# Data Hiding Tactics for Windows and Unix File Systems

HAL BERGHEL

*Identity Theft and Financial Fraud Research and Operations Center, University of Las Vegas, Las Vegas, Nevada*

DAVID HOELZER

*Enclave Forensics, Las Vegas, Nevada*

MICHAEL STHULTZ

*Identity Theft and Financial Fraud Research and Operations Center, University of Las Vegas, Las Vegas, Nevada*

**Abstract**

The phenomenon of hiding digital data is as old as the computer systems they reside on. Various incarnations of data hiding have found their way into modern computing experience from the storage of data on out-of-standard tracks on floppy disks that were beyond the reach of the operating system, to storage information in non-data fields of network packets. The common theme is that digital data hiding involves the storage of information in places where data is not expected.

Hidden data may be thought of as a special case intentionally 'dark data' versus unintentionally dark data that is concealed, undiscovered, misplaced, absent, accidentally erased, and so on. In some cases, dark and light data coexist. Watermarking provides an example where dark data (an imperceptible watermark) resides within light data. Encryption is an interesting contrast, because it produces light data with a dark message. The variations on this theme are endless.

The focus of this chapter will be intentionally dark or hidden data as it resides on modern file systems – specifically some popular Windows and Unix file systems. We will extrapolate from several examples implications on the practice of modern digital forensics and the tools that are used in support thereof.

One forensically interesting dimension of physical data hiding is those techniques that take advantage of the physical characteristics of formatted storage media to hide data. An early attempt to do this was illustrated by Camouflage (camouflage.unfiction.com) that hid data in the area between the logical end-of-file and the end of the associated cluster in which the file was placed (called file slack or slack space). Although primitive, hiding data in file slack has the dual advantage that the host or carrier file is unaffected while the hidden data is transparent to the host operating system and file managers. The disadvantage is that the hidden message is easily recovered with a basic disk editor.

The ability to hide data on computer storage media is a byproduct of the system and peripheral architectures. If all storage were bit-addressable at the operating system level, there would be no place to hide data, hence no physical concealment. But for efficiency considerations, system addressability has to be at more abstract levels (typically words in primary, and blocks in secondary). Such abstractions create digital warrens where data may go unnoticed or, in some cases, be inaccessible. We investigate some of these warrens below.

1. The Philosophy of Digital Data Hiding . . . 3
1.1. The Concept of Data Hiding . . . 3
1.2. Physical Aspect of Data Hiding . . . 3
2. Digital Storage and File Systems . . . 4
2.1. Disk Structures . . . 4
2.2. Virtual File Systems . . . 6
2.3. Partition Organization . . . 7
2.4. ExtX . . . 8
2.5. NTFS . . . 8
3. Forensic Implications . . . 10
3.1. Fat16 . . . 10
3.2. NTFS . . . 12
4. Perspectives . . . 14
5. Conclusion . . . 16
References . . . 16

# 1. The Philosophy of Digital Data Hiding

## 1.1 The Concept of Data Hiding

Digital data hiding is actually a cluster concept that spans many contexts. In modern times, nonphysical data hiding is usually associated with digital forms such as cryptography, steganography, and watermarking. Although related in the sense that they all are means to achieve secure or proprietary communications, there are differences among their three activities at a number of levels – some of which are quite subtle. To illustrate, the cryptographer's interest is primarily with obscuring the content of a message, but not the communication of the message. The steganographer, on the other hand, is concerned with hiding the very communication of the message, while the digital watermarker attempts to add sufficient metadata to a message to establish ownership, provenance, source, and so on. Cryptography and steganography share the feature that the object of interest is embedded, hidden, or obscured, whereas the object of interest in watermarking is the host or carrier which is being protected by the object that is embedded, hidden, or obscured. Further, watermarking and steganography may be used with or without cryptography; and imperceptible watermarking shares functionality with steganography, whereas perceptible watermarking does not. Overviews exist for cryptography [13], steganography [9, 14], and watermarking [3].

## 1.2 Physical Aspect of Data Hiding

However, there is also a physical aspect of digital data hiding. In this case, digital storage locations are used to hide or conceal data. Obviously, these storage locations must be somewhat obscure or detection would be trivial. Examples include covert channeling (e.g., within TCP/IP packet headers or Loki's use of the ICMP options field) or obscure or infrequently used addressable space in memory media. One of the earliest examples of this arose when microcomputers were first introduced. There was a difference between the number of tracks that a floppy drive controller could access (usually 81 or 82) and the number of tracks that were recognized by the operating system (DOS recognized only 80). The upper two tracks were used to hide data by vendors and hackers alike. For awhile they were even used for product licensing.

A modern analogue might involve taking advantage of the physical characteristics of a storage medium to hide data. The steganographic tool Camouflage was such a program (camouflage.unfiction.com). Camouflage embeds messages in the file slack (see below). This simple approach to data hiding has the advantage that the characteristics of the host or carrier message remain unaffected but are transparent to the host operating system and file managers. The disadvantage is that the hidden message is easily recovered with a hex-editor. Although this approach has not met with great success (and camouflage is no longer supported), it is a great segue into the art of data hiding by taking advantage of the physical characteristics of computer systems.

The ability to hide data in computers is a byproduct of the system and peripheral architectures. If all storage were bit-addressable at the operating system level, there would be no place to hide data. For efficiency considerations, addressability has to be at more abstract levels (typically words in primary, and blocks in secondary). Such abstractions create digital warrens where data may go unnoticed or, in some cases, be inaccessible.

## 2. Digital Storage and File Systems

### 2.1 Disk Structures

Digital storage encompasses a wide range of media including diskettes, hard drives, zip disks, USB flash drives, compact flash cards, CD-ROMs, DVDs, and so on. Since the structures of most of these media can be related to hard drives, a discussion of hard drive architecture will serve to illustrate the various data hiding mechanisms.

Hard drive technology existed long before the advent of personal computers (although the early personal computers started out with only cassette tape or diskette storage!). As PCs evolved, the emphasis on backward compatibility and the need for user convenience (i.e., making data structures totally transparent to the user) greatly influenced the development of secondary and tertiary storage technology – both physically and logically. An unintended consequence has been the creation of many places where data can be intentionally hidden or unintentionally left behind.

A functioning hard drive actually consists of a geometric structure and a set of nested data structures: hard drive, partition, file system, file, record, and field. Hidden data can be found at each of these levels. We will assume that the reader is familiar with basic hard disk geometry (cylinders, tracks, blocks, clusters, and sectors). These structures create areas on secondary storage devices where hidden data or data residue could exist. Figure 1 provides a graphical illustration of these digital warrens. In each example, the shaded area represents spaces within the structures where hidden data could reside.

The following describes the various hiding mechanisms (as illustrated in Fig. 1), starting at the level of the hard drive itself and then working down through the set of nested data structures.

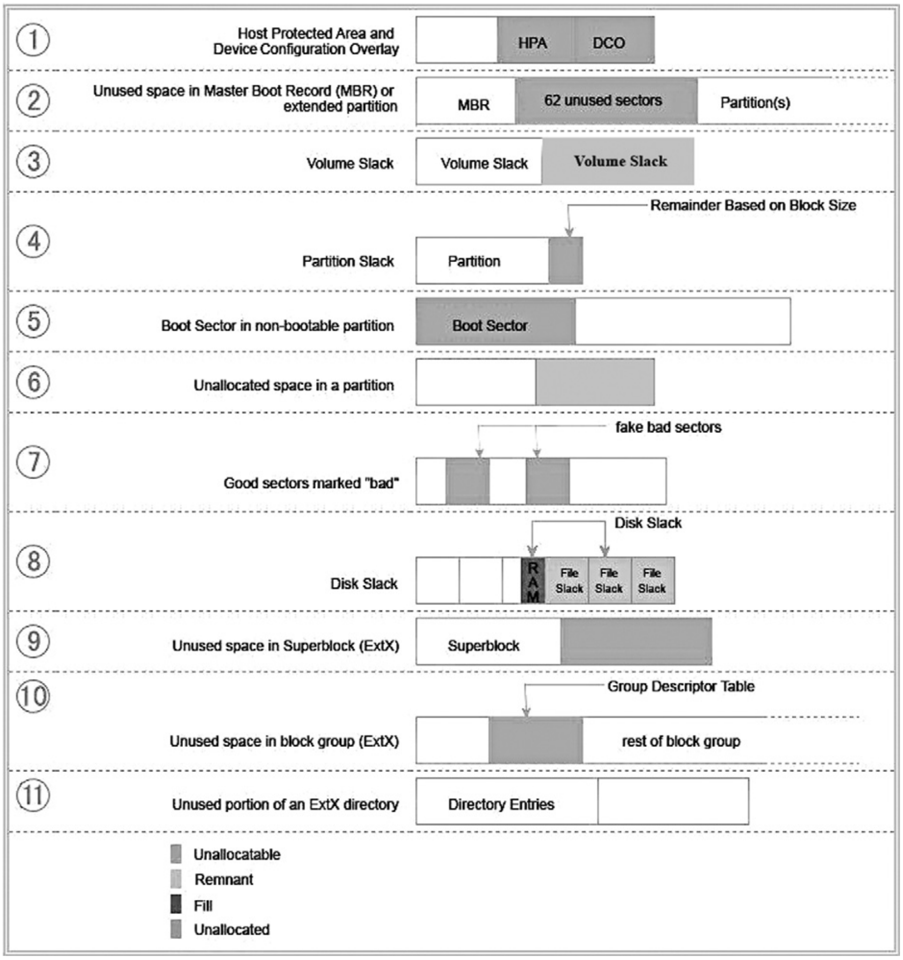

1
Host Protected Area and Device Configuration Overlay
HPA
DCO
2
Unused space in Master Boot Record (MBR) or extended partition
MBR
62 unused sectors
Partition(s)
3
Volume Slack
Volume Slack
Volume Slack
4
Remainder Based on Block Size
Partition Slack
Partition
5
Boot Sector in non-bootable partition
Boot Sector
6
Unallocated space in a partition
7
fake bad sectors
Good sectors marked "bad"
8
Disk Slack
Disk Slack
RAM
File Slack
File Slack
File Slack
9
Unused space in Superblock (ExtX)
Superblock
10
Group Descriptor Table
Unused space in block group (ExtX)
rest of block group
11
Unused portion of an ExtX directory
Directory Entries
Unallocatable
Remnant
Fill
Unallocated

FIG. 1. Digital disk warrens.

Some hard drives can have a reserved area of the disk called the *Host Protected Area* (*HPA*) (see Fig. 1, item 1). *Device Configuration Overlay* allows modification of the apparent features provided by a hard drive, for example, the number of available clusters. This was designed to be an area where computer vendors could store data that is protected from normal user activities. It is not affected by operating system utilities (format, delete, etc.) and cannot be accessed without the use of a special program that reconfigures the controller to access all physical blocks. It is not difficult, however, to write a program to access these areas, write data to them, and subsequently return the area to an HPA. This is an example of a hiding method that takes advantage of what is more or less a ‘physical’ feature of the drive architecture.

## 2.2 Virtual File Systems

At the next layer, common operating systems typically require that a hard drive be partitioned into virtual file systems before it can be used. This is true even if the entire hard drive is to be mapped onto a single partition. A partition is a set of consecutive blocks on a hard disk that appear to the operating system as a separate logical volume (drive for Windows vs directory or mount point for Unix). Note that there are several different types of partition formats. This discussion is confined to the partition format that has evolved from the original PC (often referred to as DOS partitions) and will not apply to Apple, xBSD, Sun Solaris, GPT partitioning, or multiple disk volumes. A recommended source for additional detail appears as a reference [6]. Even so, the various data hiding techniques at the data and file system levels are partition format independent.

Every hard drive using a DOS partition has space reserved at the beginning of the drive for a *Master Boot Record* (*MBR*) (see Fig. 1, item 2). This will often contain the boot code necessary to begin the initial program load of an operating system and will always contain a partition table (provided we are dealing with partitioned media; e.g., floppy disks are not typically partitioned media while fixed disks are) defining the size and location of up to four partitions. Since the MBR requires only a single sector and partitions must start on a cylinder boundary, this results in 62 sectors of *empty MBR* space where data can be hidden.

As disk sizes grew beyond the limitations of existing operating systems, there arose a need for more than four partitions on a single hard disk. *Extended partitions* (as opposed to primary partitions) were then designed that could contain multiple logical partitions. Each of these extended partitions contains a structure similar to the MBR, leaving another 62 sectors within each extended partition with the potential of harboring more hidden data. Extended partitions may contain at most one file system and one extended partition, so this design permits us to nest the extended partitions to satisfy our volume requirements. Of course, each iteration creates yet another convenient hiding place for data.

If the partitions on a hard drive do not use up all of the available space, the remaining area cannot be accessed by the operating system by conventional means (e.g., through Windows Explorer). This wasted space is called *volume slack* (see Fig. 1, item 3). It is possible to create two or more partitions, put some data into them, and then delete one of the partitions. Since deleting the partition does not actually delete the data, that data is now hidden.

## 2.3 Partition Organization

Once partitions have been defined, we are ready to move up to the next layer and create an organizational structure for each partition. Before an operating system can store and access data within a partition, a file system must be defined. Modern operating systems support one or more native file systems. A file system allocates data in blocks (or clusters) where a block consists of one or more consecutive sectors. This allocation scheme allows a smaller number of references to handle larger amounts of data, but limits us to accessing data within the file system as block-sized chunks rather than sector-sized chunks. Overall, this tends to make storage and access far more efficient than referencing each individual sector. However, if the total number of sectors in a partition is not a multiple of the block size, there will be some sectors at the end of the partition that cannot be accessed by the operating system using any typical means. This is referred to as *partition slack* (see Fig. 1, item 4) and is another place where data can be hidden.

Every partition contains a boot sector, even if that partition is not bootable. The *boot sectors in non-bootable partitions* (see Fig. 1, item 5) are available to hide data.

Any space in a partition not currently allocated (i.e., *unallocated space*) to a particular file (see Fig. 1, item 6) cannot be accessed by the operating system. Until that space has been allocated to a file, it could contain hidden data.

It is possible to manipulate the file system metadata that identifies bad blocks (e.g., the File Allocation Table in a FAT file system or $BadClus in NTFS) so that usable blocks are marked as bad and therefore will no longer be accessed by the operating system. Such *metadata manipulation* (see Fig. 1, item 7) will produce blocks that can store hidden data.

*Disk slack* (see Fig. 1, item 8) is a byproduct of a strategy to accelerate file management. Modern operating systems write data in complete 'blocks' where a block could be a sector (the minimal addressable unit of a disk) or a cluster (same concept as block in Microsoft's terms). If a file is not an exact multiple of the sector size, the operating system must pad the last sector and, in some cases (with older operating systems), this padding is data from memory (hence the historical term '*RAM slack*'). Modern operating systems tend to pad this area with nulls. If the total amount of data written does not fill an entire block, the remainder of the block from the sector boundary of the last sector within the block actually used by the file to the actual end of the block will remain unused and will likely contain data from a previously deleted file (*file slack*). It may also be effectively used to hide ephemeral data.

All of the above applies to nearly every file system in common use today, including FAT/FAT32, NTFS, and Linux Ext-based file systems. There are some potential data hiding places in Linux file systems that require a more detailed description.

## 2.4 ExtX

Ext2 and Ext3 (ExtX) file systems are divided into sections called block groups. Block groups are used to store file names, metadata, and file content. Information about block group size and configuration is stored in a superblock at the beginning of the file system, copies of which are scattered throughout the partition. The block following the superblock (if present) or the first block in every group (if not present) contains a group descriptor table with group descriptors describing the layout of each block group.

An ExtX superblock has 1,024 bytes allocated to it and the last 788 bytes are unused. There also might be some reserved area behind the superblock, depending upon the block size. We call this *superblock slack* (see Fig. 1, item 9).

There is a reserved area behind the ExtX group descriptor since the group descriptor is only 32 bytes long and the block bitmap that follows it must start on a block boundary. This means there is a minimum of 992 bytes (1,006 if you count the padding at the end of the group descriptor) where data could be hidden and more if the block size is larger than 1,024 bytes. We refer to this as *ExtX group descriptor slack* (see Fig. 1, item 10).

ExtX directories are like any other file and are allocated in blocks. The space between the last directory entry and the end of the block is unused and can be used to hide data. This is *directory slack* (see Fig. 1, item 11).

Figure 2 is a graphical illustration of the relative volatility of the various data hiding areas discussed above (cf. also [10]). The degree of persistence of the hidden data is dependent upon the characteristics of the particular area where it is hiding and the type of disk activity that has occurred since the data was written there. Figure 2 illustrates the relative persistence through normal disk activity and possible re-partitioning and/or re-formatting. The numbers in the figure correspond to the numbered items in Fig. 1.

## 2.5 NTFS

NTFS file systems also offer some unique opportunities for data hiding. The NTFS file systems used today contain innovations that provide efficient file access (for instance, B-Tree organization of files within directories) and readily accessible metadata files to manage disk organization (Microsoft's version of resource forks called Alternate Data Streams), and some other small file storage oddities as well.

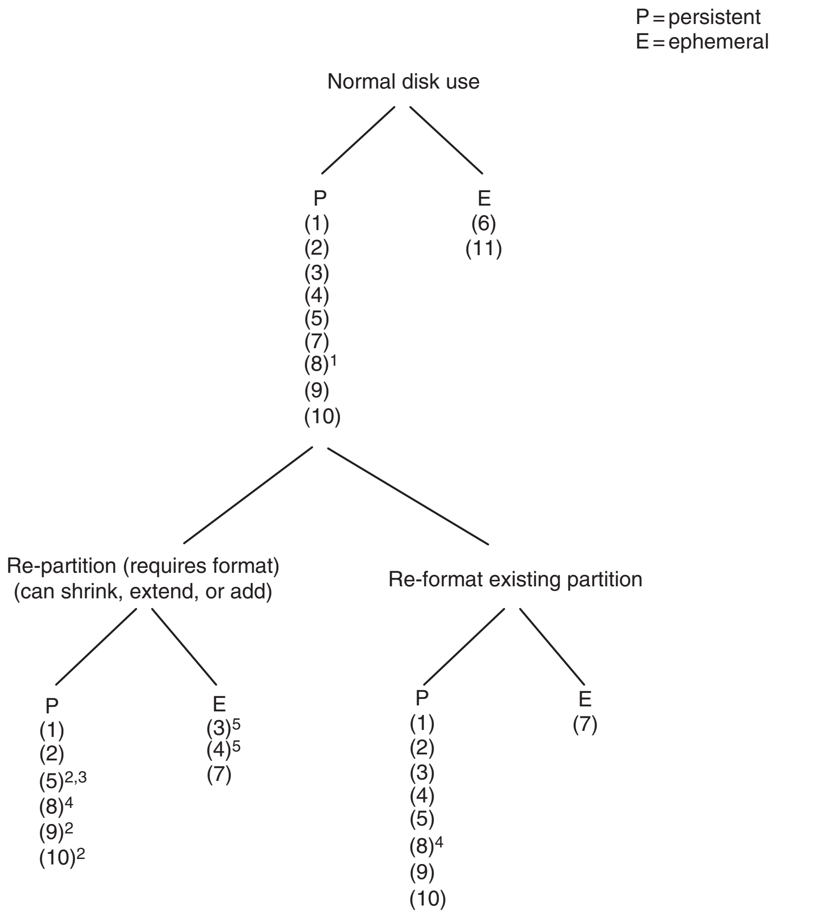

P = persistent
E = ephemeral
Normal disk use
P
(1)
(2)
(3)
(4)
(5)
(7)
(8)1
(9)
(10)
E
(6)
(11)
Re-partition (requires format)
(can shrink, extend, or add)
P
(1)
(2)
(5)2,3
(8)4
(9)2
(10)2
E
(3)5
(4)5
(7)
Re-format existing partition
P
(1)
(2)
(3)
(4)
(5)
(8)4
(9)
(10)
E
(7)
1 if file is not changed
2 if partition is not moved
3 if partition is not made bootable
4 becomes ephemeral with normal disk use
5 becomes ephemeral with normal disk use if new partition and file system extends into this space

FIG. 2. Relative volatility of data hiding areas.

When seeking to hide data, there are various strategies that might be employed. As mentioned in a more general sense, metadata manipulation may be used to conceal covert data in bad clusters ($BadClus). In fact, this same concept can be extended easily on an NTFS file system by working directly with the $Bitmap file.

The *$Bitmap file* contains a complete map marking the allocation status of every addressable cluster in the partition. Should a consistency check be run, it would become obvious should someone modify this table to hide data, but otherwise this provides a wonderful avenue for hiding data in a way that allows the data to persist for the life of the file system. Depending upon the purpose, these are far better approaches than using file slack which persists only for the life of the file.

NTFS provides some other nooks and crannies. For instance, *Alternate Data Streams* are actually additional $FILE entries associated with a parent file record within the Master File Table. We could not find Microsoft documentation regarding the number of alternate data streams that may be associated with a file or folder, but empirical testing conducted by Jason Fossen (www.fossen.net) suggests that the maximum is 4 106 regardless of the size of the ADSs themselves. These streams are not identified by typical file management tools (e.g., Windows Explorer), and so are hidden at that level. However, several utilities are available that report and manipulate alternate data streams [1]. Alternate data streams persist for the life of the attached file or folder as long as that file or folder remains in an NTFS file structure.

Small files also offer some interesting possibilities, especially when considered in conjunction with alternate data streams. NTFS has a rather unusual capability in that if a file is so small that the entire content of the file can fit within the boundaries of the Master File Table entry for the file, NTFS will store the file there. This is, of course, a convenient place to hide data. But it becomes even more interesting when a file is deleted. When a file is deleted, the clusters that were in use are released to the file system for reallocation. This includes the MFT entry itself. What if one were to create several thousand files, thus consuming several thousand MFT entries; once these files were created, a final file could be created that fits entirely within the MFT. All of these files are now deleted. In essence, we have created a piece of hidden disk space that resides within an allocated file (*$MFT and $MFTMirror*) that will persist until enough files are created to reuse that particular MFT entry. For further detail, see [2].

## 3. Forensic Implications

### 3.1 Fat16

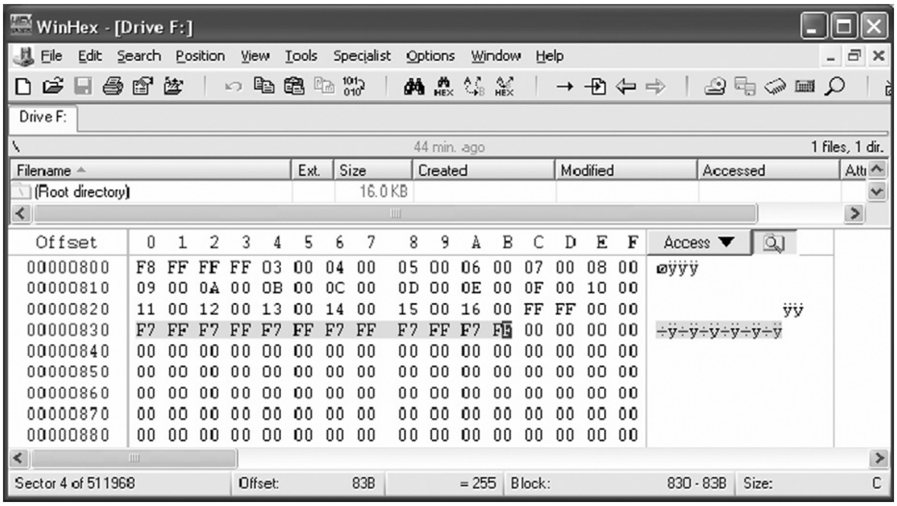
The implications of this type of intentional data hiding can be serious in the context of forensic analysis [4], [11], and [12]. Typically, forensic analysis of systems reveals that bad actors do not often take any extraordinary means to hide data beyond, perhaps, either wiping the media or encrypting the data. In fact, in most cases, the data is not even encrypted or wiped. What impact would a deep knowledge of the on-disk structures coupled with a desire to hide data have on an analysis?

To consider this, we have created several sample disk images and have embedded two pieces of data using the methods discussed in this chapter. After embedding the data, we used a tool that is commonly used by law enforcement for forensic analysis to determine how easily an analyst could uncover this hidden data.

The data that was hidden was a text string, ‘Hidden Message,’ and a GIF image containing an image of the word ‘Hidden.’ Please understand, of course, that if the investigator already knows something of what he is looking for, for example, the word ‘Hidden,’ the difficulty of recovering this data is trivial. What we are seeking to measure is whether or not the tool alerts the analyst to the presence of data in unusual locations and how likely it is that the analyst would discover the materials.

The first test was run using FAT16. As a form of control, an image file was copied to the disk using the typical file system tools. Data was hidden using two simple methods: First, the FAT16, when formatted, automatically preferred a cluster size of 4 096 bytes. The result of this was that the boot sector of 512 bytes is immediately followed by three empty sectors. Our ‘Hidden Message’ text was inserted here. In order to conceal the presence of the GIF image, the first FAT was modified so that clusters 24 through 29 were marked as bad (see Fig. 3). Finally, the GIF image was placed onto the disk beginning in cluster 24.

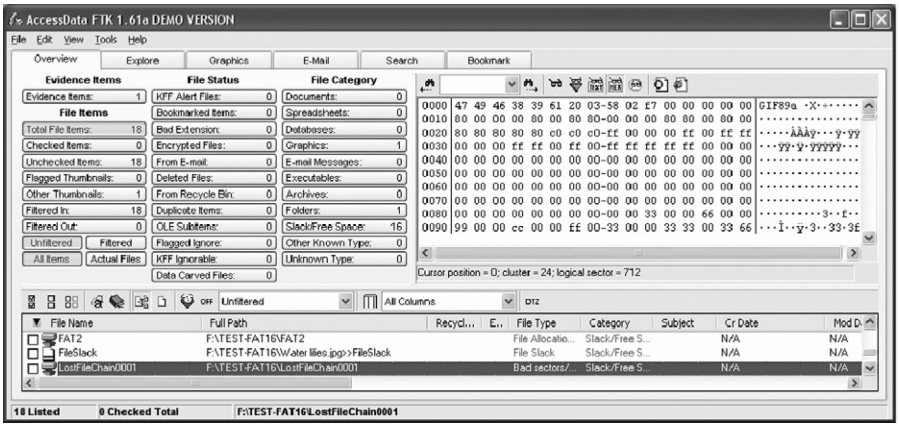
With our data hidden, we used AccessData’s Forensic Toolkit (better known as FTK) to acquire an image of the drive and perform data carving and full text indexing (see Fig. 4).


WinHex - [Drive F:]
File Edit Search Position View Tools Specialist Options Window Help
Drive F:
44 min. ago
1 files, 1 dir.
Access
Sector 4 of 511968 | Offset: 83B | = 255 | Block: 830 - 83B | Size: C

| Filename | Ext. | Size | Created | Modified | Accessed | Attr |
|---|---|---|---|---|---|---|
| (Root directory) |  | 16.0 KB |  |  |  |  |

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000800  F8 FF FF FF 03 00 04 00  05 00 06 00 07 00 08 00   øÿÿÿ
00000810  09 00 0A 00 0B 00 0C 00  0D 00 0E 00 0F 00 10 00
00000820  11 00 12 00 13 00 14 00  15 00 16 00 FF FF 00 00               ÿÿ
00000830  F7 FF F7 FF F7 FF F7 FF  F7 FF F7 FF 00 00 00 00   ÷ÿ÷ÿ÷ÿ÷ÿ÷ÿ÷ÿ
00000840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000860  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000870  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000880  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

FIG. 3. File allocation table with clusters 24–29 marked ‘bad.’



| Evidence Items | | File Status | | File Category | |
|---|---|---|---|---|---|
| Evidence Items: | 1 | KFF Alert Files: | 0 | Documents: | 0 |
| **File Items** | | Bookmarked Items: | 0 | Spreadsheets: | 0 |
| Total File Items: | 18 | Bad Extension: | 0 | Databases: | 0 |
| Checked Items: | 0 | Encrypted Files: | 0 | Graphics: | 1 |
| Unchecked Items: | 18 | From E-mail: | 0 | E-mail Messages: | 0 |
| Flagged Thumbnails: | 0 | Deleted Files: | 0 | Executables: | 0 |
| Other Thumbnails: | 1 | From Recycle Bin: | 0 | Archives: | 0 |
| Filtered In: | 18 | Duplicate Items: | 0 | Folders: | 1 |
| Filtered Out: | 0 | OLE Subitems: | 0 | Slack/Free Space: | 16 |
| | | Flagged Ignore: | 0 | Other Known Type: | 0 |
| | | KFF Ignorable: | 0 | Unknown Type: | 0 |
| | | Data Carved Files: | 0 | | |

```
0000|47 49 46 38 39 61 20 03-58 02 f7 00 00 00 00 00|GIF89a ·X·÷·····
0010|80 00 00 00 80 00 80 80-00 00 00 80 80 00 80 00|················
0020|80 80 80 80 80 c0 c0 c0-ff 00 00 00 ff 00 ff ff|·····ÀÀÀÿ···ÿ·ÿÿ
0030|00 00 00 ff ff 00 ff 00-ff ff ff ff ff 00 00 00|···ÿÿ·ÿ·ÿÿÿÿÿ···
0040|00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00|················
0050|00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00|················
0060|00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00|················
0070|00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00|················
0080|00 00 00 00 00 00 00 00-00 00 33 00 00 66 00 00|··········3··f··
0090|99 00 00 cc 00 00 ff 00-33 00 00 33 33 00 33 66|···Ì··ÿ·3··33·3f
```

Cursor position = 0; cluster = 24; logical sector = 712

Unfiltered | All Columns

| File Name | Full Path | Recycl... | E.. | File Type | Category | Subject | Cr Date | Mod D. |
|---|---|---|---|---|---|---|---|---|
| FAT2 | F:\TEST-FAT16\FAT2 | | | File Allocatio... | Slack/Free S... | | N/A | N/A |
| FileSlack | F:\TEST-FAT16\Water lilies.jpg>>FileSlack | | | File Slack | Slack/Free S... | | N/A | N/A |
| LostFileChain0001 | F:\TEST-FAT16\LostFileChain0001 | | | Bad sectors/... | Slack/Free S... | | N/A | N/A |

**18 Listed** **0 Checked Total** **F:\TEST-FAT16\LostFileChain0001**

FIG. 4. Forensics toolkit results showing hidden GIF file.

Of particular interest to us were the results from the file carver. File carvers are forensic tools (or data recovery tools) that analyze the data on a drive (typically sector by sector or block by block) without any regard for the actual logical organization of the disk. Each sector or block is then checked for known file signatures and, if one is found, the data is extracted or marked appropriately. One of the reasons that a GIF image was selected is that it has a very standard and easily recognizable 'magic number' or fingerprint: Almost every one of them begins with the marker 'GIF89a.' While FTK did find the file and mark it as a 'Lost File Chain' marked as bad sectors, it did not successfully identify this as an image.

Even though the 'hidden message' is sitting in what should be a completely barren segment of the drive, FTK makes no special mention of this text. Unless the analyst is in the habit of examining all reserved sectors for additional data or already knows to search for a keyword of 'Hidden,' it is very unlikely that this data would be found. From our experience, this is typical behavior for a forensic toolkit. In short, it is unusual for any forensic analysis tool currently on the market to draw the analyst's attention to a piece of data that is residing in an atypical location.

## 3.2 NTFS

The next set of tests was performed using an NTFS file system with a cluster size of 4096 bytes. Data was hidden using several of the techniques discussed. First, in the reserved area between the primary partition table and the first partition, 'Hidden

Message' was again embedded. Since there was additional space, we also embedded our image file, but this time the file was not embedded beginning at the beginning of a sector. Many file carvers speed up their operation by examining only the first few bytes of a sector or only the first sector in a cluster.

As an additional test, the $Bitmap file was modified to mark the final clusters of the volume as 'In Use' and the GIF image was copied into these now reserved clusters. The 'Hidden Message' was also embedded using the Alternate Data Stream technique discussed.

To accomplish this, a host file was created followed by 1,000 associated streams. The data that we wished to conceal was then copied into what amounts to stream 1,001 and the host file was then deleted. Finally, additional files were copied onto the drive to obscure the original host file MFT entries, but with the cushion of 1,000 alternate data streams, the original hidden data would still be untouched.

With all of these tasks completed, the drive was imaged and analyzed using FTK. FTK was able to identify and carve the hidden data in the slack space between the partition table and the first partition, even though it was not on a sector boundary. Unfortunately, while it was successful with the image file, it still failed to 'notice' the hidden text. What if the file that was hidden did not match a known fingerprint or were encoded in some way? FTK would fail to notify the analyst. It was also successful in identifying the image file tucked away at the tail end of the disk in space that we 'reserved.' While this is good, there was no notification that space was marked 'In Use' that was not actually allocated to any disk structure or file. Obviously, the same problem exists as does for the gap between the partition table and the partition.

What about the alternate data streams? FTK, surprisingly, made no mention of the fact that there were still ~800 remnants of alternate data streams floating around in the MFT and, of course, did not notify us to the presence of the 'Hidden Message' unless we already knew what to search for.

Overall, this means that for us to be successful today with forensic investigations, we need really sharp analysts who are willing to go above and beyond to find data when we are dealing with clever malcontents. Our tests were performed on a small drive (256 MB). Obviously, hunting down data on a 500 GB drive is well-nigh impossible without good tools and good initial information. In the long run, as drives continue to increase in size and those who desire to conceal activities become better at their trade, the forensic community will be at a greater and greater disadvantage unless we become proactive in this 'Arms Race' and move away from strict signature-based analysis of media and start building in some sort of anomaly detection capability.

## 4. Perspectives

It should be pointed out that this discussion includes methods of hiding data only within the media and file system structure and does not address other data hiding methods such as:

- Altered BIOS parameters

  The system BIOS traditionally stores the physical configuration of the drives that are connected. Typically, the BIOS is set to automatically detect the drive parameters; however, it can be configured manually. By doing so, it is possible to create a sort of pseudo HPA area beyond what is apparently the end of the disk according to the BIOS.
- Registry entries using new keys or unused keys

  There are a variety of technologies available that allow one to create a virtual file system within a file on the system. With some minor modification, it is trivial to create such a file system that is stored, not within a file, but within a set of registry keys. Large disks would be prohibitively slow to access and have a serious impact on system performance due to memory utilization.
- Swap files

  While swap files are in fact files that are known to exist on a system, if the swap mechanism is disabled or otherwise controlled, the swap space can be used for the storage of arbitrary data. While it is true that swap space is frequently analyzed by forensic investigators, it is typically viewed as a memory dump. If a portion of the data is instead formatted as a virtual encrypted disk, it is highly likely that this data will go unnoticed, dismissed as random junk.
- Renamed files (e.g., as .dll)

  Decidedly nontechnical, this remains an extremely effective way to hide almost any kind of data. Malicious code authors have been using this same technique to great effect for many years, concealing the malicious code as what appears to be a system DLL. This will likely not fool a forensic investigator, but a casual user or administrator will often overlook files masked in this way.
- Binding of one executable file to another

  Another very popular technique with malicious code authors is to redirect execution from one application to another. In some ways, this is exactly how appending viruses of the early nineties functioned in that the malicious code was appended to the existing executable and then the executable header was modified or the initial instruction modified to redirect execution to the malicious code. Following execution of the malicious code, control was handed back to the original code. These days redirection is accomplished in a variety of ways, for instance through DLL injection, which could then allow a user to subvert the typical input/output system for file and data access.

- Steganography

  This well-known technique for hiding data in plain sight is fast gaining in popularity. One of the most interesting applications of this is the StegFS driver for Linux (http://www.mcdonald.org.uk/StegFS/) which allows the user to create layers of hidden file systems on top of an existing Linux file system.
- Hiding data within documents (e.g., as metadata or using a white font)
- Hiding data within html files
- Merging Microsoft Office documents

  This is another steganographic technique. While some recent versions of software have made this more difficult (for instance, newer versions of Microsoft Word will actually display the white on white text as a grayed out text), the hiding of data in metadata or through other techniques is still possible. For instance, one tool embeds data by manipulating the white space at the end of existing lines within a document.
- Encrypted files
- Compressed files

  These two are perhaps the most direct way to store data on a system in a somewhat unreadable format, though some may contend with how well hidden the data actually is. From the point of view of someone who is looking for ‘unusual’ data, a compressed or encrypted file would be of interest. From the point of view of someone who is trying to run a file carver or a string match against a set or large disks, it is quite likely that these methods will be sufficient to hide the data from an investigator. For the compression, simply using a common compression tool would likely not be enough to day. Many tools are smart enough to take these files apart or alert the operator that a password is required. Using a older technology like LHARC could prove quite sufficient, though. This format (and others) could still be recognized, but the investigator’s tools are far less likely to understand how to decompress the data.

## 5. Conclusion

Knowing how data can be hidden within the media and file system structure means knowing how that data can be found. Note that we are talking about only the data hiding mechanisms that have been discussed here; methods such as encryption and steganography present their own sets of analysis problems.

Many of those who attempt to temporarily hide data or use a computer with nefarious intent are aware that there are ways in which this hidden data can be found. They will therefore use one of the available 'disk wiper' utilities in an attempt to eliminate the evidence. What they are not aware of is that many of these utilities are ineffective in eliminating all hidden data. See [2] for more information.

A forensic analysis tool can be as simple as a hex-editor. While effective, this approach is very labor intensive given the size of today's hard drives. There are many commercial and open source digital forensic tools available that will automate this process. However, one must exercise caution when using these tools, since not all of them are equally effective in finding all types of hidden data. It is often advisable to use more than one of these tools to perform a digital forensic investigation. For further information, see [5], [7], and [8].

Finding hidden data is further complicated by the fact that there are a number of antiforensic tools being developed. One source is the research being done by the Metasploit Project (www.metasploit.com/projects/antiforensics). An example is their Slacker tool that automatically encrypts and hides a set of data within the slack space of multiple files that are known to be unlikely to be changed. Data thus hidden cannot be detected by currently available digital forensic analysis software.

Data hiding becomes an increasingly important topic as the sophistication of information technologists on both sides of the law increases.

## References

[1] Berghel H., and Brajkovska N., 2004. Wading through alternate data streams. *Communications of the ACM*, **47**(4): 21–27.

[2] Berghel H., and Hoelzer D., What does a disk wiper wipe when a disk wiper does wipe disks. *Communications of the ACM*, **49**(8): 17–21.

[3] Berghel H., and O'Gorman L., 1996. Protecting ownership rights through digital watermarks. *IEEE Computer*, **29**(7): 101–103.

[4] Caloyannides M. A., 2001. Computer Forensics and Privacy. Artech House, Norwood, MA.

[5] Carrier B., and Winter B., 2003. Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of Digital Evidence*, **1**(4): 1–12.

[6] Carrier B., 2005. File System Forensic Analysis. Addison-Wesley, Upper Saddle River, NJ.

[7] Carvey H., 2005. Windows Forensics and Incident Recovery. Addison-Wesley, Upper Saddle River, NJ.
[8] Casey E., 2002. Handbook of Computer Crime Investigation. Academic Press, San Diego, CA.
[9] Cole E., 2003. Hiding in Plain Sight: Steganography and the Art of Covert Communication. Wiley Publishing, Indianapolis.
[10] Farmer D., and Venema W., 2005. Forensic Discovery. Addison-Wesley, Upper Saddle River, NJ.
[11] Kruse I. I., Warren G., and Heiser J. G., 2002. Computer Forensics, Incident Response Essentials. Addison-Wesley, Upper Saddle River, NJ.
[12] Nelson B., et al., 2006. Guide to Computer Forensics and Investigations. 2nd edition Course Technology, Florence, KY.
[13] Singh S., 2000. The Code Book. Anchor Books, New York.
[14] Wang H., and Wang S., Cyber warfare: steganography vs. steganalysis. *Communications of the ACM*, **47**(10): 76–82.