a binary data field containing the actual image data in PNG formatted images. See the section Recovery of Google Maps® Tiles for more information.

/mobile/Library/Calendar/Calendar.sqlitedb

SQLite database containing calendar events, times, and descriptions.

/mobile/Library/CallHistory/call_history.db

SQLite database containing an exhaustive call record. This database contains more phone numbers than are displayed through the normal GUI interface. The database logs each call, phone number dialed, timestamp, duration in seconds, and other call flags.

/mobile/Library/Cookies/Cookies.plist

Properly list (ASCII format) containing website cookies from the Safari web browser

/mobile/Library/Keyboard/dynamic-text.dat

Binary keyboard cache containing text entered by the user.

The text displayed may be out of order or consist of various “slices” of different threads assembled together. View using a hex editor or a paging utility such as `less`.

/mobile/Library/LockBackground.jpg

The current background wallpaper set for the device

/mobile/Library/Mail/Accounts.plist

Property list (binary format) containing email server account information and additional directories within the `Mail` directory where additional email is stored.

/mobile/Library/Mail/Envelope Index

SQLite database containing information about messages stored locally on the device. This database includes message headers, mailboxes, and the message data itself. This database contains six tables: `mailboxes`, `messages`, `message_data`, `properties`, `pop_uids`, and `threads`.

Non-local mail, such as that from an IMAP mailbox, is stored in a separate directory structure specified in */mobile/Library/Mail/Accounts.plist*

/mobile/Library/Maps/History.plist

Property list (ASCII format) containing Google® Maps history, including longitude and latitude of various lookups, query name (if specified), zoom level, and the name of the city or province where the query was made.

/mobile/Library/Notes/notes.db

SQLite database containing note bodies and various information about all notes stored in the device’s *Notes* application.

/mobile/Library/Preferences

Various preferences files containing configuration data for applications and services on the device.

/mobile/Library/SMS/sms.db

SQLite database containing information about SMS messages on the device including phone number, timestamp, actual text, and various carrier information.

/mobile/Library/Safari/Bookmarks.plist

Property list (binary format) containing all web browser bookmarks set on the device. These may have been set either directly on the device, or by syncing with a desktop machine.

/mobile/Library/Safari/Bookmarks.plist.anchor.plist

Property list (binary format) containing the timestamp for when bookmarks were last modified.

/mobile/Library/Safari/History.plist

Property list (binary format) containing the web browser history stored on the device.

/mobile/Library/Safari/SuspendState.plist

Propery list (binary format) containing the last state of the web browser when it was suspended. This contains a list of windows and web sites that were open so that the device can re-open them should the brower be restarted.

/mobile/Library/Voicemail/

Voicemail recordings in AMR codec are stored in this directory using the *.amr* file extension.

/mobile/Library/Voicemail/voicemail.db

SQLite database containing information about the senders of the voicemail stored on the device. Includes the sender's phone number, timestamp, callback number, message duration, expiration of the message, and the timestamp (if any) that the message was moved to the trash.

/mobile/Media/WebClips

Contains a list of web pages assigned as buttons on the device's home screen. Each page will be housed in a separate directory containing a property list named `Info.plist`. This property list contains the title and website URL of each page.

/mobile/Media/DCIM/100APPLE

Photos taken with the device's built-in camera and accompanying thumbnails

/mobile/Media/iTunes_Control/Music

Location of all music synced with the device

/root/Library/Lockdown/data_ark.plist

Property list (ASCII format) containing various information about the device and its account holder. This includes the owner's Apple Store ID, specified with `com.apple.mobile.iTunes.store-AppleID` and `com.apple.mobile.iTunes.store-UserName`, time zone information, SIM status, device name as it appears in iTunes, and firmware revision.

/root/Library/Lockdown/pair_records

This directory contains private keys used for pairing the device to a desktop machine. These records can be used to prove that a specific desktop machine was paired with the device at a given time. Certificates from this file will match certificates located on the desktop machine in one of the property lists located in either /Users/**username**/Library/Lockdown (Mac OS X) or *:\Documents and Settings\**Username**\Local Settings\Application Data\Apple Computer\Lockdown* (Windows).

On iPhone firmware versions <= 1.1.2, the *mobile* directory is replaced with *root*

Recovery of Google Maps® Tiles

The Google Maps® application stores a cache of all viewed map tiles as well as a cache of all lookups performed. The lookup cache is stored at the path */mobile/Library/Maps/History.plist* on the user partition, and can therefore be easily extracted. Actual map tiles, however, reside in a SQLite database stored at the path */mobile/Library/Caches/MapTiles/MapTiles.sqlitedb* on the user partition. To extract the actual images, perform the following steps:

1. Install the command-line SQLite client available at *http://www.sqlite.org*.

2. Copy the *MapTiles.sqlitedb* file onto the desktop machine and dump the `images` table using the command line client, as shown below. This will create a new file named maptiles.sql, which will contain SQL insert statements containing the map tile data, including the binary image data represented in hexadecimal format.

```
$ sqlite3 MapTiles.sqlitedb
SQLite version 3.4.0
Enter ".help" for instructions
sqlite> .output maptiles.sql
sqlite> .dump images
sqlite> .exit
```

3. If necessary, install Perl. Use the parse_maptiles.pl script available at *http://www.zdziarski.com/forensic-toolkit/Scripts/* to convert the file to a set of PNG images. These will be created in a directory named `maptiles` under the current working directory.

```
$ perl parse_maptiles.pl maptiles.dat
```

4. Each map tile will be extracted and given the name X,Y@Z, denoting the X, Y position on a plane and the zoom level; each zoom level essentially constitutes a separate plane.

Desktop Trace

Recovering data from an iPhone or iPod Touch device can be an important step in building evidence for a case, however information on desktop machines having been synced with the device is also of interest. Desktop information can provide evidence of trusted pairing relationships as well as store backup copies of various data files that can be used as both evidence and to further prove a relationship between the desktop and mobile device.

Desktop trace should be gathered through standard forensic recovery procedures applied to the desktop machine. Both live and deleted data can be of great use to the examiner. In this section, the types of relevant data present on the desktop will be described.

Proving Trusted Pairing Relationships

Proving that the device was paired with a particular desktop machine can be of vital importance. When paired with an iPhone or iPod Touch, the desktop and the device share certificate records. In some cases, the serial numbers of devices paired with a particular desktop can also be stored.

Pairing Records

In the last section, the directory */root/Library/Lockdown/pair_records* was mentioned as containing pairing records. Certificates inside of these pairing records are shared with the desktop machine(s) they are paired with. For example, the pairing record on our test device was located at:

/var/root/Library/Lockdown/pair_records/38798B80-D800-4691-916A-01640D8CECCD.plist

This pairing record contained the following certificate:

```
<key>DeviceCertificate</key>
        <data>
        LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNOakNDQVI2Z0F3SUJBZ0lCQURB
        TkJna3Foa2lHOXcwQkFRVUZBREFBTUI0WERUQTRNRFF3T0RFek1qUXkKTlZvWERURTRN
        RFF3TmpFek1qUXlOVm93QURDQm56QU5CZ2txaGtpRzl3MEJBUUVGQUFPQmpRQXdnWWtD
        Z1lFQQp3djBzSDgycW9pcFM4Z2hZSnJPV1BLT0U3UUR5QmIxTkpuRmF2eDZEVVdwWGEx
        NXhmN2JiN2VaVlAzaXZrZGtUCkpBd0FPM1puT0pGQTBFUzU4NzlBTnVDM1R6cFpOT29S
        WFBhZWNlU3BmSG1RWEN6RUdCdUNDb0E5TmYwSWwxSjgKYUcxdnZPUjZTbWdFNE9ES2da
        by9UdGcybHIzTlRUSGlFbmVUWTJpSHp1OENBd0VBQWFNL01EMHdEQVlEVlIwVApBUUgv
        QkFJd0FEQWRCZ05WSFE0RUZnUVU0dnpKcGpUMDloNEVPZHFuUi9mTjVmYVhVZDB3RGdZ
        RFZSMFBBUUgvCkJBUURBZ1dnTUEwR0NTcUdTSWIzRFFFQkJRVUFBNElCQVFCa256SUZP
        ZFBYcUkrSGQ0KzJNdDRjQTM2QWgwVDgKY0NVVDJ2ZnF6WExIL3k2OFZFdnJkbU5zR1V5
        YmMwN0g4V2lIb1FtaDROMDFPdE5uNFpOUUdzK2k1QmxSRHRFcwpxUnJtanRNdGFGMkh2
        NFRpdGlBcWtsRXl3cHY2azRLRFlRUkN5OTB1MCtQbTkwempzRy8zTzR5eHJhdk51Y05M
        CnFjalRGN0hHbmZ2Y2tGSVBYeGlSMlBhb2dySUxGLytpbDVGcThIVWxldW5qbnAwbElz
        T3lqQ29sbyt4c2NpeDgKZ0FIU2pJMDBvdU85cTVkSFc2cmRRRGlKaXlLbDRUd1dOeDJH
        VEU4Sm1PZmRteFgwb21MQ2RXNWUyN0JGTHNnVgprZWh2bzZlWlpuK3EyWU5NWDFkaTNt
        akx6aHFHRXRHUisxZk5RSUtDUWEzN3ptY3lpWUtHeDFmOAotLS0tLUVORCBDRVJUSUZJ
        Q0FURS0tLS0tCg==
        </data>
```

This certificate was also found on a desktop machine paired with the device. In this case, the certificate was located in a property list named *d5d9f86cfc06f8ace3d31c551ccc69788c4579e5.plist* located on the desktop machine. The filename refers to the unique identifier assigned to the iPhone device when it was activated. See the section *Activation Records* for more information on matching the unique device identifier itself.

The location of the pairing files on a machine depend on the operating system:

⇒ **Mac OS X**

*/Users/**username**/Library/Lockdown/*

⇒ **Windows XP**

*:\Documents and Settings\**Username**\Local Settings\Application Data\Apple Computer\Lockdown*

⇒ **Windows Vista**

*:\Users\**Username**\AppData\Roaming\Apple Computer\Lockdown*

Use of tools such as `grep` and `diff` can make manual matching a relatively effortless process.

Serial Number Records

A manifest is written to the desktop machine's hard disk to keep track of the names and serial numbers of devices paired with it, allowing the examiner to verify that a desktop has trace evidence of a pairing with a mobile device. This manifest can be used to make a visual match between the serial number recorded in the file and the serial number of the mobile device. The serial number of the mobile device can be obtained by tapping the settings button on the device and then selecting `General` > `About`.

⇒ **Mac OS X**

A binary property list with a filename beginning with *com.apple.iTunes* may be found in the directory located at */Users/**username**/Library/Preferences/ByHost/*. Each host paired with the device will be assigned a separate file in this directory. Inside the property list, binary data containing information about the device is stored, but by using the `strings` tool, the examiner can dump the ASCII data encapsulated within the binary information and search for the presence of the mobile device's serial number.

⇒ **Windows XP**

A match to this serial may be found in the *iPodDevices.xml* file located on the Windows desktop machine, found at *:\Documents and Settings\**Username**\Local Settings\Application Data\Apple Computer\iTunes\iPodDevices.xml*

⇒ **Windows Vista**

A match to this serial may be found in the *iPodDevices.xml* file located on the Windows desktop machine, found at *:\Users\**Username**\AppData\Local\Apple Computer\iTunes\iPodDevices.XML*

The serial number can also be found in device backup files, if they exist on the desktop machine. See the next section for more information. This will be explained in the next section.

Device Backups

When a device is synced with a desktop machine, a backup of its configuration, address book, SMS database, camera photo cache, and other contents are stored locally. Each device paired with the desktop is assigned a unique identifier within the user's backup directory. Within this directory resides a manifest, device information, and individual data files. These are copied back to the device in the event that the device is restored to its factory settings. While a suspect can manually delete such backups, many are not aware that such backups are being made, or choose to store the backups.

⇒ **Mac OS X**

Device backups are stored in

*/Users/**username**/Library/Application Support/MobileSync/Backup/**deviceid**/*

⇒ **Windows XP**

Device backups are stored in

*:\Documents and Settings\**Username**\Application Files\MobileSync\Backup\deviceid\*

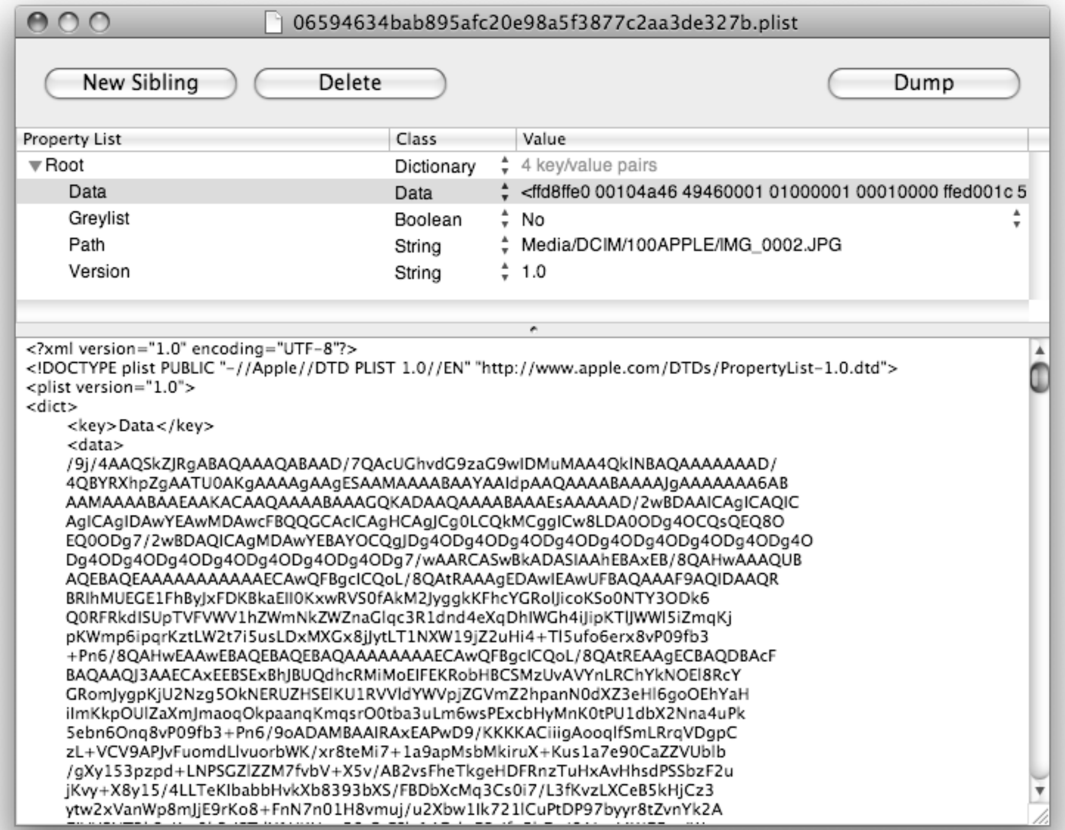
⇒ **Windows Vista**

Device backups are stored in

*:\Users\**Username**\AppData\Roaming\Apple Computer\MobileSync\Backup\deviceid\*

The information file, *Info.plist*, contains a device profile including the serial number of the paired device, firmware revision, phone number, and timestamp. Within this directory, multiple files ending with a *.mdbackup* extension will exist. Each file is a binary property list containing the filename and binary data for a file backed up from the device. The binary data can be extracted by dumping it from the property list using a property list reader or by manual techniques. A copy of the file can be renamed to have a *.plist* file extension, allowing it to be opened with a property list editor.

Once the binary data has been exported from a binary property list, and stored in a file specified as the filename within the property list, it can be analyzed using the techniques described in the section *File System Analysis*.



Property List	Class	Value
▼Root	Dictionary	4 key/value pairs
Data	Data	<ffd8ffe0 00104a46 49460001 01000001 00010000 ffed001c 5
Greylist	Boolean	No
Path	String	Media/DCIM/100APPLE/IMG_0002.JPG
Version	String	1.0

```
<?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>Data</key>
	<data>
	/9j/4AAQSkZJRgABAQAAAQABAAD/7QAcUGhvdG9zaG9wIDMuMAA4QkINBAQAAAAAAAD/
	4QBYRXhpZgAATU0AKgAAAAgAAgESAAMAAAABAAYAAIdpAAQAAAABAAAAJgAAAAAAAA6AB
	AAMAAAABAAEAAKACAAQAAAABAAAGQKADAAQAAAABAAAEsAAAAAD/2wBDAAICAgICAQIC
	AgICAgIDAwYEAwMDAwcFBQQGCAcICAgHCAgJCg0LCQkMCggICw8LDA0ODg4OCQsQEQ8O
	EQ0ODg7/2wBDAQICAgMDAwYEBAYOCQgJDg4ODg4ODg4ODg4ODg4ODg4ODg4ODg4ODg4O
	Dg4ODg4ODg4ODg4ODg4ODg4ODg7/wAARCASwBkADASIAAhEBAxEB/8QAHwAAAQUB
	AQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQR
	BRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6
	Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKj
	pKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3
	+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcF
	BAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcY
	GRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaH
	iImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk
	5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD9/KKKKACiiigAooqlfSmLRrqVDgpC
	zL+VCV9APJvFuomdLlvuorbWK/xr8teMi7+1a9apMsbMkiruX+Kus1a7e90CaZZVUblb
	/gXy153pzpd+LNPSGZIZZM7fvbV+X5v/AB2vsFheTkgeHDFRnzTuHxAvHhsdPSSbzF2u
	jKvy+X8y15/4LLTeKlbabbHvkXb8393bXS/FBDbXcMq3Cs0i7/L3fKvzLXCeB5kHjCz3
	ytw2xVanWp8mj/u2Xbw1lk721H9rKo8+FnN7n01H8vmuj/u2Xbw1lk721lCuPtDP97byr8tZvnYk2A
```

Fig. 8 Extracting a Camera Photo from a Desktop Backup File

Activation Records

Various information about the device can be obtained by decoding activation records found in *\/privat/var/root/Library/Lockdown/activation_records* on the device, which will be accessible as */root/Library/Lockdown/activation_records* if the user partition disk image is mounted locally. This information is base64 encoded and can be easily decoded to plain text using the *openssl* command line tool or other base64 decoder. The property list contained in this directory includes several different certificates including the FairPlay certificate for music on the device, however the most useful section is the *AccountToken* data. This data follows after the *AccountToken* key referenced inside the property list:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>AccountToken</key>
        <data>
        ... data follows ...
```

The data section, when decoded, contains the unique device ID assigned when the pairing relationship is made. This identifier will determine the filename of pairing records on the desktop machine. An activation ticket and hardware identities (including the IC Card, mobile subscriber, and mobile equipment identity) are also stored. For example, pasting the data portion of the *AccountToken* data into a file, the decoding process is as follows:

```
$ openssl enc -d -base64 -in filename
{
        "ActivationRandomness" = "AEC80D06-1948-494C-846E-9A9FC02CF175";
        "UniqueDeviceID" = "d5d9f86cfc06f8ace3d31c551ccc69788c4579e5";
        "ActivationTicket" =
"0200000029338284e1a7309dd143c60aa20a7176fba9d1db44860ba2e8b214c471e3d06b92089c068
26dcc7a4f06e8200228d974cf6b5518baebe3457ccaffe9395a81d5a94a8e3a7c1c71746aaebc39d9d
dc3acf2fd359448dd2d2379782606a4eec99e62298c26439d299606bbadb00d9439b63cfed42921f76
7d8316ce42e212082c58a1e5ee1fb619e0fb2f753b0f86a2db7cace003e5a47efb32a2b4e33d1787d0
f6681edfc0737877ee6a28cec242418402cfda695060bd75f396c909c0b1ba3236519d29291012fbda
dd2c8d0d7caae1ea33ac6841b3b6d64ca69145f7b072304a4f980d907d10b18bee9dd5df8cd8aea6ff
11b339e8cc34d7f572c6de69c53076e8a4f057e46cf6ebe879480f62e1f966abb1f05049b328a3cb47
d7208521901e6772c393251f13ce9ed9daaf21240617a89a813e7c48dbacd099d84979984deecc01e8
42da38a199e9e6ef67b84325f18a73c2f9f0fb4c11ce4933eed7728960ad637565e5589dc0faeb84a2
8990d71fceb0757f9131e4c151a48df520d427a66c2d2f2d0d4270d4e756c9baa9600da7f62f8dacf7
ab83bb454d5e48e078bad04ade6b98661859c3e9606a5e983a8f7e37d8fac3b9cc091d518e5b153e84
04486533bfc1aa20af4a6633245bc2de2afbf820f9065bae956690481d0df591dc1073011e6caf8d47
f8278f7a0d526a14948c33cc8f252e03c40d6f91c9a6229770eac49b2498630a468061892420518576
dfc0e045598475b68cedb071e1bf41476569da801081a39e7e658698bb54875ba74ed0af5c95c3fe03
7b9c8f5f547c926baa9dd055a4264";
        "IntegratedCircuitCardIdentity" = "89014103211656554643";
        "InternationalMobileSubscriberIdentity" = "310410165655464";
        "InternationalMobileEquipmentIdentity" = "011472002196598";
```

Technical Procedure

In this section, the low level technical details used by the iLiberty+ tool will be explained. These techniques are intended for those desiring a technical explanation of the procedure or who seek to reproduce or re-implement it, and is not necessary information for general forensic examination.

Many different methods have been devised by the iPhone development community to gain access to an iPhone or iPod Touch's operating system, however very few of them are able to do so without destroying evidence, or the entire file system altogether. The technique used in this manual is one considered to be forensically safe in that it is capable of accessing the device without corrupting user data.

This technique gains access to the operating system by booting an unsigned RAM disk from the iPhone's resident memory. This RAM disk is copied into the iPhone's memory and booted using the Apple's private MobileDevice framework. Version 7.4.2 of the device framework is specifically used here, and the procedure changes for newer versions of the framework. You will therefore require this framework from a copy of iTunes 7.4.2 in order to reproduce the procedure.

Once the unsigned RAM disk is booted, it is then capable of mounting the device's system partition and install a payload to enable shell access, surveillance, or any other type of package. When the device boots back into its normal operating mode, the installed payload will be executed, thereby granting access to the device. A custom RAM disk is used in order to install this recovery payload. The RAM disk is a disk image containing the necessary ARM-architecture files to boot and install such a custom payload on the iPhone. The RAM disk itself is padded with 0x800 bytes to contain an 8900 header, and may additionally pad between 0xCC2000 and 0xD1000 zero bytes to assist in aligning the execution space of the disk.

Once a custom RAM disk has been assembled, it is executed using private and undocumented function calls within Apple's *MobileDevice* framework. In short, the procedure involves the following:

1. The device is placed into recovery mode either manually (by holding the `Home` + `Power` buttons until forced into recovery mode), or using the *MobileDevice* function `AMDeviceEnterRecovery`.

2. The RAM disk image is sent to the device using the private `__sendFileToDevice` function after looking up its symbol address in the framework.

3. The following commands are sent to the device using the private `__sendCommandToDevice` function after looking up its symbol address in the framework. This sets the kernel's boot arguments to boot from a RAM disk, and specifies its memory address to the approximate location of the custom image copied to the device in step 1.

```
setenv boot-args rd=md0 -s -x pmd0=0x9340000.0xA00000
saveenv
fsboot
```

Depending on the capacity and firmware version of the device, different memory addresses may be necessary. The memory address `0x09CC2000.0x0133D000` has also been reported to succeed.

4. Once the RAM disk has booted, and the payload has been delivered, the device can be booted back into normal operating mode by sending the following commands to the device using `__sendCommandToDevice`:

```
setenv boot-args [Empty]
setenv auto-boot true
saveenv
fsboot
```

Depending on the version of iPhone firmware, the `fsboot` command may be replaced with `bootx`.

Source Code Examples

The following source code illustrates the process of booting an unsigned RAM disk in C. The example waits for the device to be connected in recovery mode and then issues the commands to send and boot a RAM disk as described in the last section. The RAM disk image and needed framework library are provided by the implementer. This code was designed to run on the Mac OS X operating system running iTunes 7.4.2 *MobileDevice* framework. Comments are provided in-line.

To build this example, use the following command:

```
$ gcc –o inject-ramdisk inject-ramdisk.c –framework CoreFoundation –framework
MobileDevice –F/System/Library/PrivateFrameworks
```

The complete code for `inject-ramdisk.c` follows:

```
#include <stdio.h>
#include <mach-o/nlist.h>
#include <CoreFoundation/CoreFoundation.h>

/* Path to the MobileDevice framework is used to look up symbols and offsets */
#define MOBILEDEVICE_FRAMEWORK
"/System/Library/PrivateFrameworks/MobileDevice.framework/Versions/A/MobileDevice"

/* Used as a pointer to the iPhone/iTouch device, when booted into recovery */
typedef struct AMRecoveryModeDevice *AMRecoveryModeDevice_t;

/* Memory pointers to private functions inside the MobileDevice framework */
typedef int(*symbol)  (AMRecoveryModeDevice_t, CFStringRef) \
    __attribute__ ((regparm(2)));
static symbol sendCommandToDevice;
static symbol sendFileToDevice;

/* Very simple symbol lookup. Returns the position of the function in memory */
static unsigned int loadSymbol (const char *path, const char *name)
{
    struct nlist nl[2];
    memset(&nl, 0, sizeof(nl));
    nl[0].n_un.n_name = (char *) name;
    if (nlist(path, nl) < 0 || nl[0].n_type == N_UNDF) {
        return 0;
    }
    return nl[0].n_value;
}

/* How to proceed when the device is connected in recovery mode.
 * This is the function responsible for sending the ramdisk image and booting
 * into the memory location containing it. */

void Recovery_Connect(AMRecoveryModeDevice_t device) {
    int r;

    fprintf(stderr, "Recovery_Connect: DEVICE CONNECTED in Recovery Mode\n");
```

```
        /* Upload RAM disk image from file */
        r = sendFileToDevice(device, CFSTR("ramdisk.bin"));
        fprintf(stderr, "sendFileToDevice returned %d\n", r);

        /* Set the boot environment arguments sent to the kernel */
        r = sendCommandToDevice(device,
            CFSTR("setenv boot-args rd=md0 -s -x pmd0=0x9340000.0xA00000"));
        fprintf(stderr, "sendCommandToDevice returned %d\n", r);

        /* Instruct the device to save the environment variable change */
        r = sendCommandToDevice(device, CFSTR("saveenv"));
        fprintf(stderr, "sendCommandToDevice returned %d\n", r);

        /* Invoke boot sequence (bootx may also be used) */
        r = sendCommandToDevice(device, CFSTR("fsboot"));
        fprintf(stderr, "sendCommandToDevice returned %d\n", r);
    }

    /* Used for notification only */
    void Recovery_Disconnect(AMRecoveryModeDevice_t device) {

        fprintf(stderr, "Recovery_Disconnect: Device Disconnected\n");
    }

    /* Main program loop */
    int main(int argc, char *argv[]) {
        AMRecoveryModeDevice_t recoveryModeDevice;
        unsigned int r;

        /* Find the __sendCommandToDevice and __sendFileToDevice symbols */
        sendCommandToDevice = (symbol) loadSymbol
            (MOBILEDEVICE_FRAMEWORK, "__sendCommandToDevice");
        if (!sendCommandToDevice) {
            fprintf(stderr, "ERROR: Could not locate symbol: "
                "__sendCommandToDevice in %s\n", MOBILEDEVICE_FRAMEWORK);
            return EXIT_FAILURE;
        }
        fprintf(stderr, "sendCommandToDevice: %08x\n", sendCommandToDevice);

        sendFileToDevice = (symbol) loadSymbol
            (MOBILEDEVICE_FRAMEWORK, "__sendFileToDevice");
        if (!sendFileToDevice) {
            fprintf(stderr, "ERROR: Could not locate symbol: "
                "__sendFileToDevice in %s\n", MOBILEDEVICE_FRAMEWORK);
            return EXIT_FAILURE;
        }

        /* Invoke callback functions for recovery mode connect and disconnect */
        r = AMRestoreRegisterForDeviceNotifications(
            NULL,
            Recovery_Connect,
            NULL,
            Recovery_Disconnect,
            0,
            NULL);
        fprintf(stderr, "AMRestoreRegisterForDeviceNotifications returned %d\n", r);
        fprintf(stderr, "Waiting for device in restore mode...\n");

        /* Loop */
        CFRunLoopRun();
    }
```

Once the RAM disk has been injected and booted, the operation has been complete, and whatever payload the RAM disk was written to deliver has been delivered. The device can then be returned to normal operating mode by issuing the following commands in place of those in the Recovery_Connect function:

```
        /* Reset and save the default boot-related environment variables */
        sendCommandToDevice(device, CFSTR("setenv auto-boot true"));
        sendCommandToDevice(device, CFSTR("setenv boot-args "));
        sendCommandToDevice(device, CFSTR("saveenv"));

        /* Boot the device (bootx may also be used) */
        sendCommandToDevice(device, CFSTR("fsboot"));
```

The device will now boot into normal operating mode for all subsequent boots.

Revision History

Rev. 0	Initial Release
Rev. 1	Formatting Improvements
	Addition of AMR Foremost rule
	Added note about /var/mobile <= 1.1.2
	Added this Change History
	Added Captions
	Image Conversion to B/W
Rev. 2	Added note about iLiberty detection and iTunes
	Updated screen captures
	Simplified instructions by adding custom Forensic-Toolkit payload
	Added information for custom jailbreak payload for v1.51 Mac
	Removed warnings about SSH key generation (now generated on toolkit install)
Rev. 3	Minor technical enhancements
	Elaborated on Windows instructions
	Included Windows screenshots
Rev. 4	Added Scalpel as faster variant of Foremost
	Added “images of interest” section
Rev. 5	New document versioning system
	Editorial changes
	Added information about restore mode and evidence destruction
	Elaborated on update mode
	Provided links to Apple firmware packages
Rev. 6	Added *Power-On Device Modifications* and accompanying audit
Rev. 7	Elaborated on Windows toolkit installation and caveats
	Provided forced-recovery mode installation technique, for troublesome devices
	Added section regarding cross-contamination by syncing
	Added instructions for booting devices out of recovery mode
	Updated iLiberty+ for Windows documentation for manual payload activation
	Changed links to new forensic toolkit repository URLs
Rev. 8	Added low-level technical procedure for booting unsigned RAM disk
	Editorial changes
Rev. 9	Added source code examples
	Moved low-level technical procedures to end of manual
Rev. 10	Minor corrections to *Power-On Device Modifications*
	Added section *Encrypted Recovery of the Media Partition*
	Added section *Determining Firmware Version*
	Expanded section *File System Analysis*
	Added section *Proving Evidentiary Pairing Relationships*
Rev. 11	Mentioned use of virtual machines in section *Cross-Contamination of Evidence and Syncing*
	Added section *Device Backups* to new heading *Desktop Trace*
Rev. 12	Corrected netcat command line arguments for Windows
	Added remote port forwarding example for PuTTY (GUI SSH client)
	Added section *Activation Records*
	Added additional information for Windows Vista file paths
	Formatting fixes
	Elaborated on various warnings and explanations
	Added new discovery that user data is preserved after a full restore
	Added alternative OS X arguments for netcat
	Added file installation disclosure (for files written safely)
	Added instructions for mounting media partition read-only
	Added instructions for md5 checksum of read-only partition (requires 0507 payload or newer)
Rev. 13	Added section *Circumventing Passcode Protection*
	Added icon and Map Tile format for PNG recovery with Scalpel/Foremost
	Added additional sqlite3 syntax
	Added Scaplel/Foremost rule for SQLite databases
	Added recovery of Google Maps® Tile images