CHAPTER 13

Investigating Mac OS X Systems

In Chapter 12, we noted that Windows examinations could be challenging. Part server, part workstation—performing a live response or examination of a Mac OS X system is no less of a complex endeavor. Both environments deserve their own standalone book on forensic examinations. The objective of this chapter, however, is to discuss the fundamental sources of evidence, as applied to common incident response investigations. Since the last edition of this book in the early 2000s, Apple systems have become far more prevalent in organizations, moving beyond the marketing and art departments. With that expansion, forensic examination tools have matured significantly.

This chapter is divided into subsections that focus on specific sources of evidence, rather than providing a complete guide to the potential data sources in Mac OS X. We cover a bit of fundamentals in each section. The topics include the following sources of evidence:

- The HFS+ file system
- Core operating system data
- Spotlight data
- System and application logging
- Application and system configuration

As in Chapter 12, we provide information on the data's role in supporting operating system functionality, what you need to collect as part of your forensic acquisition process, and how you can analyze or interpret the evidence. At the end of the chapter, we include a review that summarizes all the Mac OS X forensic artifacts we presented. In several sections we present different methods to obtain data during a live response, as opposed to what one would obtain through a forensic examination.

We'll start with the basics of the file system used by Mac OS X.

HFS+ AND FILE SYSTEM ANALYSIS

The Hierarchical File System (HFS+) has its roots in the HFS file system introduced in 1985. Drives were comparatively tiny at the time, and, naturally, the standard was updated to allow for larger, faster drives as well as to include additional features. HFS+, initially released in 1998 and whose development continues to this day, supports functionality expected of any modern file system, including the following features:

- Journaling
- Hard links
- Symbolic links
- Encryption
- ~8EB file size
- ~8EB volume size
- Resizable volumes
- Attribute structures

Notably absent from the list is native file system support for sparse files. These are common on most moderns file systems and allow a system to efficiently store files that contain empty space. The Mac OS X implementation of sparse files is handled in software: the Virtual File System driver layer of the operating system.

We start this section with a quick introduction to the layout of the file system and continue on to artifacts that may help you during analysis. As we go along, we show how to gain access to the structures using low-level file system utilities. We've found that current releases of common forensic examination suites are slightly unpredictable in the manner in which they interpret the HFS+ file system. When relegated to the task of simple file system browsing, the suites perform well, so whenever you need to get into the details, always reach for more specialized tools.

Volume Layout

An HFS+ volume consists of nine structures: boot blocks, a volume header, the allocation file, the extents overflow file, the catalog file, the attributes file, the startup file, the alternate volume header, and reserved blocks. The size of the allocation blocks is determined at the time the file system is created. What you would see if you examined a freshly formatted disk is a layout that matches Figure 13-1.

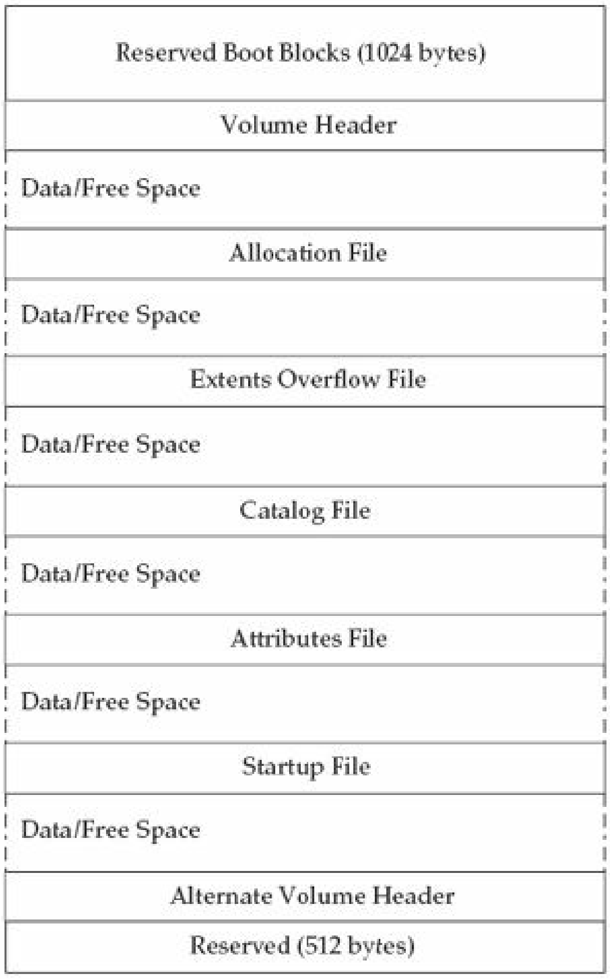


Figure 13-1. Anatomy of a disk

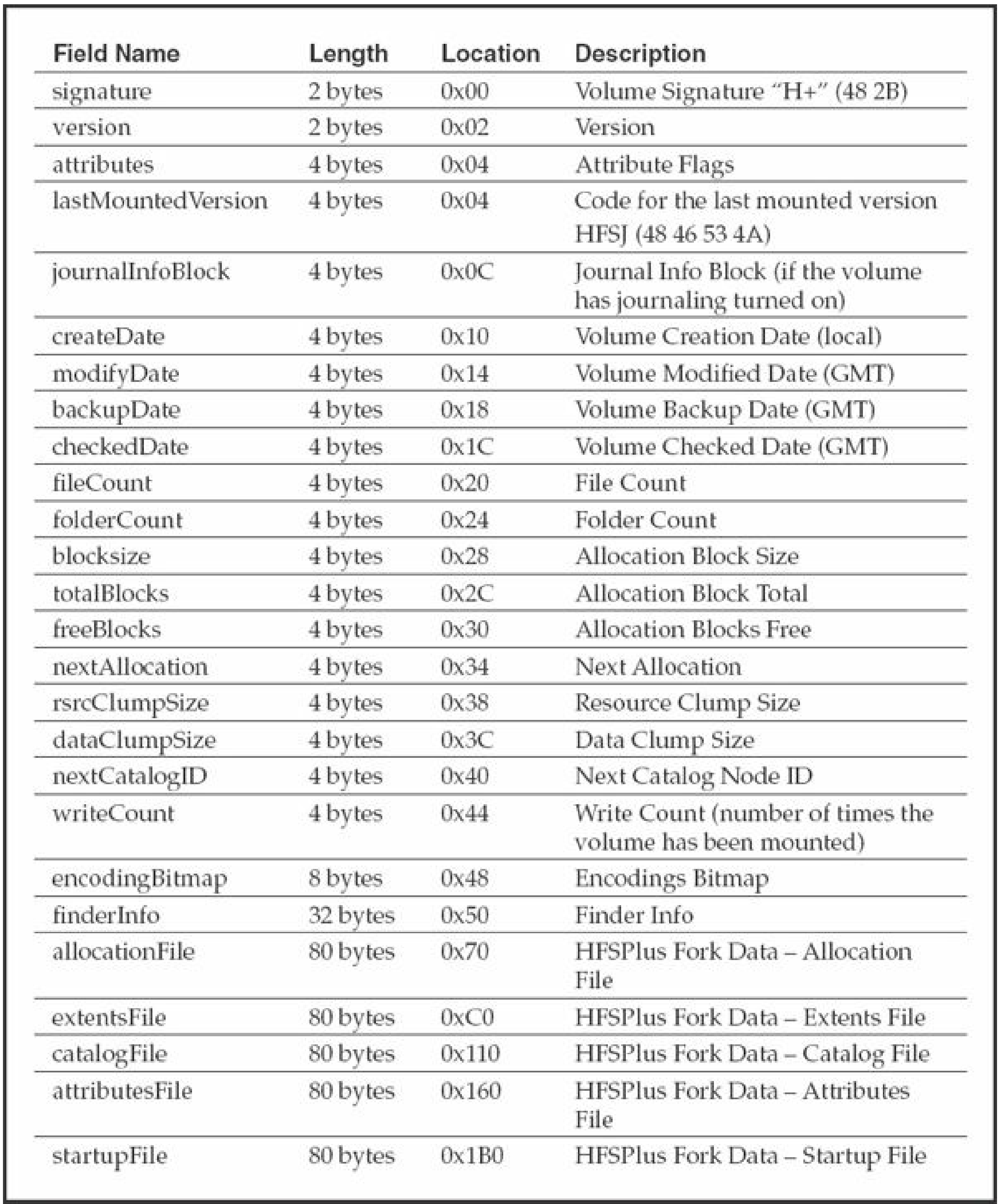
This layout is significantly different from FAT or NTFS, so we'll walk through a quick introduction. More detailed information can be found in Apple Technical Note 1150: HFS Plus Volume Format.

Boot Blocks

The first 1,024 bytes of the volume are reserved for use as boot blocks and may contain information required at startup. Boot blocks are not required by modern operating systems; therefore, this space is typically empty. However, the Mac OS Finder may write to this space when the System Folder changes.

Volume Header and Alternate Volume Header

The volume header is always located 1,024 bytes (two 512-byte allocation blocks) from the beginning the volume. The volume header contains information about the volume, including the location of the other structures. Table 13-1 lists the contents of the volume header.



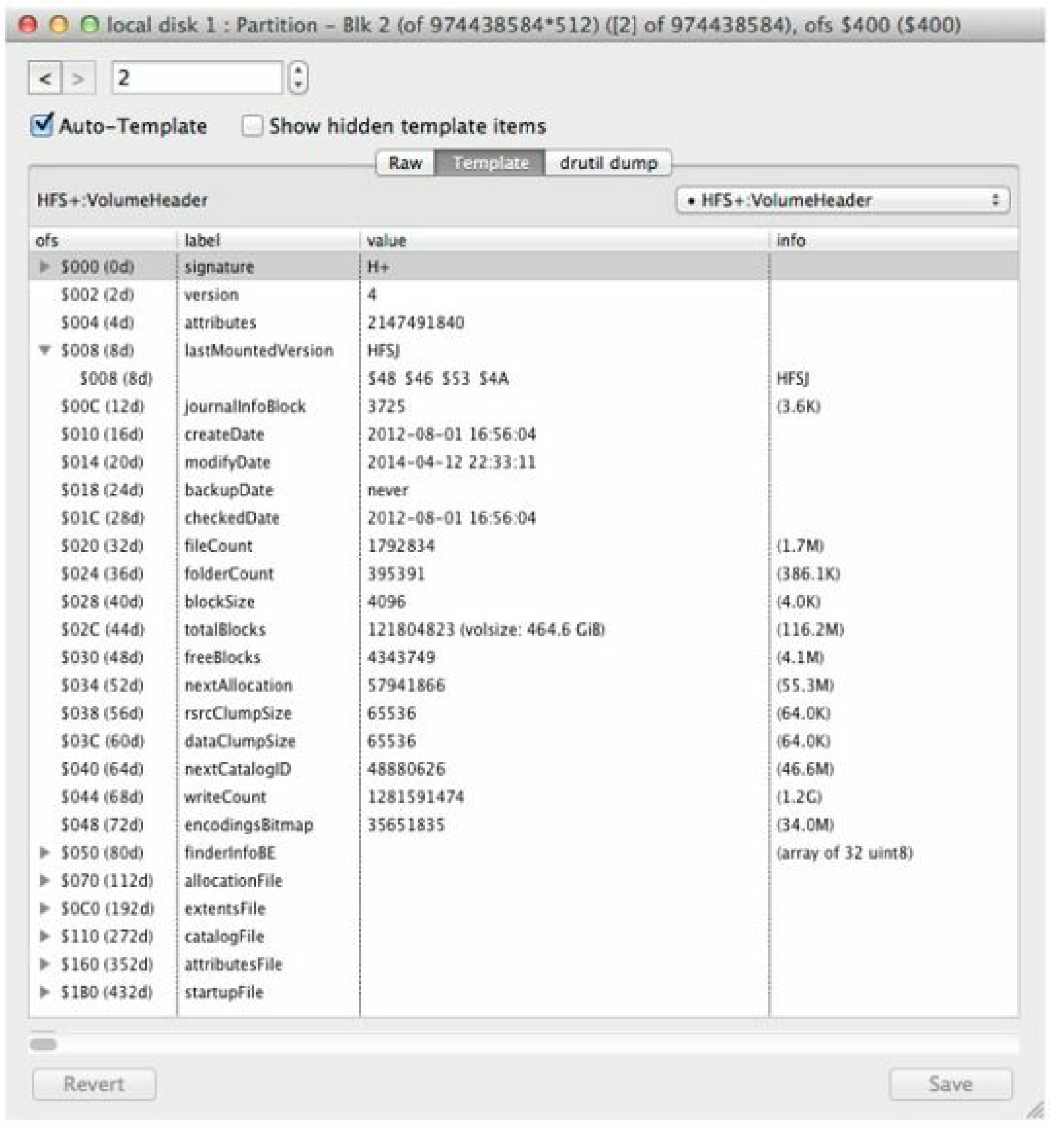
Field Name	Length	Location	Description
signature	2 bytes	0x00	Volume Signature "H+" (48 2B)
version	2 bytes	0x02	Version
attributes	4 bytes	0x04	Attribute Flags
lastMountedVersion	4 bytes	0x04	Code for the last mounted version HFSJ (48 46 53 4A)
journalInfoBlock	4 bytes	0x0C	Journal Info Block (if the volume has journaling turned on)
createDate	4 bytes	0x10	Volume Creation Date (local)
modifyDate	4 bytes	0x14	Volume Modified Date (GMT)
backupDate	4 bytes	0x18	Volume Backup Date (GMT)
checkedDate	4 bytes	0x1C	Volume Checked Date (GMT)
fileCount	4 bytes	0x20	File Count
folderCount	4 bytes	0x24	Folder Count
blocksize	4 bytes	0x28	Allocation Block Size
totalBlocks	4 bytes	0x2C	Allocation Block Total
freeBlocks	4 bytes	0x30	Allocation Blocks Free
nextAllocation	4 bytes	0x34	Next Allocation
rsrcClumpSize	4 bytes	0x38	Resource Clump Size
dataClumpSize	4 bytes	0x3C	Data Clump Size
nextCatalogID	4 bytes	0x40	Next Catalog Node ID
writeCount	4 bytes	0x44	Write Count (number of times the volume has been mounted)
encodingBitmap	8 bytes	0x48	Encodings Bitmap
finderInfo	32 bytes	0x50	Finder Info
allocationFile	80 bytes	0x70	HFSPlus Fork Data – Allocation File
extentsFile	80 bytes	0xC0	HFSPlus Fork Data – Extents File
catalogFile	80 bytes	0x110	HFSPlus Fork Data – Catalog File
attributesFile	80 bytes	0x160	HFSPlus Fork Data – Attributes File
startupFile	80 bytes	0x1B0	HFSPlus Fork Data – Startup File

Table 13-1. Volume Header Structure

Note the last five entries in the volume header. These contain a location and size for the five special files shown in Figure 13-1. Using iBored, a cross-platform data viewer that can interpret and represent structures based on templates, we can inspect these structures easily.

Figure 13-2 shows the contents of the second block that contains the volume header.

This disk was created on August 1, 2012 and was last modified on April 12, 2014.



local disk 1 : Partition - Blk 2 (of 974438584*512) ([2] of 974438584), ofs $400 ($400)

2

[✓ Auto-Template] Show hidden template items

Raw | Template | drutil dump

HFS+:VolumeHeader

• HFS+:VolumeHeader

ofs	label	value	info
▸ $000 (0d)	signature	H+	
$002 (2d)	version	4	
$004 (4d)	attributes	2147491840	
▾ $008 (8d)	lastMountedVersion	HFSJ	
$008 (8d)		$48 $46 $53 $4A	HFSJ
$00C (12d)	journalInfoBlock	3725	(3.6K)
$010 (16d)	createDate	2012-08-01 16:56:04	
$014 (20d)	modifyDate	2014-04-12 22:33:11	
$018 (24d)	backupDate	never	
$01C (28d)	checkedDate	2012-08-01 16:56:04	
$020 (32d)	fileCount	1792834	(1.7M)
$024 (36d)	folderCount	395391	(386.1K)
$028 (40d)	blockSize	4096	(4.0K)
$02C (44d)	totalBlocks	121804823 (volsize: 464.6 GiB)	(116.2M)
$030 (48d)	freeBlocks	4343749	(4.1M)
$034 (52d)	nextAllocation	57941866	(55.3M)
$038 (56d)	rsrcClumpSize	65536	(64.0K)
$03C (60d)	dataClumpSize	65536	(64.0K)
$040 (64d)	nextCatalogID	48880626	(46.6M)
$044 (68d)	writeCount	1281591474	(1.2G)
$048 (72d)	encodingsBitmap	35651835	(34.0M)
▸ $050 (80d)	finderInfoBE		(array of 32 uint8)
▸ $070 (112d)	allocationFile		
▸ $0C0 (192d)	extentsFile		
▸ $110 (272d)	catalogFile		
▸ $160 (352d)	attributesFile		
▸ $1B0 (432d)	startupFile		

Revert | Save

Figure 13-2. HFS+ volume header

HFS+ dates are stored in seconds since midnight January 1, 1904 GMT. An important thing to note is the volume creation date stored in the volume header is local time, not GMT. An HFS+ volume may have four dates: create date, modify date, backup date, and checked date.

On the topic of date stamps, HFS+ stores four dates for each file. It stores the normal file access, file modify, and inode change times. In addition, an inode birth time is recorded, indicating the file's creation date. During a live response, you can access these timestamps with the stat command. Note that in some situations, you will need to run commands with elevated privileges to gain access to the data.

GO GET IT ON THE WEB

iBored apps.tempel.org/iBored

The alternate volume header is always located 1,024 bytes from the end of the volume and occupies 512 bytes. The alternate volume header is a copy of the volume header that allows for recovery in the event the volume header becomes corrupted.

Allocation File

The allocation file is just what it sounds like—a record of available allocation blocks. It contains a bit for every block; when a bit is not set, it indicates the space is available for use.

Extents Overflow File

Each file in an HFS+ system has an associated list of extents or contiguous allocation blocks that belong to the file forks. Each extent is defined by a pair of numbers: the first is the allocation block of the extent, and the second is the number of allocation blocks assigned to the extent. The catalog B-tree maintains a record of the first eight extents. If there are more than eight extents in a fork, the remaining ones are stored in the extents overflow file. Forks are explained more fully in the "Attributes File" section.

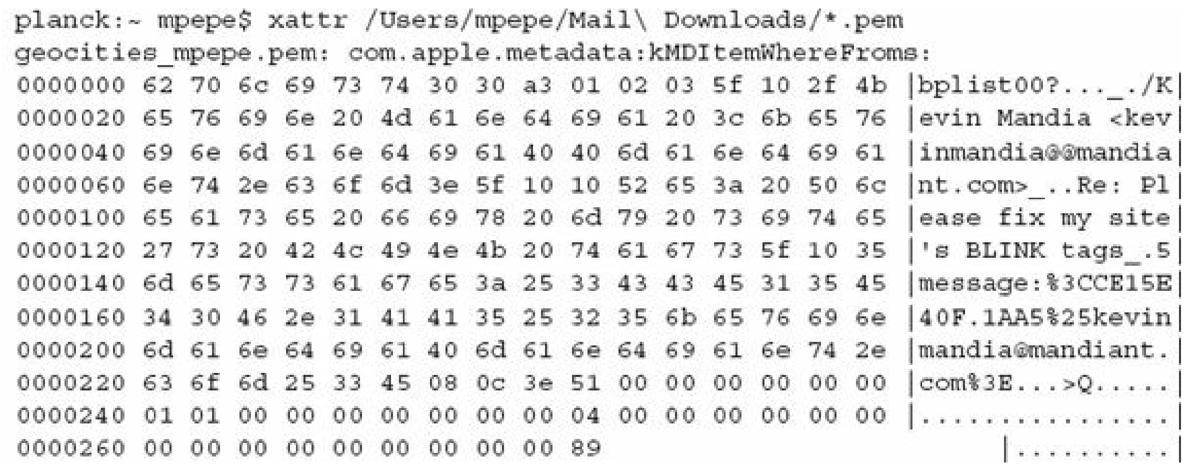
Catalog File

The catalog file is a file detailing the hierarchy of files and folders in the volume. Each file and folder in the volume has a unique catalog node ID (CNID). For folders, the CNID is called the Folder or Directory ID; for files, it is the File ID. For each folder and file there is a parent ID that is the CNID of the folder containing the folder or file. The first 16 CNIDs are reserved for use by Apple for important files in the volume.

Attributes File

The attributes file is an optional part of the file system standard. It is defined for use by *named forks*, which can be considered to be additional metadata assigned to a file that is not stored as a part of the file's entry itself. This functionality is what Microsoft was looking to achieve by introducing alternate data streams.

Extended information stored for a file may be examined with the xattr command. In the following example, the file /Users/mpepe/Mail Downloads/geocities_mpepe.pem was originally received by the user as an attachment to an e-mail sent by kmandia. This information was captured in metadata by the relatively recent feature in Mac OS X that stores the location from which the file was received in a binary plist format. If you happen to see a window come up the next time you open a file retrieved by Safari, this is where Finder gets that source information.

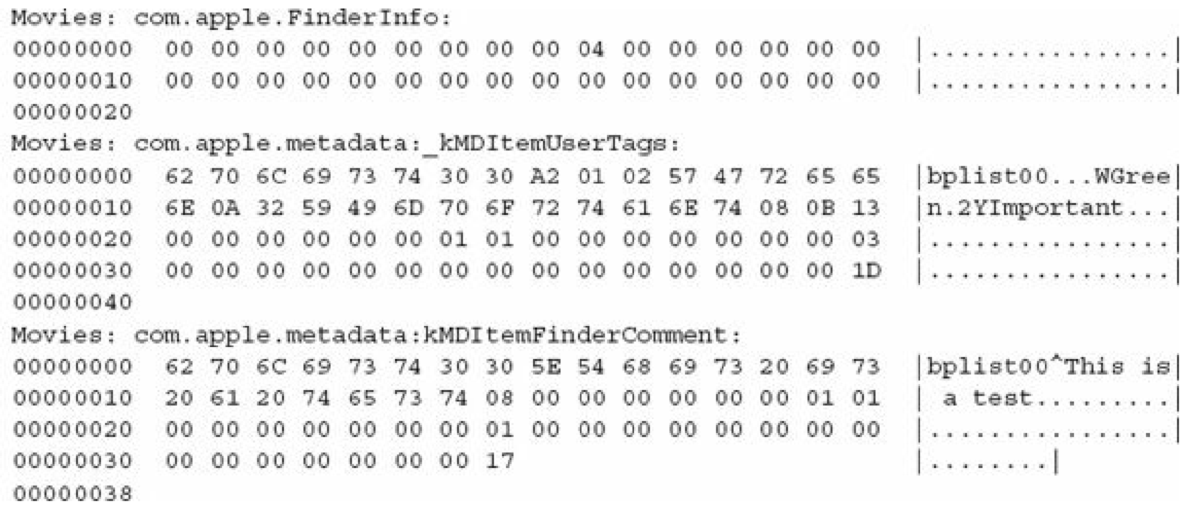


```
planck:~ mpepe$ xattr /Users/mpepe/Mail\ Downloads/*.pem
geocities_mpepe.pem: com.apple.metadata:kMDItemWhereFroms:
0000000 62 70 6c 69 73 74 30 30 a3 01 02 03 5f 10 2f 4b |bplist00?..._./K|
0000020 65 76 69 6e 20 4d 61 6e 64 69 61 20 3c 6b 65 76 |evin Mandia <kev|
0000040 69 6e 6d 61 6e 64 69 61 40 40 6d 61 6e 64 69 61 |inmandia@@mandia|
0000060 6e 74 2e 63 6f 6d 3e 5f 10 10 52 65 3a 20 50 6c |nt.com>_..Re: Pl|
0000100 65 61 73 65 20 66 69 78 20 6d 79 20 73 69 74 65 |ease fix my site|
0000120 27 73 20 42 4c 49 4e 4b 20 74 61 67 73 5f 10 35 |'s BLINK tags_.5|
0000140 6d 65 73 73 61 67 65 3a 25 33 43 43 45 31 35 45 |message:%3CCE15E|
0000160 34 30 46 2e 31 41 41 35 25 32 35 6b 65 76 69 6e |40F.1AA5%25kevin|
0000200 6d 61 6e 64 69 61 40 6d 61 6e 64 69 61 6e 74 2e |mandia@mandiant.|
0000220 63 6f 6d 25 33 45 08 0c 3e 51 00 00 00 00 00 00 |com%3E...>Q.....|
0000240 01 01 00 00 00 00 00 00 00 04 00 00 00 00 00 00 |................|
0000260 00 00 00 00 00 00 00 00 00 89                             |..........|
```

A small number of Mac OS X services will place data in named forks. To get a sense of what to expect from a normal file system image, run `xattr -lr` * on a freshly installed system. This will show all extended attributes and their creator. Most key/value pairs you will see are generated by com.apple.metadata and include kMDItemDownloadedDate, kMDItemWhereFroms, kMDItemFinderComment, and kMDItemUserTags. The owner com.apple.metadata maintains a number of predefined metadata objects, but developers are free to generate their own. Note that the creator is a classification rather than a particular process or application. In these examples, com.apple.metadata is data originating from Safari as well as Finder. If you were to change the tags on a file in Finder, such as turning the item green or tagging it as "Important" through the file's Info pane, those tags are stored in the named forks described earlier.

Note For additional information on the predefined metadata objects, perform a search on http://developer.apple.com for the document titled "MDItem Reference."

Here is an example of the Movies directory after the tags "Green" and "Important" were set, as well as a comment, through the file's Info pane:



```
Movies: com.apple.FinderInfo:
00000000  00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
00000020
Movies: com.apple.metadata:_kMDItemUserTags:
00000000  62 70 6C 69 73 74 30 30 A2 01 02 57 47 72 65 65  |bplist00...WGree|
00000010  6E 0A 32 59 49 6D 70 6F 72 74 61 6E 74 08 0B 13  |n.2YImportant...|
00000020  00 00 00 00 00 00 01 01 00 00 00 00 00 00 00 03  |................|
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 1D  |................|
00000040
Movies: com.apple.metadata:kMDItemFinderComment:
00000000  62 70 6C 69 73 74 30 30 5E 54 68 69 73 20 69 73  |bplist00^This is|
00000010  20 61 20 74 65 73 74 08 00 00 00 00 00 00 01 01  | a test.........|
00000020  00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 17                          |........|
00000038
```

Startup File
The startup file is intended to hold information needed when booting a system that does not have built-in (ROM) support for HFS+. Mac OS X does not use the startup file. In most cases the startup file is zero bytes.

File System Services
Apple developed and adopted a few technologies that support advanced functionality. The two main features that potentially harbor relevant information are Spotlight (the metadata indexer) and Managed Storage (the process responsible for revision control). Forensic tools to review the data from these sources are limited; however, we expect that will change over time.

Spotlight
Spotlight is a metadata indexing and searching service in Mac OS X. When new files are created, the Metadata framework ingests the file and indexes based on the plugins available at the time. A stock Mac OS X 10.9 install has approximately 23 importers that can parse various file formats, from Application Bundles to Mail. As additional software is installed, other importers can be loaded, thus expanding the Metadata framework's ability to recognize other formats. Certain types of importers only import metadata. Others will index all text in the incoming file. An example of the latter is the mail.mdimporter import plugin. On a running Mac OS X system, execute mdfind with a search string. If the string is in a file whose complete contents have been indexed, the mdfind application will return the full path of the matching file. Note that indexes for nonlocal file systems, such as removable media and network shares, are stored with the data itself. If the volume is not connected, an mdfind search will not return data from the disconnected source.

Naturally, the question for us is whether we can take advantage of any artifacts that the Metadata framework stores or leaves behind. Cleanup of stale data from deleted files is fast. In our testing, nodes in the index that corresponded to a deleted file were removed immediately. Unfortunately, no tools are currently available that can reliably parse the data stored by the Spotlight indexer once it's extracted from a drive image. Currently, the data maintained by Spotlight is useful only in a live response context.

Managed Storage

In Mac OS X Lion (10.7), Apple provided a new framework for developers that allows applications created for Mac OS X Lion and beyond to adopt a continuous save model for documents and files, as opposed to the traditional user-requested save operation. A side effect is that it leaves a good amount of data behind on the file system as files are modified. The daemon that manages this function is called revisiond, and it maintains data on volumes under the "hidden" directory /.DocumentRevisions-V100.

Under that directory, the service stores the file data in PerUID and the database in db-V1. To review the files stored in the versioning system, open the SQLite database/.DocumentRevisions-V100/db-V1/db.sqlite. The database's four tables track the file, the revisions retained for each, and storage locations. Note that if you are extracting the files for review, you will need to also copy the file ending in "-wal", which is a write-ahead log file that contains pertinent data.

As an example, let's examine the contents of an active db.sqlite file. The database is shown in Figure 13-3. We'll have more information on reviewing SQLite databases later in the chapter. To gain access to this database, we started a root shell via `sudo bash` and copied the files out to a temporary directory (~/book_tmp) to prevent any race conditions caused by having the database opened by multiple processes.

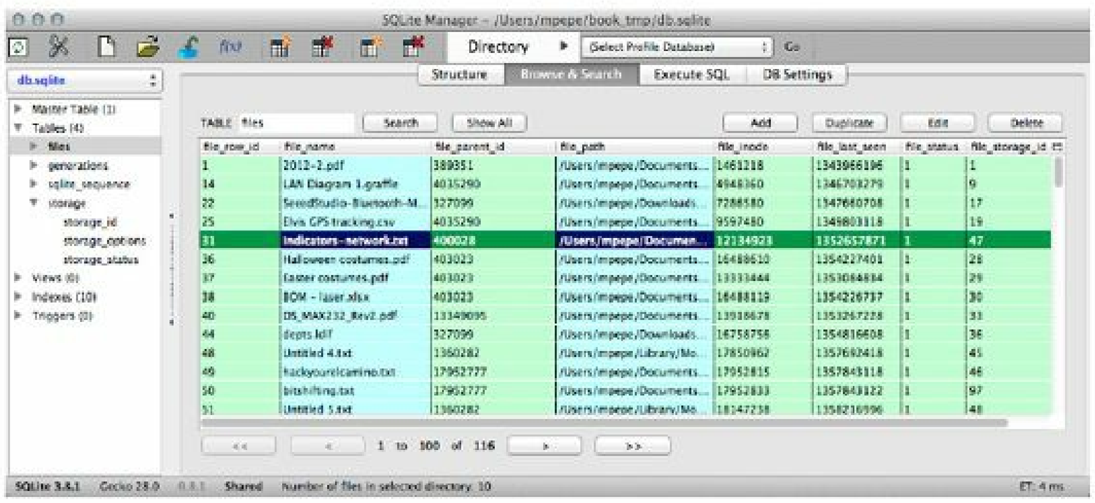


TABLE files

file_row_id	file_name	file_parent_id	file_path	file_inode	file_last_seen	file_status	file_storage_id
1	2012-2.pdf	389351	/Users/mpepe/Documents...	1461218	1343966196	1	1
14	LAN Diagram 1.graffle	4035290	/Users/mpepe/Documents...	4946360	1346703279	1	9
22	SeeedStudio-Bluetooth-M...	327099	/Users/mpepe/Downloads...	7288580	1347660708	1	17
25	Elvis GPS tracking.csv	4035290	/Users/mpepe/Documents...	9597480	1349803318	1	19
31	Indicators-network.txt	400028	/Users/mpepe/Documen...	12134923	1352657871	1	47
36	Halloween costumes.pdf	403023	/Users/mpepe/Documents...	16488610	1354227401	1	28
37	Easter costumes.pdf	403023	/Users/mpepe/Documents...	13333444	1353084834	1	29
38	BOM - laser.xlsx	403023	/Users/mpepe/Documents...	16488119	1354226737	1	30
40	DS_MAX232_Rev2.pdf	13349095	/Users/mpepe/Documents...	13938678	1353267228	1	33
44	depts.ldif	327099	/Users/mpepe/Downloads...	16758758	1354816608	1	36
48	Untitled 4.txt	1360282	/Users/mpepe/Library/Mo...	17850962	1357692418	1	45
49	hackyourelcamino.txt	17952777	/Users/mpepe/Documents...	17952815	1357843118	1	46
50	bitshifting.txt	17952777	/Users/mpepe/Documents...	17952833	1357843122	1	97
51	Untitled 5.txt	1360282	/Users/mpepe/Library/Mo...	18147238	1358216996	1	48

Number of files in selected directory: 10

Figure 13-3. Document Revision database in SQLite Manager

GO GET IT ON THE WEB

We use a Firefox plugin called SQLite Manager to quickly review SQLite databases. Search for this application in the Add-Ons menu.

The selected file, Indicators-network.txt, has a file_last_seen date of 1352657871 (or Sunday, 11 Nov 2012 18:17:51 GMT) and a file_storage_id of 47. We use that storage ID to reference the records in the "generations" table. Figure 13-4 shows the 14 revisions made to the file. The first entry shows a timestamp equal to the original creation date. The second shows that it was then edited on Wed, 21 Nov 2012 19:31:12 GMT.

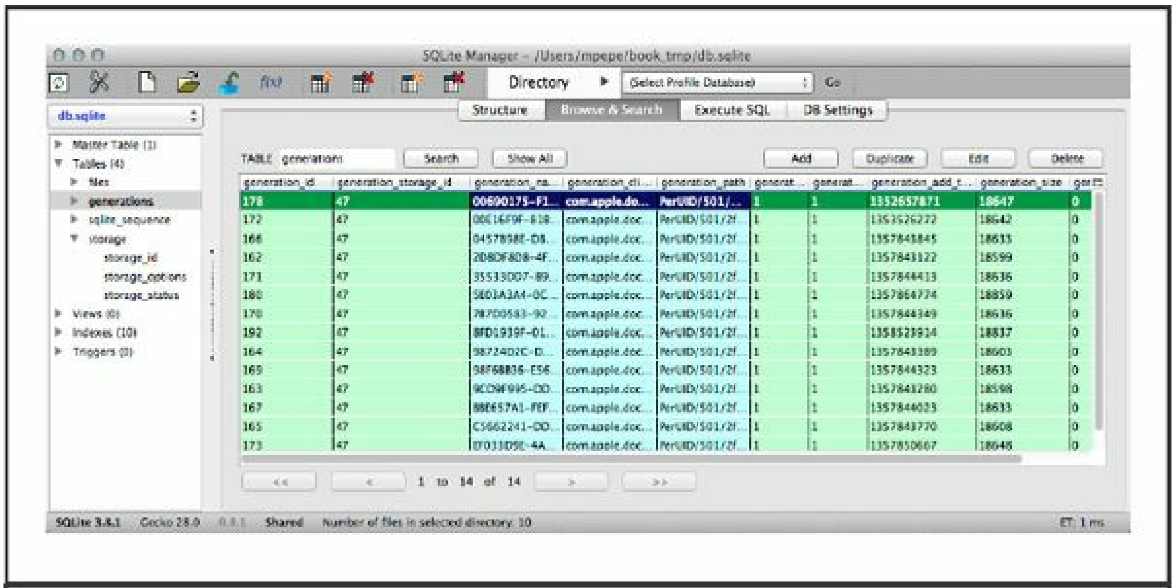


TABLE generations

generation_id	generation_storage_id	generation_na...	generation_cl...	generation_path	generat...	generat...	generation_add_t...	generation_size	gen
178	47	00690175-F1...	com.apple.do...	PerUID/501/...	1	1	1352657871	18647	0
172	47	00E16F9F-818...	com.apple.doc...	PerUID/501/2f...	1	1	1353526272	18642	0
166	47	0457B58E-D8...	com.apple.doc...	PerUID/501/2f...	1	1	1357843845	18633	0
162	47	2D8DF8D8-4F...	com.apple.doc...	PerUID/501/2f...	1	1	1357843122	18599	0
171	47	35533DD7-89...	com.apple.doc...	PerUID/501/2f...	1	1	1357844413	18636	0
180	47	5E03A3A4-0C...	com.apple.doc...	PerUID/501/2f...	1	1	1357864774	18859	0
170	47	78700583-92...	com.apple.doc...	PerUID/501/2f...	1	1	1357844349	18636	0
192	47	8FD1939F-01...	com.apple.doc...	PerUID/501/2f...	1	1	1358523914	18837	0
164	47	98724D2C-D...	com.apple.doc...	PerUID/501/2f...	1	1	1357843389	18603	0
169	47	98F68B36-E56...	com.apple.doc...	PerUID/501/2f...	1	1	1357844323	18633	0
163	47	9CD9F995-DD...	com.apple.doc...	PerUID/501/2f...	1	1	1357843280	18598	0
167	47	8BE657A1-FEF...	com.apple.doc...	PerUID/501/2f...	1	1	1357844023	18633	0
165	47	C5662241-DD...	com.apple.doc...	PerUID/501/2f...	1	1	1357843770	18608	0
173	47	EF031D5E-4A...	com.apple.doc...	PerUID/501/2f...	1	1	1357850667	18646	0

1 to 14 of 14

SQLite 3.8.1 Gecko 28.0 0.8.1 Shared Number of files in selected directory: 10 ET: 1 ms

Figure 13-4. Document revisions for file Indicators-network.txt

---

Note Recall that Mac OS X is derived from BSD, so when you come across a timestamp, it will be the number of seconds since the Unix epoch (00:00:00 UTC, January 1, 1970). You can use the Unix command `date` with options `-j -u -r (seconds)` to convert from seconds. Keep in mind that numerous online date calculators are available if you prefer to use one.

If we drill down into the record itself, the file name of the stored revision can be examined, as shown in the generation_path field in Figure 13-5.

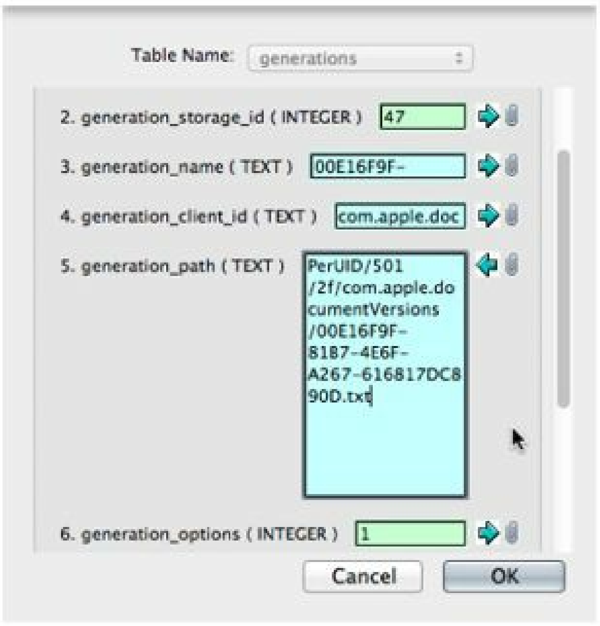


Table Name: generations

2. generation_storage_id (INTEGER) 47

3. generation_name (TEXT) 00E16F9F-

4. generation_client_id (TEXT) com.apple.doc

5. generation_path (TEXT) PerUID/501/2f/com.apple.documentVersions/00E16F9F-81B7-4E6F-A267-616817DC890D.txt

6. generation_options (INTEGER) 1

Figure 13-5. Revision 2 of Indicators-network.txt

We can then go into the file system and review 00E16F9F-81B7-4E6F-A267-616817DC890D.txt to see the state at that time. Because this is a simple text file, we can also use the diff command to compare revisions, if that is necessary for the investigation. As you may expect, the generation_add_date matches the HFS+ file record for the revision. This can be another data point if date manipulation is an issue in an investigation.

CORE OPERATING SYSTEM DATA

Most forensic analysts spend the vast majority of their time examining various versions of Windows operating systems. Making the transition to Mac OS X can be a bit disorienting at first. Where does one search for evidence on an HFS+ file system? Depending on the rights the user is operating under, the answer could be "anywhere." Let's first address common usage patterns and how the system is configured for most users.

File System Layout

Apple defines four "domains" for data classification. The four domains are local, system, network, and user. Each domain is defined by a set of paths, resources, and access controls. The local domain consists of applications and configurations that are shared among all users of the system. A user requires administrative privileges to modify the data in this domain. The following directories fall under the local domain:

- /Applications
- /Developer
- /Library

The second domain, system, contains data installed by Apple as well as a few specialized low-level utilities. Files in the following directories require administrative privileges to modify. Included in the system domain are all of the traditional Unix structures: /bin, /usr, /dev, /etc, and so on. This domain is typically the most useful during intrusion investigations due to the location of the system logs. The /System directory is considered to be in the system domain.

The third domain, network, is where applications and data is stored that will be shared among a network of systems and users. In practice, this domain is rarely populated with data. This is located under the /Network directory.

The fourth domain, user, is the source of data that will apply to most other investigations. The user domain contains user home directories and a shared directory. Generally, all user-created content and configurations will be found under the /Users top-level directory.

As we stated earlier, if a user has sufficient permissions, user-generated files and applications may break this model, particularly if the user is Unix savvy and installs additional console applications or the MacPorts package manager. Incidentally, we highly suggest that you install MacPorts on your Mac examination workstations. It provides a simple means to get a large number of packages from the BSD Ports tree on your system. Note that in order to get a working Ports tree in your environment, you'll need to install the command-line developer tools. The simplest means to launch this process is to run gcc on the command line. The system will identify that the tools are not present and will launch an installer. As you find tools and utilities that can help during an examination, check whether someone has created a package in MacPorts before compiling from source. If so, it usually saves a bit of time.

GO GET IT ON THE WEB

MacPorts www.macports.org

Note

Additional information on the Mac OS X file environment can be found on the Apple Mac Developer site. We've noticed that over the years, the information has moved between different portions of the developer guide. Search http://developer.apple.com for the document titled "File System Programming Guide" for OS X.

The following subsections detail the three primary domains—local, system, and user—and the nature of the data you'll find in each. We discuss the recovery and analysis of specific artifacts in the next section. Note that we are omitting the network domain because it rarely is present and is similar to the local domain.

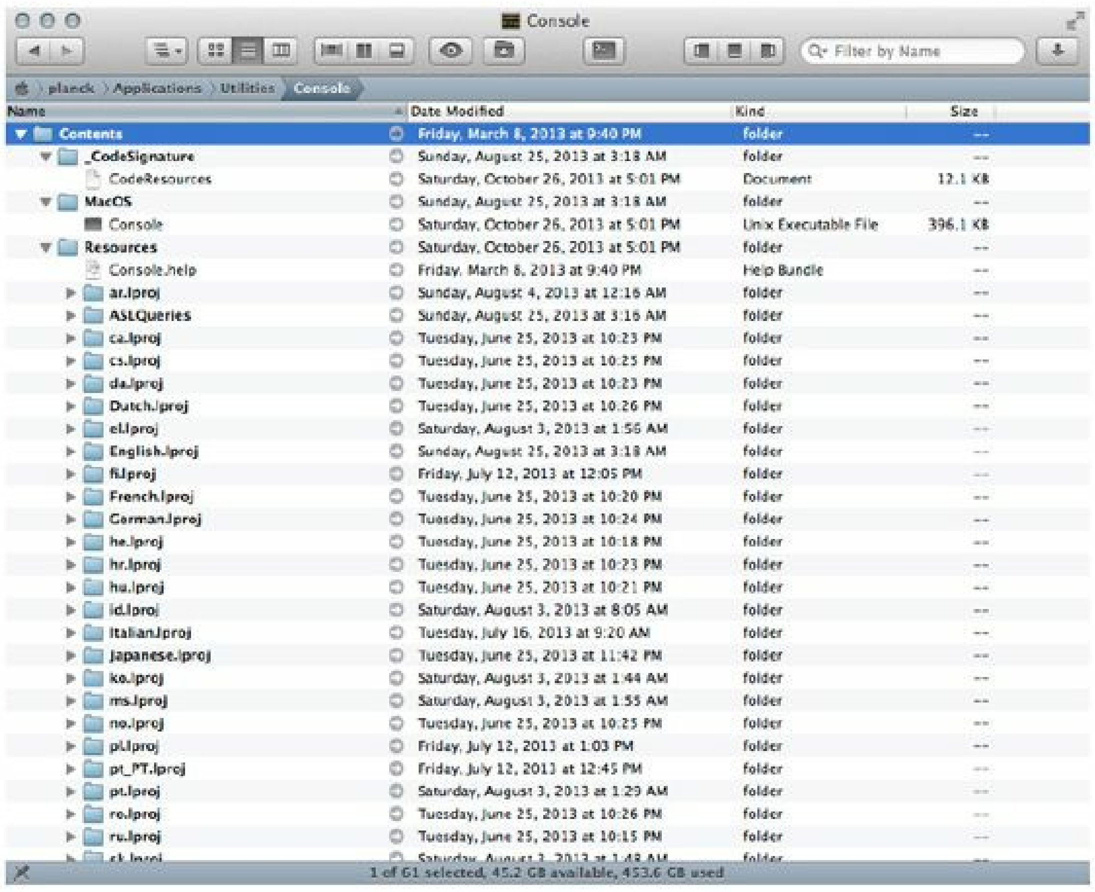
The Local Domain

In the local domain are three directories you should be familiar with. The first is /Applications. This is the directory where nearly every application is installed (by you or the App Store). We'll cover how to determine what software is installed and when it was installed shortly. Let's digress for a moment to talk about Application Bundles.

Application Bundles are directory structures in a standardized format and extension. They contain nearly everything an application requires in order to run, including executable code, graphical resources, configuration files, libraries, and helper applications and scripts. They provide an interesting means of distributing applications, because the Finder treats the structure as a single file with its interface, unless specifically requested otherwise. The most common Application Bundle extensions are

- **.app** Launchable applications
- **.framework** Dynamic shared libraries and their resources
- **.plugin** Helper applications or drivers for other applications
- **.kext** Dynamically loadable kernel modules

If you were to open an Application Bundle, either by selecting Show Package Contents from Finder or by traversing the directories on the terminal, you'd find several subdirectories under the Contents folder. These subdirectories—MacOS and, optionally, Resources, Library, Frameworks, PlugIns, and SharedSupport—contain the items listed previously. There are no limitations on what a developer can put in these directories, and they often contain useful tools and files. For example, within the Library folder in the VMWare Fusion bundle, you will find all of the command-line utilities that manage the VMWare hypervisor. An example of an Application Bundle is shown in Figure 13-6. The figure displays the contents of the Console application included with Mac OS X, accessible by Control-clicking the file and selecting Show Package Contents from the context menu.



Name	Date Modified	Kind	Size
Contents	Friday, March 8, 2013 at 9:40 PM	folder	--
_CodeSignature	Sunday, August 25, 2013 at 3:18 AM	folder	--
CodeResources	Saturday, October 26, 2013 at 5:01 PM	Document	12.1 KB
MacOS	Sunday, August 25, 2013 at 3:18 AM	folder	--
Console	Saturday, October 26, 2013 at 5:01 PM	Unix Executable File	396.1 KB
Resources	Saturday, October 26, 2013 at 5:01 PM	folder	--
Console.help	Friday, March 8, 2013 at 9:40 PM	Help Bundle	--
ar.lproj	Sunday, August 4, 2013 at 12:16 AM	folder	--
ASLQueries	Sunday, August 25, 2013 at 3:16 AM	folder	--
ca.lproj	Tuesday, June 25, 2013 at 10:23 PM	folder	--
cs.lproj	Tuesday, June 25, 2013 at 10:25 PM	folder	--
da.lproj	Tuesday, June 25, 2013 at 10:23 PM	folder	--
Dutch.lproj	Tuesday, June 25, 2013 at 10:26 PM	folder	--
el.lproj	Saturday, August 3, 2013 at 1:56 AM	folder	--
English.lproj	Sunday, August 25, 2013 at 3:18 AM	folder	--
fi.lproj	Friday, July 12, 2013 at 12:05 PM	folder	--
French.lproj	Tuesday, June 25, 2013 at 10:20 PM	folder	--
German.lproj	Tuesday, June 25, 2013 at 10:24 PM	folder	--
he.lproj	Tuesday, June 25, 2013 at 10:18 PM	folder	--
hr.lproj	Tuesday, June 25, 2013 at 10:23 PM	folder	--
hu.lproj	Tuesday, June 25, 2013 at 10:21 PM	folder	--
id.lproj	Saturday, August 3, 2013 at 8:05 AM	folder	--
Italian.lproj	Tuesday, July 16, 2013 at 9:20 AM	folder	--
Japanese.lproj	Tuesday, June 25, 2013 at 11:42 PM	folder	--
ko.lproj	Saturday, August 3, 2013 at 1:44 AM	folder	--
ms.lproj	Saturday, August 3, 2013 at 1:55 AM	folder	--
no.lproj	Tuesday, June 25, 2013 at 10:25 PM	folder	--
pl.lproj	Friday, July 12, 2013 at 1:03 PM	folder	--
pt_PT.lproj	Friday, July 12, 2013 at 12:45 PM	folder	--
pt.lproj	Saturday, August 3, 2013 at 1:29 AM	folder	--
ro.lproj	Tuesday, June 25, 2013 at 10:26 PM	folder	--
ru.lproj	Tuesday, June 25, 2013 at 10:15 PM	folder	--
sk.lproj	Saturday, August 3, 2013 at 1:48 AM	folder	--

1 of 61 selected, 45.2 GB available, 453.6 GB used

Figure 13-6. Contents of an AppBundle

You may have noticed in the figure that the file Console.help is a Help Bundle. This is, in fact, a bundle within the Console Application Bundle. We could view the package's contents as well and observe the same directory structure.

Why is all of this important? The package contents may contain additional metadata that could be relevant to your investigation. Time and date stamps may tell you when an application was installed. Additional "helper" programs may give you clues about the functionality of an application. It's also a great place to hide data.

Continuing on with the first domain, local, there are two more directories that Apple groups with Applications. Developer is a path used by XCode, Apple's development environment. Until recently, all the development tools, SDKs, documentation, and debugging tools were stored in this directory on the root of the drive. In later versions of XCode, the tools locations have changed; however, you may still see this directory from time to time.

The final directory we cover in the local domain is /Library. As you examine a file system, you'll notice that there are a few directories under /Library that appear in several locations. Generally these are used in a similar manner by applications, depending on scope. These contain application settings for the operating system (/System/Library), settings shared between users (/Library), and user-specific settings (/Users/*username*/Library). Keep this in mind as you determine how a particular application is configured to run. The subdirectories detailed in the following table often contain relevant information.

Path	**Description**
/Library/Application Support /User/*username*/Library/Application Support	This directory is used by applications to store settings, caches, license information, and nearly anything else desired by the developer. This includes applications developed by Apple, although there is a bit of inconsistency. As an example, in the system Library directory, most Apple utilities store global resources in Library/Apple/*applicationname*. CrashReporter, Apple Pro Applications (Final Cut, for example), and the App Store are exceptions.
/Library/Caches /System/Library/Caches /User/*username*/Library/Caches	This directory is used by applications to store temporary data. The Caches directory holds a significant amount of potentially relevant data. We cover a few applications that make use of this directory in Chapter 14.
/Library/Frameworks /System/Library/Frameworks	This directory is used by applications that need to store drivers or helper applications. Generally, unless there is a situation where malicious applications are suspected, or a user has installed special applications relevant to your investigation, this directory will not contain significant information.
/Library/Keychains	This directory stores the data files for the user's Keychain. Many applications place

/System/Library/Keychains /User/*username*/Library/Keychains	passwords and certificates in this service's data store. It does, however, require the user's password to open.
/Library/Logs /User/*username*/Library/Logs	This directory contains various application logs. This directory is one of the most important to review in this domain.
/Library/Preferences /User/*username*/Library/Preferences	This directory stores application preferences, if the application allows a system API to manage them. Generally, the data files are in a property list (plist) format, which makes examination fairly simple. This directory can be (very) loosely compared to the Software hive from a Windows system.
/Library/Receipts /User/*username*/Library/Receipts	When applications are added to the system, the files in /Library/Receipts are updated. A plist named InstallHistory.plist contains information about every application installed via the OS's installer or update framework.
/Library/WebServer	Apache, installed on every copy of Mac OS X, is started when a user turns on Web Sharing. Apache's Document Root directory is this folder in /Library.

There are numerous other subdirectories to note as well, especially when you begin to analyze a system for startup tasks. As we give more specific examples on what an investigator is interested in when performing an incident response, we'll refer back to these subdirectories.

Note Applications are free to use cache and Application Support directories as they see fit; however, there are a couple of very common file types that you'll find in use by nearly every application. Specifically, you will find property lists (plists) and SQLite databases to be quite prevalent. Although a multitude of options are available, we use a couple utilities frequently to review data in these two formats. We use Firefox Plugin SQLite Manager to review SQLite databases, and we use plutil on Mac OS X and plist Explorer on Windows for plist examination.

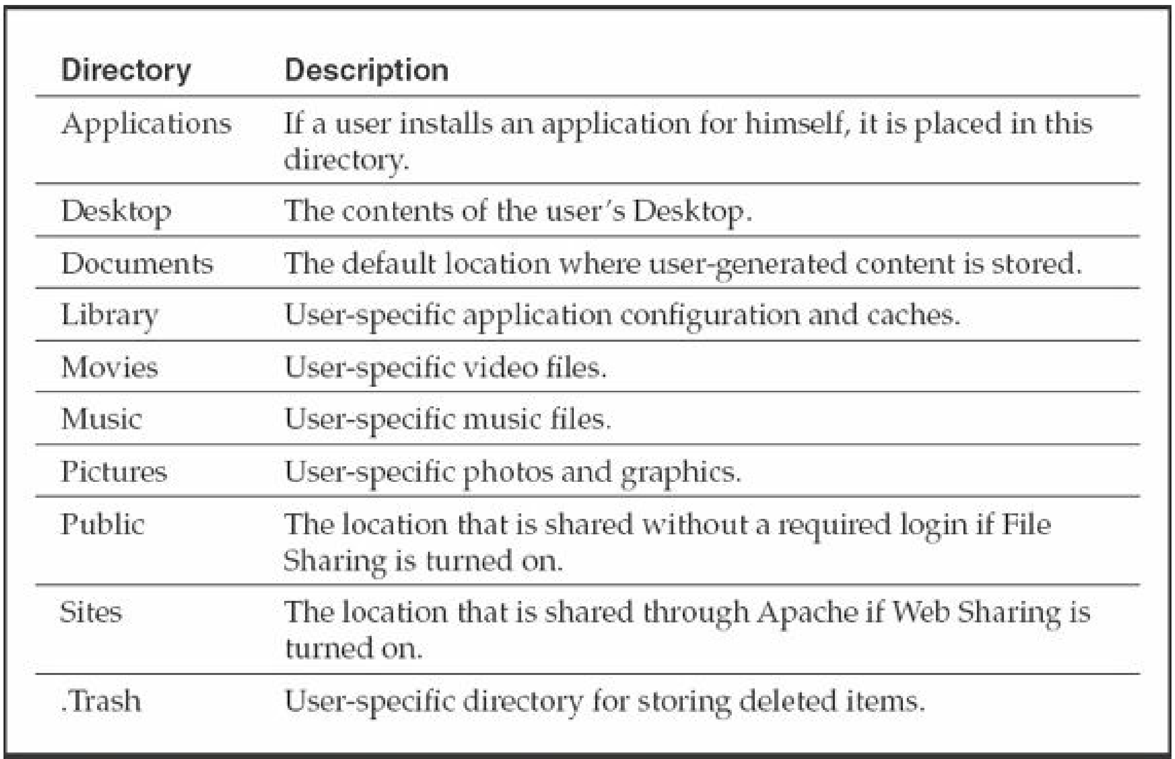
The System Domain

The system domain may appear at first to be a region of the drive that is less than exciting. In fact, because it includes all the traditional Unix paths as well as Mac OS X application logging, there are situations where an investigation may be based on findings solely from this domain. In the /System directory, you'll find a structure that resembles the /Library directory discussed earlier. Here, you will find many locations where applications can maintain persistence on a Mac OS X system. We'll cover these locations in a later section; however, it's good to note here that a user must have administrator-level privileges to create or modify files in the system domain.

Included in the system domain are many great sources of evidence. The traditional Unix application binary and library directories contain data that typically does not differ from one installation to the next, assuming matching Mac OS X versions. The exception to this is if the user is advanced and either compiles utilities manually or uses the MacPorts or Fink distributions, typically placed in /opt. Among numerous other artifacts, you can find system logs in /var/log, numerous databases in /var/db, records of printed data in the CUPS log directory, and the system sleep image.

The User Domain

The user domain is where most, if not all, user-created content resides. In the directory /Users, you'll find a directory for each individual account, as well as a shared directory. When a new user account is created, the user's home directory is populated with the directories listed in Table 13-2.



Directory	Description
Applications	If a user installs an application for himself, it is placed in this directory.
Desktop	The contents of the user's Desktop.
Documents	The default location where user-generated content is stored.
Library	User-specific application configuration and caches.
Movies	User-specific video files.
Music	User-specific music files.
Pictures	User-specific photos and graphics.
Public	The location that is shared without a required login if File Sharing is turned on.
Sites	The location that is shared through Apache if Web Sharing is turned on.
.Trash	User-specific directory for storing deleted items.

Table 13-2. Directories in the User Domain

From our experience, the majority of the time you spend examining the user domain will be in the Library and Documents directories.

In the next portion of this chapter, we explore a number of specific sources of evidence and present methods to perform analysis. Many of these sources are useful in several types of investigations, not only computer intrusions.

User and Service Configuration

Since Mac OS X was first released, the method used to store and track user accounts has matured. Before Mac OS X 10.5, the operating system had inherited user management from NeXTstep. This scheme, NetInfo, was an interesting combination of Sun's NIS+ user management and DNS. Unfortunately, many problems arose from how this daemon was designed, and over time Apple converted to LDAP for enterprise management and Directory Services for local user management. When examining a static drive, this may be the first thing you notice, if you typically gather a list of authorized users. Directory Services does not store user account information within traditional Unix files such as /etc/passwd and /etc/groups. Instead, the data for the local system is located in SQLite databases and binary-formatted property lists.

The Evidence

The Directory Service stores its data in /private/var/db/dslocal. Within this directory, the databases (or nodes) for the local system reside within nodes/Default. As you examine the files, you will notice directories and plist files that correspond to many Unix environment configuration options. In addition to user accounts and groups, you'll find that Directory Services manages several other configuration items:

- **aliases** Local routing for internal mail
- **computers** Kerberos information for the local system
- **config** Kerberos and Share information
- **network** Loopback network information
- **sharepoints** Directories that are shared out to other systems through SMB or AFP

Also resident in the /private/var/db/dslocal directory structure is a SQLite database named sqlindex. This database maintains the creation and modification time for the plist files in the directory structure as well as additional information on the relationships between the data.

A periodic CRON job backs up this entire directory. The backup file is located at /private/var/db/dslocal-backup.xar and is a normal gzip tar file.

Analysis

The Directory Services data will show what the service configuration was at the time of imaging and can be helpful in determining when certain configuration events occurred. During an IR, we may need to determine when a particular share was created or whether an account existed and its privilege level. Each data source within Directory Services may yield significant information.

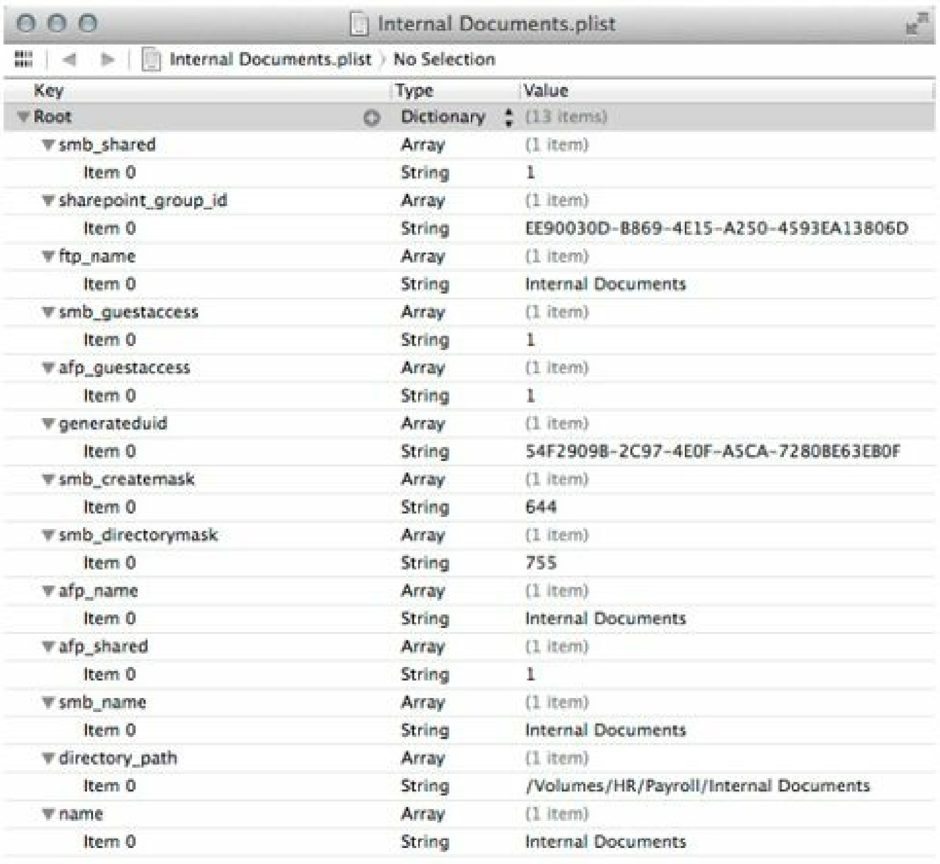
User Accounts In the "users" node, a binary plist file represents each user account. The plist file typically includes the properties listed in Table 13-3. Note that the order or presence of particular properties depends on each system's configuration.

jpegphoto	Naprivs	passwordpolicyoptions
picture	_writers_picture	hint
shell	_writers_realname	realname
name	_writers_UserCertificate	home
KerberosKeys	ShadowHashData	uid
_writers_passwd	LinkedIdentity	generateduid
gid	Passwd	_writers_hint
_writers_jpegphoto		

Table 13-3. User Properties

The most useful fields we examine are jpegphoto, picture, realname, name, home, generateduid, and uid. In addition, we can check the rec:users table in sqlindex. The filetime field can help you determine the original date of installation and the last time a user's record was modified. The photo-related fields are not very interesting in intrusion investigations; however, it's surprising how often photos taken with the built-in webcam are used as account images.

Sharepoints The sharepoints node contains a binary plist for each shared directory. When a user turns on File Sharing for a directory, the system creates a binary plist that contains 13 attributes, including the status of the share for AFP, SMB, and FTP, the sharepoint names for each service, and the shared path. An example is shown in Figure 13-7. This example shows the properties for a share named Internal Documents at the path /Volumes/HR/Payroll/Internal Documents.


Internal Documents.plist

Internal Documents.plist › No Selection

Key	Type	Value
Root	Dictionary	(13 items)
smb_shared	Array	(1 item)
Item 0	String	1
sharepoint_group_id	Array	(1 item)
Item 0	String	EE90030D-B869-4E15-A250-4593EA13806D
ftp_name	Array	(1 item)
Item 0	String	Internal Documents
smb_guestaccess	Array	(1 item)
Item 0	String	1
afp_guestaccess	Array	(1 item)
Item 0	String	1
generateduid	Array	(1 item)
Item 0	String	54F2909B-2C97-4E0F-A5CA-7280BE63EB0F
smb_createmask	Array	(1 item)
Item 0	String	644
smb_directorymask	Array	(1 item)
Item 0	String	755
afp_name	Array	(1 item)
Item 0	String	Internal Documents
afp_shared	Array	(1 item)
Item 0	String	1
smb_name	Array	(1 item)
Item 0	String	Internal Documents
directory_path	Array	(1 item)
Item 0	String	/Volumes/HR/Payroll/Internal Documents
name	Array	(1 item)
Item 0	String	Internal Documents

Figure 13-7. Sharepoint definition properties

As with the "users" node, you can determine when the sharepoint was created by examining the sqlindex database file. Figure 13-8 shows the entry when the sharepoint is created. In this case, the file time is 1382822698 (Saturday, 26 Oct 2013 21:24:58 GMT).

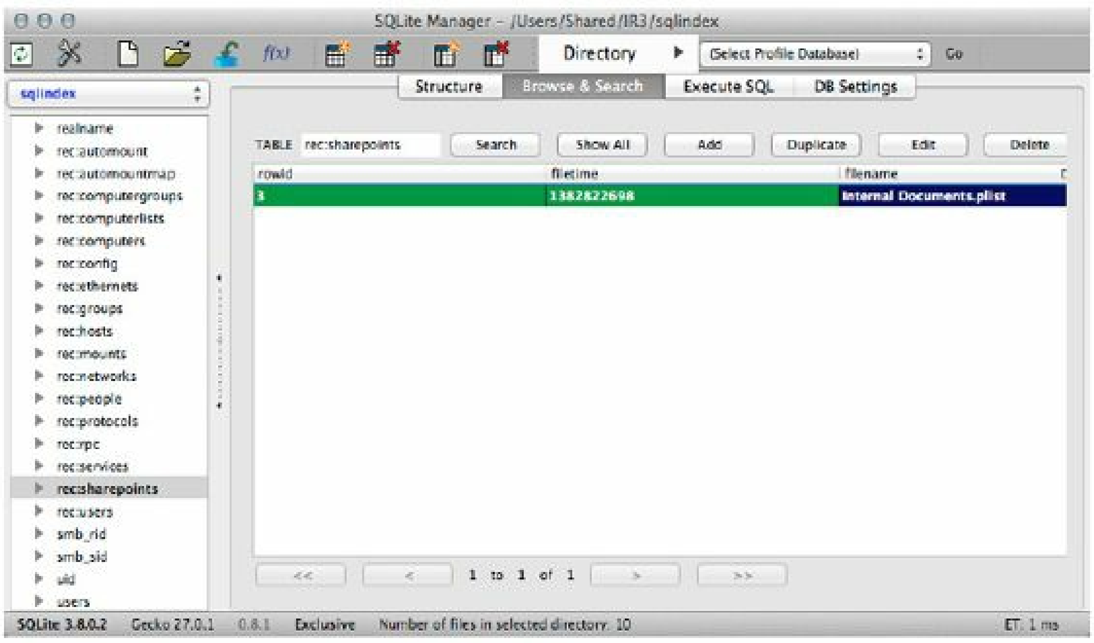


TABLE rec:sharepoints

rowid	filetime	filename
3	1382822698	Internal Documents.plist

SQLite 3.8.0.2 Gecko 27.0.1 0.8.1 Exclusive Number of files in selected directory: 10 ET: 1 ms

Figure 13-8. Sharepoint creation time

Trash and Deleted Files

Like Linux and Windows, files deleted via the graphical user interface are retained temporarily before permanent deletion. Mac OS X stores files marked for deletion in three different locations, depending on the location of the original file and the user that performs the deletion:

- **/.Trashes** Volume-wide trash folder
- **~/.Trash** User-specific trash folder
- **/private/var/root/.Trash** Root user's trash folder

The Evidence

On removable volumes, such as USB drives, a volume-wide trash folder will be created (for example, /Volumes/USBDRIVE/.Trashes) when the volume is mounted. Any time a user deletes a file from the volume, Mac OS X will create a directory for that user ID within the volume-specific trash folder (for example, /Volumes/USBDRIVE/.Trashes/501 for uid 501). A copy of each file deleted by that user ID is stored in that trash folder.

System Auditing, Databases, and Logging

Since its early days, one of Apple's primary design principles has been to ensure that the environment and ecosystem for its products maintain a consistently good user experience. For better or worse, this principle drives software design decisions that can benefit us, as investigators. Many daemons that help ensure that things "just work" for the end user maintain detailed logs and databases. There are many great sources of forensic artifacts, enough to warrant an entire book on the topic. Many of these data sources are more helpful in incidents where user activities can contribute significant findings to your investigation than in computer intrusion incidents. Because of these two points, this section is an introduction to the amount and nature of data you can examine.

System Auditing and Databases

Mac OS X has a powerful auditing system known as the Open Source Basic Security Module (OpenBSM). This system can log file access, inbound and outbound network connections, and the execution of applications and their command-line options. Unfortunately, the default configuration does not retain detailed information and is of limited use in an IR. To view the non-ASCII OpenBSM log files, use the praudit command. OpenBSM is a cross-platform suite, so you can export data from a forensic image of an Mac OS X drive and use praudit on Linux, if you are more comfortable in that environment. Another option is to use a native Mac OS X tool called Audit Explorer. Available in the App Store, Audit Explorer will process the OpenBSM output files, allowing for easy examination.

The configuration files for OpenBSM are in /etc/security. The primary file, audit_control, specifies that the log data is stored in /private/var/audit. During a live response, you will want to retain the contents of this entire directory. Each log file is named for the time period for which it contains events.

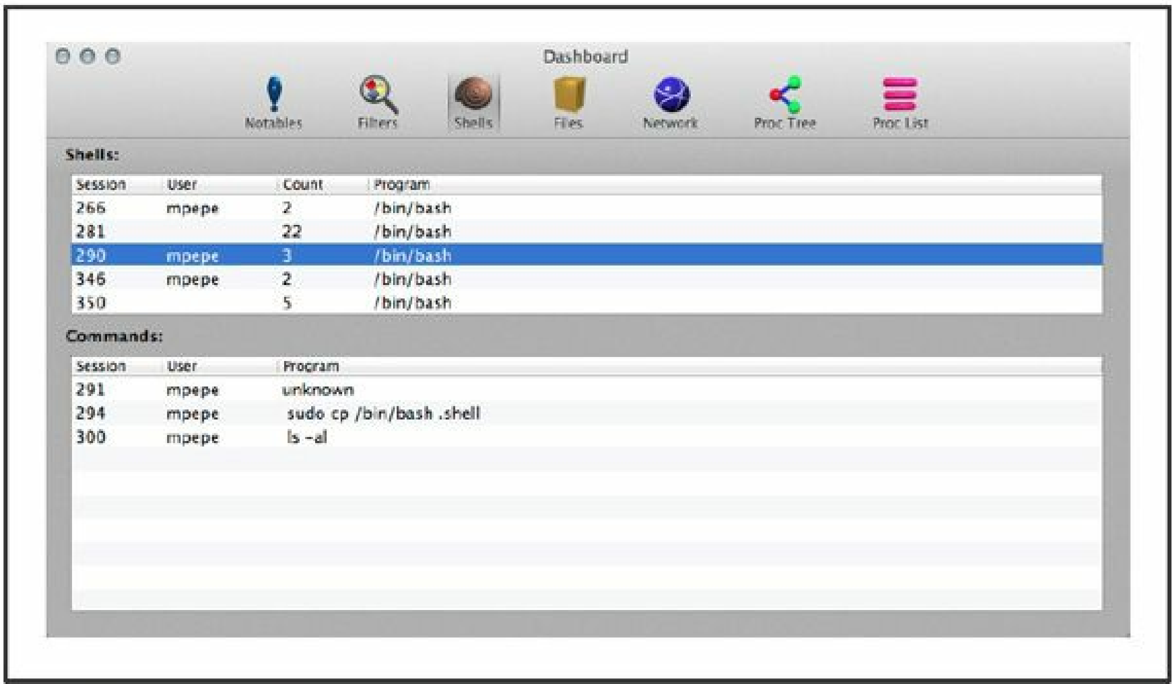
Note For more information on the OpenBSM project, visit http://www.trustedbsd.org/openbsm.html.

You can improve the fidelity of the data logged by auditd before an incident occurs. In the auditd configuration file, /etc/security/audit_control, make the following changes and reboot:

```
flags:all
naflags:lo,aa,pc,nt
policy:cnt,argv
filesz:1G
expire-after:10G
```

This would log everything for all users, plus login/logout, administrative events, processes, and network activity for processes whose actions are not attributable to a specific user. In the event you need more detail, you can instruct it to retain environment variables for each process. Be aware that this increases the size of the log files.

The most compelling reason to turn on these options can be summarized by Figure 13-9. This is Audit Explorer showing every command run for a short Secure Shell session. Reconstructing attacker activity is greatly simplified when Full Auditing is activated.



Shells:

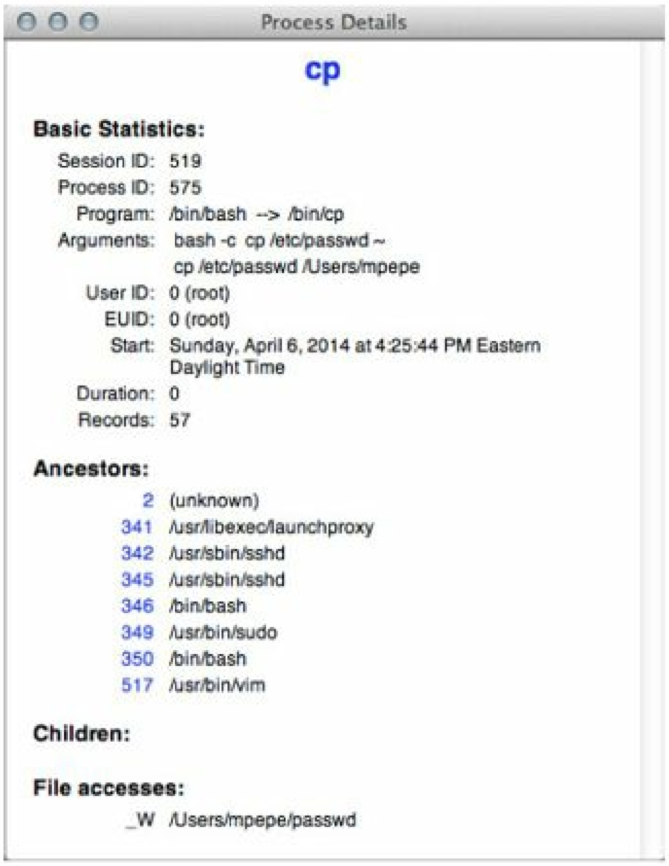
Session	User	Count	Program
266	mpepe	2	/bin/bash
281		22	/bin/bash
290	mpepe	3	/bin/bash
346	mpepe	2	/bin/bash
350		5	/bin/bash

Commands:

Session	User	Program
291	mpepe	unknown
294	mpepe	sudo cp /bin/bash .shell
300	mpepe	ls -al

Figure 13-9. Audited shell activity

Even activity performed with the old “run from within vi” trick gets captured. In Figure 13-10, the command `cp /etc/passwd ~` was run with the ! command in vi. Note that the parent PID is vim.



```
Process Details

cp

Basic Statistics:
  Session ID:  519
  Process ID:  575
     Program:  /bin/bash --> /bin/cp
   Arguments:  bash -c cp /etc/passwd ~
               cp /etc/passwd /Users/mpepe
     User ID:  0 (root)
        EUID:  0 (root)
       Start:  Sunday, April 6, 2014 at 4:25:44 PM Eastern
               Daylight Time
    Duration:  0
     Records:  57

Ancestors:
           2  (unknown)
         341  /usr/libexec/launchproxy
         342  /usr/sbin/sshd
         345  /usr/sbin/sshd
         346  /bin/bash
         349  /usr/bin/sudo
         350  /bin/bash
         517  /usr/bin/vim

Children:

File accesses:
          _W  /Users/mpepe/passwd
```

Figure 13-10. Commands issued within a shell

Mac OS X has a large number of "helper services" that run in the background. These services provide support to the system and user-facing applications by tracking events or common data. Most of the helper applications maintain state through the use of database files in SQLite or property list formats. At any point in time, the number of running system services may exceed 40 independent processes. A subset of those are interesting to us because they either perform a primary function or provide "helper services" to other applications. Here are a few examples of helper services:

- **airportd** Manages connections to wireless networks.
- **aosnotifyd** Daemon for the Find My Mac service.
- **pboard** The system pasteboard. Manages copy/cut/paste operations.
- **sharingd** Manages sharing data and drives with other systems.
- **spindump_agent** Helps Spindump monitor and report on application errors or hangs.

Some of these services retain data that can be reviewed, depending on the type of investigation you are performing. As an example, let’s look at the information that airportd retains.

Many services, including airportd, run in an application sandbox and therefore require a sandbox profile to run. We can use this information to track down the various data storage locations an application uses. Apple stores the application sandbox configuration files in /usr/share/sandbox. An excerpt from airportd’s configuration is shown here:

```
(allow file*
    (literal "/dev/io8log")
    (literal "/dev/io8logmt")
    (literal "/dev/io8logtemp")
    (regex #"^/dev/pf")
    (regex #"^/dev/bpf")
    (regex #"^/Library/Preferences/SystemConfiguration/preferences\.plist")
    (regex #"^/Library/Preferences/SystemConfiguration/com\
            .apple\.airport\.preferences\.plist")
    (regex #"^/Library/Preferences/SystemConfiguration/com\
            .apple\.wifi\.message-tracer\.plist")
)
```

Let’s review the contents of /Library/Preferences/SystemConfiguration/com.apple.airport.preferences.plist using the plutil command:

```
bash-3.2# plutil -p
/Library/Preferences/SystemConfiguration/com.apple.airport.preferences.plist
```

The output shows that the file is where airportd stores a list of access points and networks it connects to. We provide an example of the output in the next section as part of a scenario.

System and Application Logging

Like most Unix-derived features in Mac OS X, Apple has taken a bit of liberty in how system and application logging is performed. When performing an examination (or live response) of an Mac OS X system, you’ll want to be aware of the several locations where evidence may be found.

On Mac OS X workstations, you’ll find logs and a great number of forensic artifacts in the three primary locations listed next (note that /var/log is a symlink to /private/var/log on the file system):

- /private/var/log
- /Library/Logs

- /Users/*username*/Library/Logs
- /Users/*username*

The last item on the list refers to the “hidden” log files that are maintained by console applications. The user’s home directory contains shell histories, MySQL history, and other session log data. Generally, the logs are in plain text and can be analyzed in numerous ways, including cat, less, grep, text editors, Highlighter, or by importing into a log management tool such as Sawmill or LogStash. When log files are stored in a binary format, such as through Apple System Log (ASL) or auditd, you need to use a set of utilities to convert them to a human-readable form. We’ll walk through this conversion in the upcoming “Analysis” section.

The Evidence

The syslog and the Apple System Log (ASL) daemons are the processes responsible for most log data on Mac OS X. The largest volume of log data can be found in /private/var/log, so we will start there.

If you have exposure to logging facilities in modern Linux distributions, many of the files will be immediately familiar. Naturally, you’ll find a few Mac OS X–specific items. During examinations of Unix systems, one of the first files to review is the syslog configuration file. This configuration file defines where messages from syslog-capable daemons get placed or redirected. Apple has added an additional layer, called the Apple System Log (ASL) service. The ASL configuration file contains some entries that are traditionally defined in /etc/syslog.conf. In the default configuration file shown next, you can see that the file named system.log stores most messages. Of particular interest are the two facilities named auth and authpriv. These two facilities contain messages from all login events on the system.

```
##
# configuration file for syslogd and aslmanager
##

# authpriv messages are root/admin readable
? [= Facility authpriv] access 0 80

# remoteauth critical, alert, and emergency messages are root/admin readable
? [= Facility remoteauth] [<= Level critical] access 0 80

# broadcast emergency messages
? [= Level emergency] broadcast

# save kernel [PID 0] and launchd [PID 1] messages
? [<= PID 1] store

# ignore "internal" facility
? [= Facility internal] ignore

# save everything from emergency to notice
? [<= Level notice] store

# Rules for /var/log/system.log
> system.log mode=0640 format=bsd rotate=seq compress file_max=5M all_max=50M
? [= Sender kernel] file system.log
? [<= Level notice] file system.log

? [= Facility auth] [<= Level info] file system.log
? [= Facility authpriv] [<= Level info] file system.log

# Facility com.apple.alf.logging gets saved in appfirewall.log
? [= Facility com.apple.alf.logging] file appfirewall.log file_max=5M all_max=50M
```

The ASL provides additional logging features to Mac OS X applications, which allow logs to be stored and managed more efficiently. Normal syslog messages follow a simple structure:

- When the message arrives at a syslog daemon
- The source of the message
- The severity and type of the message
- The message itself

ASL is more flexible in what it can process. When a developer sends logs through ASL, they can specify a number of custom key/value pairs in addition to the syslog format. One can link to external files, specify the user or group ID that should receive the message, and set an expiration time for the message. In recent documents, Apple has suggested that developers begin using hashtags in the message text to assist in log searches. The suggested hashtags include #System, #Attention, #Security, and #Error. It’s quite an update from the old syslog.

ASL will store data in the directory /private/var/log/asl. The files are not in plain text, so you must use syslog to display their contents. To view a single file, use the following command. You can transcode the entire directory with the `-d flag, instead of -f.`

```
planck:~ user$ sudo syslog -f /private/var/log/asl/<file>.asl
```

The file names for the ASL logs have three defined formats. The first stores events for a particular user or group. The first portion of the file name is the date. The second represents the origin of the events. Here's an example:

- **2013.11.09.U501.asl** This file contains events from UID 501 on 9 November 2013.
- **2013.11.09.G80.asl** This file contains events from GID 80 on 9 November 2013.
- **2013.11.09.U0.G80.asl** This file contains events from UID 0, GID 80 on 9 November 2013. These events may include actions performed within a sudo context switch, because GID 80 is admin. When you look at the file names, compare them with the contents of /etc/passwd and /etc/group to get the mapping correct.

The second format has the two characters "BB" at the beginning of the file name. The date portion of the file name is typically one year ahead. The entries in these files, typically authentication events, are intended to be retained for a long period of time.

The final naming convention used in this log directory begins with the string "AUX". These are directories that contain the backtrace data for crashed or abnormally terminated applications. These files are in plain text, so the syslog command noted earlier is not necessary for analysis.

Analysis

Analysis of syslog and ASL data from a Mac OS X system is fairly straightforward. The logs are primarily text based, so as an analyst, you have a great deal of flexibility. Depending on the volume and format, we use several Unix tools such as cat, grep, and awk to carve up the log data. In some cases, importing the data into a spreadsheet yields the best view. Finally, if the logs are rather large and we want to look at several log formats concurrently, we may import the data into a log management tool, such as Sawmill, Splunk, or Logstash. These all provide a number of input transforms and good search capabilities.

GO GET IT ON THE WEB

Logstash www.logstash.net
Sawmill www.sawmill.net
Splunk www.splunk.com

The system logs on a Mac OS X system collect an incredible amount of information from all daemons and applications. It can be difficult to enumerate the types of data you may find in the system log, so in a moment, we'll walk through a couple of excerpts from our own logs to illustrate the information you may find with a bit of searching.

OpenBSM process audit log data is stored in /private/var/audit. This logging facility tracks all authentication events and stores them in a non-ASCII file format. The OpenBSM project is cross-platform, allowing us to use a Linux or BSD platform for analysis. To extract human-parsable events from the data files, use the utility praudit. In the following example, praudit is used to extract authentication activity. Three events are shown, each multilist record begins with "header" and ends with "trailer" and a total message length. We've selected the "simple" output format for this section; however, you may find the XML-formatted output to be easier to parse because record labels are included.

The extracts shown were generated with the command line `praudit -s [filename]`. We manually searched for the authentication events. The -s flag makes the output easy to put into a printed format, but it omits the field descriptions. If you use XML output with the –x option, all the field names will be printed.

```
header,326,11,AUE_ssauthorize,0,Sat Nov  9 20:27:45 2014, + 134 msec
subject,user,user,staff,user,staff,27748,100005,27749,0.0.0.0
text,system.preferences
text,client /System/Library/PrivateFrameworks/
      SystemAdministration.framework/XPCServices/writeconfig.xpc
text,creator /System/Library/PreferencePanes/
      SharingPref.prefPane/Contents/XPCServices/
      com.apple.preferences.sharing.remoteservice.xpc
return,success,0
trailer,326

header,88,11,AUE_ssauthorize,0,Sat Nov  9 20:27:45 2014, + 135 msec
subject,-1,root,wheel,root,wheel,27471,100000,27472,0.0.0.0
text,begin evaluation
return,success,0
trailer,88

header,252,11,AUE_ssauthorize,0,Sat Nov  9 20:27:45 2014, + 137 msec
subject,-1,root,wheel,root,wheel,27471,100000,27472,0.0.0.0
text,com.apple.ServiceManagement.daemons.modify
text,client /usr/libexec/launchdadd
text,creator /System/Library/PrivateFrameworks/
      SystemAdministration.framework/XPCServices/writeconfig.xpc
return,success,0
```

The first record shows that the user named “user” opened the Sharing pane in System Preferences at 20:27 on Saturday, November 9, 2013. Note that the file date and time (not shown in the excerpt) tells us the year the log entries were written. Due to the AUE_ssauthorize event type, the second record shows a successful authentication event, similar to a `su root`, allowing the normal user the ability to make changes. The third record shows that a change was successful. We can’t tell from this log what property was changed, unfortunately.

Approximately three minutes later, the following event was recorded. The same user ID logged in to the system through a Secure Shell session, where the event type was AUE_openssh.

```
header,110,11,AUE_openssh,0,Sat Nov  9 20:31:29 2014, + 759 msec
subject_ex,user,user,staff,user,staff,27775,27775,56091,::1
text,successful login user
return,success,0
trailer,110
```

The next example is an excerpt from /private/var/log/system.log that was generated after the network cable was pulled from the Ethernet interface on a MacBook Pro:

```
Jan 23 20:08:24 planck kernel[0]: AppleBCM5701Ethernet [en0]:
      Link down (womp disabled, proxy idle)
Jan 23 20:08:25 planck.local configd[17]: setting hostname to "planck.local"
Jan 23 20:08:26 planck.local configd[17]: network changed:
      v4(en0-:10.1.4.105) DNS- Proxy- SMB-
Jan 23 20:08:26 planck.local netbiosd[11488]: network_reachability_changed:
      network is not reachable, netbiosd is shutting down
Jan 23 20:08:26 planck.local com.apple.iCloudHelper[14236]:
      AOSKit ERROR: Config request failed,
      url=https://setup.icloud.com/configurations/init, requestHeaders=
      {
          "Accept-Language" = "en-us";
          "X-Mme-Client-Info" = "<MacBookPro8,2> <Mac Mac OS X;10.9.2;13C64>
            <com.apple.AOSKit/176>";
          "X-Mme-Country" = US;
          "X-Mme-Nac-Version" = 11A457;
          "X-Mme-Timezone" = EDT;
      },
      error=Error Domain=kCFErrorDomainCFNetwork Code=-1009 "The Internet
      connection appears to be offline." UserInfo=0x092e027fa09b
      {NSErrorFailingURLStringKey=https://setup.icloud.com/configurations/
      init, NSLocalizedDescription=The Internet connection appears to be
      offline., NSErrorFailingURLKey=https://setup.icloud.com/
      configurations/init}, httpStatusCode=-1, responseHeaders=
      (null)
Jan 23 20:08:41 planck.local AddressBookSourceSync[14238]:
      tcp_connection_destination_prepare_complete 1 connectx to
      10.1.4.4#443 failed: 51 - Network is unreachable
```

Let's walk through the log entries and see what information can be extracted. In the first line, we get a notice from the kernel that the interface has gone down. This event triggers a series of actions, the first being an update to the system's network configuration. The host's domain name is changed to ".local" by the process known as configd. Configd then logs the address it is releasing (10.1.4.105) and notes that the services DNS, Proxy, and SMB (Samba file sharing) have been deactivated (note the minus signs in the output string). One of the daemons responsible for a portion of the SMB protocol, netbiosd, reports it is shutting down. On the fifth line, the iCloudHelper process failed to get a "config request" filled. This tells us that there is an iCloud account tied to the system, although by now, that may already be known. Another thing to note here is that many services and applications send the current OS version into syslog. This can be very useful when you suspect that the primary user has attempted to alter log data or the operating system itself. Finally, the last line shows that the AddressBookSourceSync process failed to connect to 10.1.4.4. Again, this is another potential lead that may inform you of user data located on remote servers.

A short time later, the following messages were sent to syslog:

```
Jan 23 20:09:16 planck.local KernelEventAgent[99]: tid 54485244
      received event(s) VQ_DEAD (32)
Jan 23 20:09:16 planck.local KernelEventAgent[99]: tid 54485244
      type 'afpfs', mounted on '/Volumes/TMBackup', from
      '//TimeMachine@nas0%28TimeMachine%29._afpovertcp._tcp.local/TMBackup',
      dead
Jan 23 20:09:16 planck.local KernelEventAgent[99]: tid 54485244
      force unmount
      //TimeMachine@nas0%28TimeMachine%29._afpovertcp._tcp.local/TMBackup
      from /Volumes/TMBackup
Jan 23 20:09:16 planck.local KernelEventAgent[99]: tid 54485244
      found 1 filesystem(s) with problem(s)
```

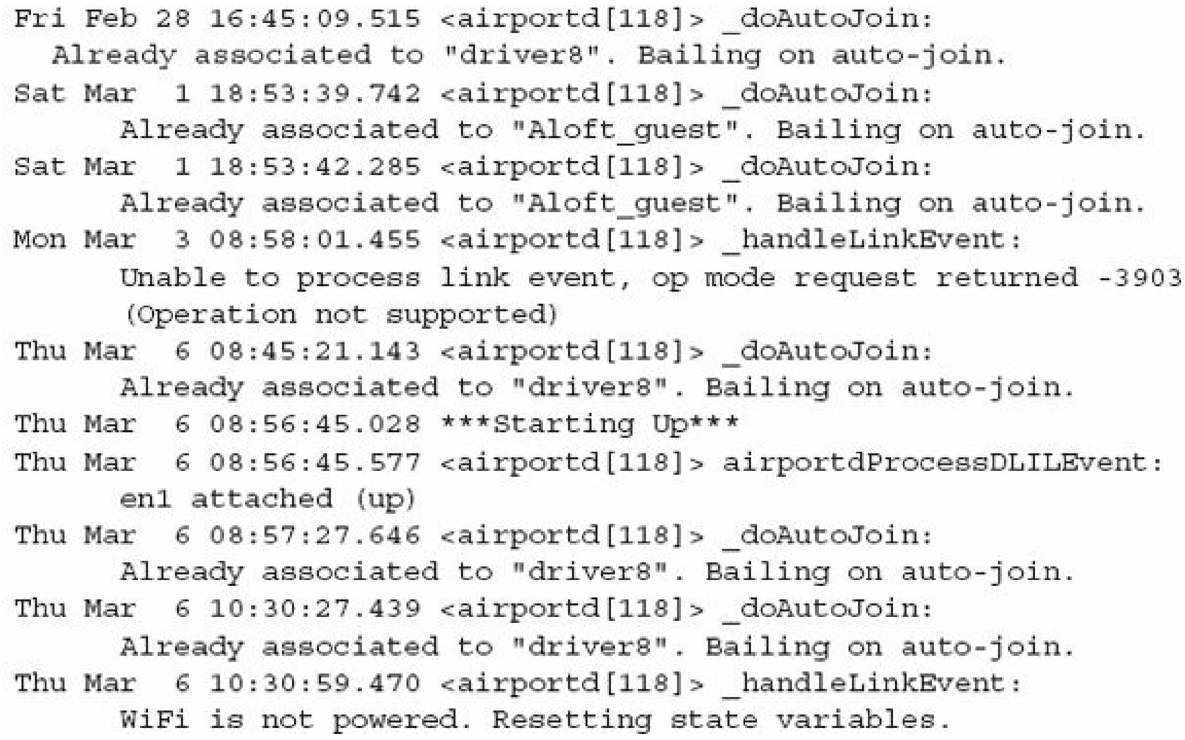
This is yet another lead. It appears that this system was configured to use a remote disk for Time Machine backups. The system had an Apple File Protocol (AFP) connection to a mount point (TimeMachine) on a system known as nas0. If you were searching an unfamiliar work area, it's time to start searching the local LAN for network storage devices named nas0.

When the network cable is replaced, services restart and the process just shown is replayed (hopefully successfully). The following excerpt shows the network services restarting (note the plus signs in the second log entry):

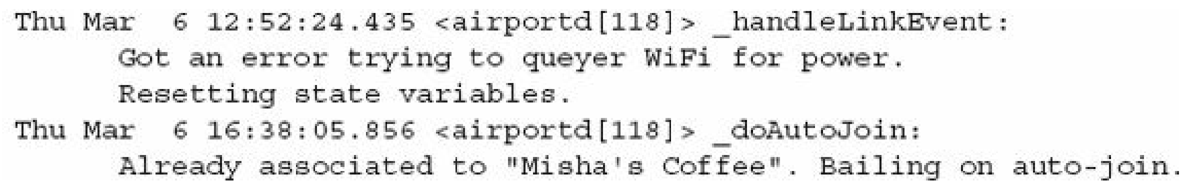
```
Jan 23 20:11:50 planck kernel[0]: Ethernet [AppleBCM5701Ethernet]:
      Link up on en0, 1-Gigabit, Full-duplex, Symmetric flow-control,
      Debug [796d,2321,0de1,0300,cde1,3c00]
Jan 23 20:11:50 planck.local configd[17]: network changed:
      v4(en0+:10.1.4.105) DNS+ Proxy+ SMB+
Jan 23 20:11:50 planck.domainname.net configd[17]:
      setting hostname to "planck.domainname.net"
Jan 23 20:12:42 planck.local com.apple.usbmuxd[77]:
       SendAttachNotification Device
      88:53:95:12:60:1c@fe80::8a53:95ff:fe12:601c._apple-mobdev2._tcp.local.
      has already appeared on interface 5. Suppressing duplicate
      attach notification.
```

The last entry, from usbmuxd, is an interesting message. It tells us that when the laptop regained network access, it reconnected to an Apple mobile device that had been paired with iTunes. The device has a MAC address of 88:53:95:12:60:1c. Besides an indication that additional platforms are in use, you should start thinking about how to get access to the backups of the device that likely reside on the system.

The next example is from the very descriptively named wifi.log in /private/var/log. We discussed the service named airportd earlier. As you may expect, this log from airportd contains IEEE 802.11 association events. The following excerpt shows a series of associations during a seven-day period:



```
Fri Feb 28 16:45:09.515 <airportd[118]> _doAutoJoin:
  Already associated to "driver8". Bailing on auto-join.
Sat Mar  1 18:53:39.742 <airportd[118]> _doAutoJoin:
      Already associated to "Aloft_guest". Bailing on auto-join.
Sat Mar  1 18:53:42.285 <airportd[118]> _doAutoJoin:
      Already associated to "Aloft_guest". Bailing on auto-join.
Mon Mar  3 08:58:01.455 <airportd[118]> _handleLinkEvent:
      Unable to process link event, op mode request returned -3903
      (Operation not supported)
Thu Mar  6 08:45:21.143 <airportd[118]> _doAutoJoin:
      Already associated to "driver8". Bailing on auto-join.
Thu Mar  6 08:56:45.028 ***Starting Up***
Thu Mar  6 08:56:45.577 <airportd[118]> airportdProcessDLILEvent:
      en1 attached (up)
Thu Mar  6 08:57:27.646 <airportd[118]> _doAutoJoin:
      Already associated to "driver8". Bailing on auto-join.
Thu Mar  6 10:30:27.439 <airportd[118]> _doAutoJoin:
      Already associated to "driver8". Bailing on auto-join.
Thu Mar  6 10:30:59.470 <airportd[118]> _handleLinkEvent:
      WiFi is not powered. Resetting state variables.
```



```
Thu Mar  6 12:52:24.435 <airportd[118]> _handleLinkEvent:
      Got an error trying to queyer WiFi for power.
      Resetting state variables.
Thu Mar  6 16:38:05.856 <airportd[118]> _doAutoJoin:
      Already associated to "Misha's Coffee". Bailing on auto-join.
```

This log was created in 2014, and it shows a sequence of events where the user was connected to access point driver8 on a Friday, Aloft_guest on a Saturday, back to driver8 on Monday, and then Misha's Coffee on Thursday.

As we mentioned previously, one of Apple's strengths is a well-polished user experience, so let's look at other files to determine whether airportd or any other daemons captured information related to these entries in wifi.log. We mentioned earlier that airportd maintained a list of recent 802.11 station associations in /Library/Preferences/SystemConfiguration/com.apple.airport.preferences.plist. Let's look in that file to see if additional data was captured about these access points.

By executing plutil -p we can quickly find relevant entries. In the following excerpt, we learn that the access point named driver8 has a BSSID (or access point MAC address) of c0:c1:c0:15:6e:e2:

```
"CachedScanRecord" => {
    "SSID_STR" => "driver8"
  "WPS_PROB_RESP_IE" => {
    "IE_KEY_WPS_RESP_TYPE" => 3
    "IE_KEY_WPS_MODEL_NAME" => "WNDR4500"
    "IE_KEY_WPS_SERIAL_NUM" => "4536"
    "IE_KEY_WPS_RF_BANDS" => 3
    "IE_KEY_WPS_DEV_NAME" => "driver8"
    "IE_KEY_WPS_MANUFACTURER" => "NETGEAR, Inc."
  }
  "BEACON_INT" => 100
  "AGE" => 0
  "RSSI" => -86
  "BSSID" => "c0:c1:c0:15:6e:e2"
  "AP_MODE" => 2
  "SSID" => <64726976 657238>
  "CHANNEL" => 6
}
```

Additional searches of the Airport property list for the SSIDs Aloft_guest and Misha's Coffee yield the following BSSIDs: 00:1a:1e:ce:d9:10 and 84:1b:5e:f7:25:e1. With this information, we can retrace the locations that the system visited over that seven-day period. Through sites such as the Wireless Geographic Logging Engine and tools such as iSniff GPS, we can tell that this user is likely to live in Arlington, Virginia, visits the National Harbor and Old Town Alexandria, and is an R.E.M. fan. In a traditional investigation where the actions of a user are of more interest than actions by an intruder, you could then use those data points to review timelines and other usergenerated content.

GO GET IT ON THE WEB

iSniff GPS github.com/hubert3/iSniff-GPS
Wireless Geographic Logging Engine wigle.net

Scheduled Tasks and Services

Apple has replaced the traditional rc and CRON systems with launchd. For a number of releases, Apple has been migrating away from traditional cron. Launchd has a number of advantages, including the ability to recognize that laptops and workstations are actually powered down or put to sleep, and it will reschedule missed tasks accordingly. Additionally, it provides a framework for events to fire from other triggers than simply time.

The Evidence

All launchd configurations are stored as XML in five directories, listed next. The list is taken directly from the man page for launchd.

```
~/Library/LaunchAgents          Per-user agents provided by the user.
/Library/LaunchAgents           Per-user agents provided by the administrator.
/Library/LaunchDaemons          System-wide daemons provided by
                                the administrator.
/System/Library/LaunchAgents    Per-user agents provided by Mac OS X.
/System/Library/LaunchDaemons   System-wide daemons provided by Mac OS X.
```

Additionally, the command that manages launchd, launchctl, maintains lists of commands to execute at startup. These configuration files are listed here:

```
$HOME/.launchd.conf
/etc/launchd.conf
```

The LaunchDaemon directories store XML definition files for services, such as sshd and Apache. The LaunchAgent directories store definition files for CRON-like actions. Tasks, when loaded into launchd, are considered to be active. The options in the XML dictate when certain actions take place. The key StartCalendarInterval is equivalent to normal CRON. The man page for launchd.plist has the complete listing of property list keys, but here are a few of the interesting properties for LaunchAgents that are available:

- **KeepAlive** This property tells launchd to ensure that the process is kept alive under certain conditions. It can be used as a watchdog to restart jobs, should they exit for any reason.
- **WatchPaths** Launchd will start the task if the path is modified.
- **StartOnMount** The task is started whenever a file system is mounted successfully.
- **ExitTimeout** Launchd can force-terminate a process if its run time exceeds a given value.

LaunchDaemons are analogous to run control files, but again provide additional capabilities. In addition to the properties listed, one can also set socket options and request that the Bonjour service publish an announcement.

Note

Bonjour is Apple’s implementation of Zero Configuration Networking and allows systems to broadcast the availability of services over a local network. This is how many of Apple’s applications discover people and devices nearby, from printers and scanners to humans using iChat.

Analysis

The service configuration files for LaunchAgents and LaunchDaemons are plain-text XML, so analysis is simple. When reviewing each for signs of malicious persistence, use the file system and BOM files to determine whether the services are legitimate. Note that both LaunchAgents and LaunchDaemons can execute a shell just as easily as an application, so follow all Program key/value pairs.

As an example of a valid LaunchDaemon service, the following excerpt is from the sshd service declaration file in /System/Library/LaunchDaemons:

```
<dict>
        <key>Disabled</key>
        <true/>
        <key>Label</key>
        <string>com.openssh.sshd</string>
        <key>Program</key>
        <string>/usr/libexec/sshd-keygen-wrapper</string>
        <key>ProgramArguments</key>
        <array>
                <string>/usr/sbin/sshd</string>
                <string>-i</string>
        </array>
        <key>Sockets</key>
        <dict>
                <key>Listeners</key>
                <dict>
                        <key>SockServiceName</key>
                        <string>ssh</string>
                        <key>Bonjour</key>
                        <array>
                                <string>ssh</string>
                                <string>sftp-ssh</string>
                        </array>
                </dict>
        </dict>
        <key>inetdCompatibility</key>
        <dict>
                <key>Wait</key>
                <false/>
        </dict>
        <key>StandardErrorPath</key>
        <string>/dev/null</string>
        <key>SHAuthorizationRight</key>
        <string>system.preferences</string>
        <key>POSIXSpawnType</key>
        <string>Interactive</string>
</dict>
```

The XML file tells launchd that the sshd service is currently disabled. However, if we were to enable the service, it would be advertised over Bonjour as sftp-ssh.

Application Installers

When applications are installed, the installer framework will store information about the files that are placed on the drive. In /private/var/db/receipts, there are typically two files retained per installation: a bill of materials (BOM) containing a complete inventory of files, and a plist that lists the install date, package identifier, and path access control lists. The inventory contains file names, complete paths, file system metadata, and a 32-bit checksum. The following excerpt shows the metadata for a recent install of Autodesk AutoCAD WS:

```
planck# plutil -p com.autodesk.mac.AutoCAD-WS.plist
{
  "PackageVersion" => "2.0.3"
  "PackageIdentifier" => "com.autodesk.mac.AutoCAD-WS"
  "InstallPrefixPath" => "Applications"
  "InstallDate" => 2013-09-13 18:28:48 +0000
  "PackageFileName" => "com.autodesk.mac.AutoCAD-WS.pkg"
  "InstallProcessName" => "storeagent"
}
```

We can tell from the plist file that the application was installed on September 13, 2013. In the associated BOM, we can get a list of the files that were placed on the file system during the install. The following output shows the first six lines (the full output is 234-lines long):

```
planck# lsbom -pfMTSc  com.autodesk.mac.AutoCAD-WS.bom
.       drwxr-xr-x
./AutoCAD WS.app        drwxr-xr-x
./AutoCAD WS.app/Contents       drwxr-xr-x
./AutoCAD WS.app/Contents/Info.plist    -rw-r--r--
      Thu Jul 11 07:25:35 2013        2,303   647692177
./AutoCAD WS.app/Contents/MacOS drwxr-xr-x
./AutoCAD WS.app/Contents/MacOS/AutoCAD WS      -rwxr-xr-x
      Tue Jul 16 19:36:53 2013        11,630,544      184272354
```

We listed the contents of this BOM with the tool lsbom. The options you see here format the output to include the (f) file name, (M) file mode, (T) formatted modification time, (S) formatted size, and (C) CRC32 checksum.

A REVIEW: ANSWERING COMMON INVESTIGATIVE QUESTIONS

Let's take a step back from the weeds and address common situations that you will likely encounter with performing IR investigations. As in Chapter 12, we'll give you a few scenarios and possible solutions. There are typically a number of methods to answer each question, and the methods will change as new versions of Mac OS X are released.

What sources of evidence can I use for timeline analysis?

Artifact	Time-based Evidence Source
HFS+ directory entries	File Access, File Modify, Inode change, Inode Birth timestamps
Syslog and ASL entries	Entry generated time
Wireless connection logs	Entry generated time
Spotlight indexer	Created and modified timestamps
Cron jobs	Scheduled run time, previous run times in logs
OS install date	File Access, File Modify, Inode change, Inode Birth timestamps
Application install date	BOM files
OpenBSM entries	Entry generated time
Application plist files	File system metadata; dates tracked by the applications and set in their plist files
Document revisions	Revision creation dates
Document metadata	Dates stored by applications within certain types of data files

What services were running or what shares were available when the system was imaged?

Artifact	Evidence Source

Directory Services	List of SMB and AFP shares
Contents of /var/run	State files and PID files

What system information can I gather from a static image?

Artifact	**Evidence Source**
System host name	/Library/Preferences/SystemConfiguration/preferences.plist.
OS version information	/System/Library/CoreService/SystemVersion.plist.
IP addresses	If defined in the Network Preferences, /Library/Preferences/SystemConfiguration/preferences.plist. If configured for DHCP, /private/var/db/dhcpclient/leases/.
Date of OS install	File creation date of /private/var/db/.AppleSetupDone or the InstallDate value in /private/var/db/receipts/com.apple.pkg.InstallMacOSX.plist.
System time zone, connected printers (via color profiles)	/Library/Preferences/.GlobalPreferences.plist. Note that if the user allows Location Services to set the time zone, this file contains the latitude and longitude of the recent locations.

What sources of evidence can prove that a file was recently opened?

Artifact	**Evidence Available**
/Users/*username*/Library/Preferences/com.apple.recentitems.plist	The 10 most recently run applications, connected server information, and documents recently accessed by the user

What artifacts can provide evidence of deleted files?

Artifact	Evidence Available
/Users/*username*/.Trash	Items moved to Trash. If Trash has been emptied, this folder will be empty. Note that files removed outside of Finder are not moved to a Trash folder and are immediately unlinked.

What files are configured to maintain persistence (automatically run) upon bootup or user login?

Artifact	Evidence Available
/Library/LaunchAgents /System/Library/LaunchAgents ~/Library/LaunchAgents	Agents started by the system or per user
/Library/LaunchDaemons /System/Library/LaunchDaemons ~/Library/LaunchDaemons	Daemons started by the system or per user
/Library/StartupItems /System/Library/StartupItems	Legacy system startup items (predating launchd)

Who interactively logged on to a system? At what date(s) and time(s) did a given user log on to a system?

Artifact	Evidence Available
Authentication logs	Contents of /var/log/authd.log and ASL logs in /var/log/asl
File system	Creation of user profile directory (for example /Users/[username]) and associated subdirectories; configuration plists after interactive logon by an account
utmp data	The current logged-in users and the processes that are running

SO WHAT?

Apple Mac OS X has become prevalent in many organizations, and although its presence in server roles is infrequent, many organizations have adopted the platform as an alternative to Microsoft Windows. We've found the platform in every department, from the traditional marketing and sales force to software engineering and IT. A limited number of tools exist to reliably perform examination on the platform, specifically from BlackBag Technologies, but your best bet is currently not the top-tier forensic suites. Oftentimes, examination of a drive image in the native environment or a Linux system will yield the most reliable results. This short chapter touched on a few of the most useful sources of evidence, but there are numerous others that have yet to be fully documented.

QUESTIONS

1. In an investigation, a network administrator notifies you that a Mac OS X system appears to be generating significant traffic that doesn't correlate with normal activity. How would you identify the source of the traffic on the system?
2. What would be the top files you would want to obtain during a live response to augment the data sources discussed in Chapter 7?
3. The Unix `touch` command allows one to update a file's access and modification times. What other sources of time-based data could one use to establish whether manipulation occurred?
4. Name a few persistence mechanisms that an attacker can use to maintain access to a compromised Mac OS X system. What live response commands can you use to automate a review for those mechanisms?