

# Hide and Seek: Concealing and Recovering Hard Disk Data

James Madison University Infosec Techreport  
Department of Computer Science

JMU-INFOSEC-TR-2006-002

Steven Dillon

July 2006

# Hide and Seek: Concealing and Recovering Hard Disk Data

Steven Dillon  
CS633 Computer Forensics  
Term Paper  
Professor: Florian Buchholz  
James Madison University  
757-555-1212  
dillonsw@jmu.edu

## ABSTRACT

A good forensics examiner should be aware of the techniques that an attacker, intruder, malicious program, virus, or worm is capable of utilizing. Indeed, a significant body of knowledge will be required to track down the steps taken by an adversary, piece together the puzzle of events, and successfully defend conclusions in a court of law. While not every incident investigated will be found to have been carried out by someone with the skill of an expert, automated tools are becoming more sophisticated on a daily basis and can give even a script kiddie power and capabilities that previously did not exist.

During the commission of a crime it is quite natural, and a common human trait, to attempt to cover up, disguise, or otherwise hide evidence of criminal activity. In some cases, hiding data is part of the master plan and helps ensure the success rate of the illicit activities. Deleting data from a hard disk can be used as a hiding technique as well as an attempt to conceal evidence.

To be successful, forensics investigators must know and understand these methods. Additionally, they must possess the ability to recover as much case related data as possible; discovering and reclaiming hidden and deleted data.

The focal points of this paper include: physical characteristics of a hard drive; distinguishing attributes for the FAT32, NTFS, and Ext2fs file systems; techniques used to both hide and find digital data; software tools available for deleting and recovering data; true data deletion; applicable regulations regarding data destruction; and data recovery capabilities. The subject areas are intended to provide background to the neophyte forensics practitioner or those with an interest in the field.

## 1. INTRODUCTION

To understand how differences in file systems influence hard disk data hiding, deleting, and recovery, an overview of the physical characteristics of a hard drive is essential. Especially important is the use of magnetization, the composition of the materials used on modern disk platters, and disk layout as it relates to the organization of tracks and sectors. Disk formatting as well as the mechanism used to read and write data to the media is also meaningful background information. These data points will all be required to fully grasp concepts and techniques used during investigations that a forensics examiner may be called upon to perform.

While areas of commonality exist between hiding and recovering data, individual file systems have peculiarities that affect

operating system and application behaviors. These differences bring about variances in attainable results both for the miscreant and the investigator. This paper will focus on the FAT32, NTFS, and Ext2fs file systems (arguably the most common file systems in use today) as they relate to hiding, deleting, and recovering data. File system specifics and their structures will be used to provide additional background information as well as assistance in the complete understanding of the subject area.

With the fundamentals sufficiently covered, the various methods of hiding data can be introduced. Slack space provides a nearly omnipresent and easily exploited area for concealing data. There are different types of slack space, and, as previously alluded to, slack space is affected by file system particulars. As such, slack space will be considered for the FAT32, NTFS, and Ext2fs file systems. Additionally, the results of a slack space experiment performed on both the NTFS and Ext2fs file systems are presented. Some data hiding techniques are specific to a particular file system, such as Alternate Data Streams (a capability that some system administrators, especially those who carry the responsibility of keeping their computers protected against intruders, may reasonably label a nuisance). The origin of this functionality is discussed and examples of hiding data and executing programs from within an Alternate Data Stream are provided. Other methods concerning data concealment are considered as well. For example, disguising files, making use of hidden attributes, and using deletion to hide data. Finally, the importance of these techniques are considered from the forensics examiner's perspective.

It is likely that most paranoid computer users have at least heard rumors concerning the disposition of their deleted data. However, it is just as unlikely that the majority of computer users (including professionals) understand the real, true story that takes place behind the scenes of a delete button click. By itself, that makes data deletion an interesting subject and one that requires an explanation as to what has caused the persistence of this problematic area of computing. Once again, differences can be found between the FAT32, NTFS, and Ext2fs file systems concerning their deletion mechanisms. The concept of disk wiping and deletion is then introduced before proceeding to platform specific tools (and examples) for both the Windows and Linux operating systems. To conclude this section, relevance for the forensics examiner is, once again, considered and special attention is paid to the topic of drive slack space.

Complete data removal may very well be impossible without physically destroying the disk platter(s) on which it was originally written. The question of true deletion along with the recoverability controversy, reasons why deletion is such a

difficult beast to tame, and low-level details concerning disk coding and channel information (useful in gaining a more complete understanding of the problem space) are then covered. The effectiveness of potential destruction methods (e.g. breaking a disk platter into pieces with a hammer and subjecting a hard drive to a degausser) are also discussed. Finally, legal requirements and government standards are considered.

The forensics examiner's ability to recover hidden and deleted data is very important. As a counterpoint to the methods used to hide data, techniques that can be utilized to discover and reclaim data will be discussed. The most popular tools (by way of references and recommendations found during this paper's research phase) for both the Windows and Linux platforms are given. This information includes highlighted capabilities, file systems support, and other relevant, potential decision making, considerations. Where software tools are unable to extract deleted data, radical and/or exotic techniques can be employed. The use of such tools would most likely only be considered in cases of extreme importance, but they offer a glimpse into the reality that nearly everything you do on a computer is traceable.

## 2. HARD DISK DESCRIPTION

### 2.1 Physical Characteristics

Common hard disks are inherently magnetic. Through the use of magnetization, a specific pattern can be created to hold a single bit of information; the most basic unit of data. The single bit value represents either a zero or one. To form data, bits are grouped into bytes or characters. The size of the bytes will vary depending on the computer and the device, but are commonly eight bits long. The use of the term character and byte are often used interchangeably while the octet label is a more precise description of the actual number of bits. For the purposes of this paper, a byte and a character will be considered equivalent and consist of 8 bits (i.e. an octet). The capacity of a hard disk is the number of bytes capable of being stored on it, which is typically in the Gigabyte range for most home computers sold today [13].

The first magnetic media (known as "particulate media") consisted of a platter shaped in the form of a slim circular disk that was constructed from a non-magnetic material such as glass or aluminum. The platter was then coated with a thin layer of magnetic material, usually iron (III) oxide (the magnetic medium) and aluminum oxide (for abrasive resistance). Modern magnetic media is called "thin-film media" (see Figure 1) and consists of a metal substrate with thin layers of materials essentially grafted on top of it. The layers form what could be described as a sandwich and are deposited on the metal disk by physical vapor deposition (atoms of different materials are formed on the surface using a chemical evaporation technique). A thin carbon overcoat increases mechanical durability and slows corrosion of the magnetic layer beneath it while a final layer of lubricant is used to minimize the wear of the carbon layer [19, 23].

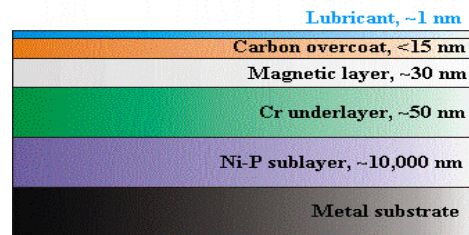


Figure 1. Thin-film media layers [23].

A single-sided disk is one in which information is only stored on one of the surfaces while a double-sided disk utilizes both surfaces. Multiple disks can be assembled into a stack known as a disk pack to increase the storage capacity of a single device. The information on a disk surface is stored in concentric circles known as tracks (that have a distinct diameter). In devices that contain a disk pack, tracks with identical diameters across the surfaces form a cylinder. A disk surface can hold between hundreds and thousands of tracks. Adjacent tracks are separated by gaps to prevent or minimize errors due to misalignment or magnetic field interference. To simplify the electronics required, the same number of bits are stored on each track. As such, the density, in bits per linear inch, increases when moving from the outermost track to the innermost track (the same technique used for a phonograph record). Due to the large amount of potential information that can be stored in a track, it is separated into smaller blocks called sectors (see Figure 2). The sectors, formed by the process of dividing a track, are hard-coded on the disk surface and unalterable.

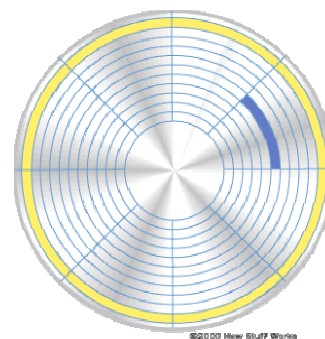


Figure 2. Example track (yellow) and sector (blue) [2].

Several methods exist for the organization of sectors. For example, the use of an angle or arc from the center of the disk as it extends outward through tracks. Similarly, the use of smaller angles at the center moving towards the disk edge to maintain a uniform recording density can be used. Another technique known as Zone Bit Recording facilitates the same number of sectors per arc for a range of cylinders.

Disk formatting is performed by the operating system and is the process of dividing a track into equal-sized disk blocks often referred to as pages. Once established, the block sizes can not be dynamically modified. Block sizes typically range from 512 to 4,096 bytes. The aforementioned hard-coded sectors will frequently further subdivide the sectors into blocks during the formatting procedure. The blocks are separated by fixed-size interblock gaps that contain specially coded control information

(established during disk initialization) and are used to resolve block layout attributes.

## 2.2 Data Access and Transfer

A disk is a random access addressable device. Data transfer between the disk and main memory is done in disk block units. The physical hardware address of a block is a combination of the cylinder number, track number (surface number within the cylinder where the track is located), and block number (within the track). A single number referred to as the LBA (Logical Block Address) in most modern systems is given to the disk I/O hardware. The LBA is then mapped to the correct block by the disk drive controller. Some devices transfer several contiguous blocks, known as a cluster, on or off the disk as a unit.

The mechanism that reads from and writes to the disk is the read/write head, which is a conducting coil. The head's ability to write is based on the fact that electricity flowing through a coil produces a magnetic field. Electric pulses are transferred to the head causing magnetic patterns to be recorded on the surface below it. The patterns are different for positive and negative currents. The facility to read is based on the fact that a magnetic field moving relative to a coil produces an electrical current in the coil. When the disk surface is passed under the head, a current is generated of the same polarity that recorded it.

Traditionally, a fixed air gap existed between the head and the disk platter since the head was positioned at a stationary distance above the disk platter. In order for the head to read and write properly, it must have the capacity to generate or sense an electromagnetic field of sufficient magnitude. The narrower the head is, the closer it must be to the disk surface to operate correctly. By narrowing the head, creation of narrower tracks is possible, which increases the data density (a desirable quality). Unfortunately, bringing the head closer to the platter increases the risk of error from impurities or imperfections. To overcome this difficulty, the Winchester disk was developed. The Winchester heads are used in sealed drives that are nearly free of contaminants. These heads are designed to operate closer to the platter's surface than rigid disk heads. In actuality, the head is an aerodynamic foil that rests lightly on the disk's surface while the disk is motionless and is lifted above the surface by air pressure generated from a spinning disk.

The read/write head is attached to a mechanical arm that is connected to an actuator that can be moved (in unison where multiple arms exist) to precisely the right location over the track specified by the block address. The disk or disk pack is continuously spun by a motor at a constant speed (normally between 5,400 and 15,000 rpm). When the read/write head has been moved to the right track and the desired block is under it, the electronic component of the read/write head is activated. Disk units that utilize such a design are known as movable-head disks (see Figure 3). Fixed-head disks consist of fixed read/write heads and have as many heads as there are tracks. The fixed-head configuration uses electronic switching to select the appropriate head, which makes the system much faster than a movable-head disk unit [13, 32].

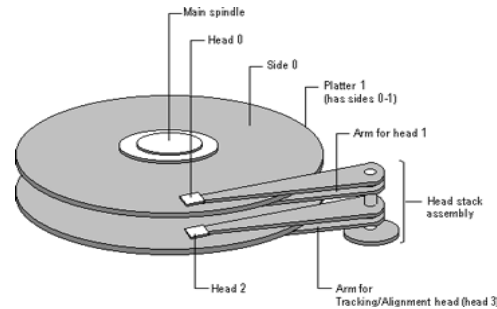


Figure 3. Movable-head disk pack [16].

A disk drive typically contains an embedded disk controller that interfaces with the computer and controls the drive. The controller takes high-level I/O commands, positions the arm, and causes the read/write action to be performed. The controller can be accessed over one of a number of bus types including: ATA (IDE, EIDE), Serial ATA, SCSI, SAS, FireWire (aka IEEE 1394), USB, and Fibre Channel [13, 19].

The time required to position the read/write head to the correct track is called the seek time (normally between 3 and 10 milliseconds). An additional delay called rotational delay or latency is incurred while the desired block spins into place under the read/write head. The faster the rotation, the less latency there will be. Transferring the data to and from the device is referred to as the block transfer time. Therefore, the total amount of time required to locate and transfer an arbitrary block, given its address, is the sum of the seek time, rotational delay, and block transfer time. Modern workstation and server hard drives can achieve transfer speeds of up to 100 MB/second [13, 32].

## 3. FILE SYSTEMS

To achieve a more fundamental understanding of hiding and recovering data, an examination of the FAT32, NTFS, and Ext2fs file systems is necessary.

### 3.1 FAT32 File System

The FAT32 file system was introduced in the Microsoft Windows 95 OEM Service Release 2 to address the FAT16 limitation of no more than a two gigabyte volume size. The FAT32 BPB (BIOS Parameter Block) exactly matches the BPB used in both the FAT16 and FAT12 file systems for maximum compatibility and to ensure that all FAT file system drivers understand and properly support the volume. The BPB structure contains the BPB\_BytsPerSec, BPB\_SecPerClus, BPB\_RsvdSecCnt, BPB\_SecPerTrk, BPB\_HiddSec, and BPB\_TotSec32 values (i.e. the most pertinent data to this discussion).

The BPB\_BytsPerSec field holds the count of bytes per sector. The value can only be one of the following values: 512, 1024, 2048, or 4096. For maximum compatibility, only the 512 value should be used (especially considering the amount of FAT code that is "hard wired" to that size and fails to check the field to determine what the value actually is). However, the Microsoft operating systems will properly support the 1024, 2048, and 4096 values.

The BPB\_SecPerClus field holds the number of sectors per allocation unit. The value must be a power of two that is greater

than zero. The legal values are 1, 2, 4, 8, 16, 32, 64, and 128. However, it should be noted that a warning concerning a “bytes per cluster” value that exceeds 32K is documented (apparently, cluster sizes greater than 32K do not function properly and many applications will not work on a FAT volume with such a configuration).

The BPB\_RsvdSecCnt field holds the number of reserved sectors in the reserved region of the volume starting at the first sector of the volume. For FAT32 volumes, the value is typically 32. Once again, the “hard wired” FAT code statement pertains here due to systems that set the value to one and don’t bother to inspect the field’s value.

The BPB\_SecPerTrk field holds the sectors per track and is only relevant for media that have a geometry. That is, the volume has been broken down into tracks by multiple heads and cylinders that are visible on interrupt 0x13.

The BPB\_HiddSec field holds the count of hidden sectors that precede the partition that contains this FAT volume. The field is only relevant for media on interrupt 0x13 and is always zero unless the media has been partitioned. The appropriate value for the field is operating system specific.

Finally, the BPB\_TotSec32 field holds the total count of sectors on the volume. The count includes the count of all sectors in all four regions of the volume. For FAT32 volumes, the value must be non-zero.

The FAT data structure is important as well since it defines a singly linked list of the clusters of a file. It can be considered a regular file that has a special attribute indicating that it is actually a directory. More precisely, it is a directory whose contents are a series of 32 byte FAT directory entries. Essentially, the FAT is a mapping of the data region of a volume by cluster numbers. The FAT type (32 in this case) is determined solely by the count of clusters on the volume; that is, there is no such thing as a FAT32 volume that has less than 65,525 clusters. Additionally, not all of the 32 bits in a FAT32 FAT entry are actually used. The high 4 bits are reserved and only modified when the volume is formatted and the entire 32-bit FAT entry is zeroed.

Data is associated with a file by utilizing the cluster number of the first cluster of the recorded file. The cluster location in the FAT contains either an EOC (End Of Clusterchain) mark or the cluster number of the next cluster of the file. The cluster number whose cluster entry in the FAT contains the EOC mark is allocated to the file as the final cluster [27].

A cluster (also known as an allocation unit) is the smallest amount of disk space that can be allocated to hold a file. Cluster size is determined by the number of sectors that the cluster contains. If 512-byte sectors are used, then a 512-byte cluster only contains a single sector. When a cluster consists of 4,096 bytes, then eight sectors must be utilized. The cluster size is dependent on the size of the volume. For example, a 4KB cluster size can support up to an 8GB volume, and a volume size of 32GB requires a 16KB cluster size. The smaller the cluster size, the more efficient disk storage is since the amount of existing space that cannot be used by other files is reduced [34].

The maximum possible number of clusters on a FAT32 volume is 268,435,445. If the 32KB per cluster is used for the FAT, then a

maximum disk size of approximately 8TB is possible. However, it is not possible to format a volume larger than 32GB using FAT32 and attempting to do so will result in the “Logical Disk Manager: Volume size too big” error [22].

## 3.2 NTFS File System

The NTFS offers several features that are exclusive to NTFS volumes and represent significant advances over the FAT32 file system. These characteristics include: file and folder permissions, encryption, disk quotas, file compression, mounted drives, distributed link tracking, sparse files, multiple data streams (an ability that will be discussed in more depth below), POSIX compliance, NTFS change journaling, and an indexing service [34].

The NTFS file system has both a BPB and an Extended BPB. The NTFS BPB shares some entries with both the FAT16 and FAT32 BPB. The Bytes Per Sector field contains the physical hardware sector size. For most disks created in the United States, this value will be 512. The Sectors Per Cluster field is the number of sectors in a cluster. A typical cluster size is 4,096 bytes (making the value of this field 8). The Reserved Sectors field will always be zero for NTFS due to the fact that the boot sector is always placed at the beginning of the partition. If a non-zero value is present, the NTFS will fail to mount the volume. The Total Sectors field has a field length of 8 bytes and represents the total number of sectors on the hard disk. The Clusters Per MFT Record is a single byte field that is used to maintain the size of each record. The NTFS creates a file record for all files and a folder record for all folders created on a volume. Files and folders smaller than this size are contained within the MFT (Master File Table) itself. If the number in this field is positive (up to 7F), then it represents the number of clusters per MFT record. However, if the number is negative (i.e. a value from 80 to FF), then the size of the file record is two raised to the absolute value of this number. Very similarly, the Clusters Per Index Buffer field is another single byte field that holds the size of each index buffer, which is used to allocate directory space. The same positive and negative value range is used here to determine the size.

The Master File Table is similar to the FAT in the FAT32 file system in that it stores information required to retrieve files from the NTFS partition. The MFT is created during the formatting of a volume and is a relational database that consists of rows of file records and columns of file attributes. The MFT contains at least one entry for every file on an NTFS volume (including the MFT itself). NTFS reserves the first 16 records (approximately 16KB) of the MFT for metadata files while the remaining records contain the file and folder records on the volume. A backup MFT (known as the MFT mirror) is also created and both MFT locations are recorded in the boot sector. The MFT is guaranteed exclusive use of 12.5 percent of the volume by default (this is a tunable value of 12.5, 25, 37.5, or 50 percent), which is known as the MFT zone. The space is not used to store data unless the remainder of the volume becomes full.

The default cluster sizes used by NTFS attempt to ensure a more efficient file organization structure. For example, a volume size of 7MB to 512MB requires a default cluster size of 512 bytes while a volume size of 2GB to 2TB only requires a 4KB cluster size (an eightfold increase in cluster size provides a volume that

can be up to 4,000 times larger; a significant improvement over FAT32). A Disk Management snap-in allows cluster sizes of up to 64KB to be specified during NTFS volume formatting. However, file compression is not supported on clusters that are greater than 4KB, which also happens to be the reason that the default value never exceeds the 4KB limit.

On an NTFS volume, all allocated sectors belong to a file. NTFS views files and folders as a set of attributes. Resident attributes are file attributes that can fit within the MFT file record for a given file. The file name and time stamp attributes are always resident. When the amount of information exceeds the amount that can be stored in an MFT file record, some attributes become nonresident and are allocated one or more additional clusters of disk space. A portion of the nonresident attribute set remains in the MFT and functions as pointers to the external cluster locations. Typically, files and folders that are 900 bytes or smaller are completely contained within the file's MFT record. Each file normally only requires a single file record, however, highly fragmented files or a file with a large number of attributes may require more than one file record. In this case, the first record of the file (known as the base file record), stores the location of the other required file records. Folders that are too large to fit within an MFT structure are organized as B-tree structures. The B-tree structures have records with pointers to external clusters containing the folder entries that could not be stored within the original MFT structure.

NTFS also provides a space saving mechanism known as a sparse file that can be utilized in cases where a file contains both meaningful data as well as large sections of data composed of zeros. The savings are realized through the allocation technique used. If an NTFS file is tagged as being sparse, then NTFS will only allocate clusters for the data explicitly specified by the application. Areas of the file that do not have specified ranges are represented by non-allocated space on the disk. When the sparse file is read back into the application from the allocated ranges, the data is returned as it was stored and any data read from non-allocated ranges is returned as zeros. A file that is a 1-GB sparse file, might only occupy 64KB of actual disk space. This capability is unique to NTFS and will be lost if the file is copied or moved to a FAT volume or to an NTFS volume mounted on an operating system without sparse file support [20].

### 3.3 Ext2fs File System

The Ext2fs is based on Extfs and was designed to be evolutionary yet still leave room for future improvements. It is the de facto Linux standard file system. Linux file systems implement a common set of Unix operating system concepts where files are represented by inodes, directories are simply files that act as containers for entries, and devices are accessed through I/O requests on special files. Files are represented by the inode structure, which contains a file description, file type, access rights, owner information, timestamps, size, and pointers to data blocks (addresses of data blocks that have been allocated to a file).

The Ext2fs supports standard Unix file types including regular files, directories, device special files, and symbolic links as well as adding additional functionality. For example, BSD or System V Release 4 semantics can be selected at mount time allowing an administrator to select file creation characteristics.

Ext2fs is also capable of handling a 4TB partition size, however, to accomplish this, Ext2fs must typically reserve 5% of the available blocks. Block sizes can be chosen when creating the file system and are typically 1,024, 2,048, or 4,096 bytes in size. To speed up link operations, Ext2fs uses a fast symbolic link that does not require any file system data block use; that is, the link information is stored in the inode itself. File system state is kept in a special field of the superblock where different mounting modes (e.g. read/write versus read-only) carry various states.

The Ext2fs was strongly influenced by the BSD file system and its constitution consists of block groups (which are analogous to BSD's FFS's cylinder groups). However, block groups are not tied to the physical layout of the blocks on disk. Each block group contains a redundant copy of crucial file system control information as well as a portion of the file system (e.g. a block bitmap, and inode bitmap, a piece of the inode table, and data blocks). In this way, Ext2fs can easily recover from the corruption of a superblock. When data is written to a file, up to eight adjacent blocks are preallocated upon the allocation of a new block, which improves write performance under heavy load (preallocation hit rates are approximately 75% even on very full file systems). Additionally, since contiguous blocks are allocated to files, future sequential reads are faster [4].

The Ext2fs superblock is the first block and it stores meta-information concerning the entire file system. The superblock structure also holds the following crucial values: `s_frag_size` (size of a fragment in bytes); `s_frags_per_block` (the number of fragments per block); `s_inodes_per_block` (the number of inodes per block); `s_frags_per_group` (the number of fragments in a group); `s_blocks_per_group` (the number of blocks in a group); and `s_inodes_per_group` (the number of inodes in a group).

Data blocks are addressed by 32 bit numbers starting with zero within each block group. The inode (which is addressed starting with the value one) can store 15 block addresses. The first 12 blocks (i.e. elements 0 – 11) of an inode store the addresses of the first 12 blocks in the file. If a file requires more than 12 blocks, then block 12 (known as an indirect block) will hold the addresses of the additional required blocks. Block 13 is called a double indirect block and block 14 is a triple indirect block (see Figure 4).

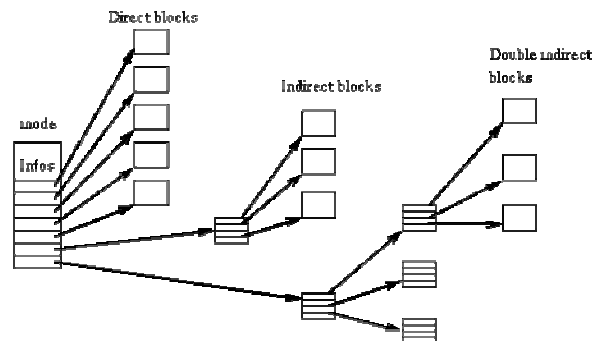


Figure 4. Inode with direct and indirect blocks [12].

## 4. HIDING DATA

### 4.1 Slack Space

There are several types of slack space that can be created on a hard disk. The space that exists between the end of a logical file and the end of a sector is known as RAM slack due to the fact that in some operating systems the space was filled with a dump of the currently available random access memory. The space that exists in a file between the sector that holds the end of the logical file and the final sector in a logical grouping (i.e. a cluster or a block) is called drive slack. The combination of drive slack and RAM slack is referred to as file slack. Finally, volume slack (also known as wasted space) is the portion of space that exists beyond the last addressable logical grouping and the end of the physical disk [5].

#### 4.1.1 FAT32 and NTFS Slack Space

As previously mentioned, all Microsoft operating systems read and write data in blocks known as clusters. The clusters are defined during volume formatting and have an even number of sectors (e.g. 8 sectors of 512 bytes each for a total of 4,096 bytes). On a system that has a default cluster size of 4,096 bytes, the smallest file size that can be created on disk is the size of a single cluster [21].

I performed a quick experiment on a Windows XP machine (which utilizes NTFS) by opening the Notepad application and creating a file with the words “Hello World” in it. I then saved the file as HelloWorld.txt and inspected it using both Windows Explorer and the Properties of the file itself. The Windows Explorer application reported the file’s size as 1KB, however, the true size on disk can be found through the file’s Properties GUI. The HelloWorld.txt file’s Properties showed that it actually only occupies 11 bytes, but takes up 4,096 bytes on disk (see Figure 5).

The reason for the size discrepancy is file slack space. Recall that the file slack space (4,085 bytes from my experiment above) is the total of the RAM slack space (501 bytes) and the disk slack space (3,584 bytes). The values can easily be seen with some simple math. The size of a sector on my disk drive is 512 bytes and the actual file content only occupies 11 bytes. Therefore, the RAM slack space is  $512 - 11 = 501$  bytes. The number of sectors per cluster, established during the formatting of my volume (an 80GB drive where the default value of 4KB clusters was used), is four. As such, one sector of 512 bytes has already been utilized leaving seven unused sectors ( $7 \text{ sectors} \times 512 \text{ bytes} = 3,584$  bytes). Thus the total file slack space is  $3,584 \text{ bytes} + 501 \text{ bytes} = 4,085$  bytes (see Figure 6).

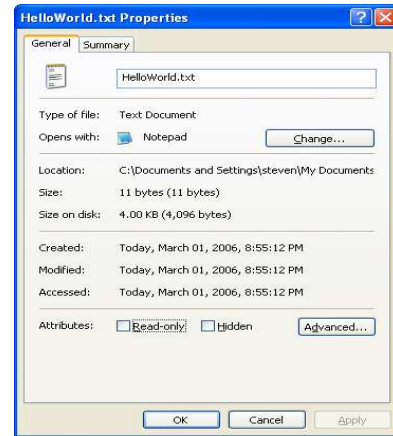


Figure 5. HelloWorld.txt Properties GUI

On a different Microsoft operating system that uses FAT32 where the sectors on the disk are 512 bytes and the number of sectors per cluster is 8, the exact same scenario would have played out. Even though the NTFS is more efficient and offers many improvements over the FAT32 file system, their slack space behavior can be identical. Modern home systems have volume sizes well over the FAT32 maximum supported formatting size of 32GB and usually come preloaded with a newer operating system. However, older systems that have large volumes will, in general, have much more slack space. For example, if the same experiment were conducted on a FAT32 system that required a cluster size of 32KB with a sector size of 512 bytes, then the disk slack space for the HelloWorld.txt file would be an amazing 63 sectors or 32,256 bytes.

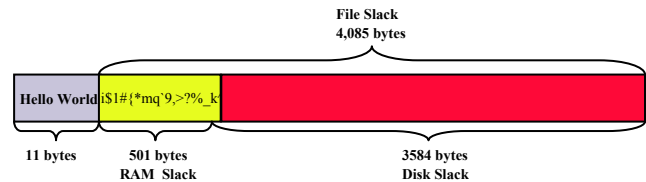


Figure 6. RAM and disk slack space (not to scale).

#### 4.1.2 Ext2fs Slack Space

The Ext2fs file system accesses addressable areas of a hard disk using blocks. As previously mentioned, most hard disks use a sector size of 512 bytes. A typical Ext2fs block will consist of two, four, or eight sectors, which means the block size will be 1,024, 2,048, or 4,096 bytes. Therefore the physical amount of space required by a file will always be a multiple of the block size. For example, on an Ext2fs that utilizes a 1,024 byte block size a file that only contains 4 bytes of actual information will nevertheless take up 1,024 bytes on the hard disk. If the content of the same file is then increased by 1,566 bytes (thereby increasing the file size to 1,570 bytes), the on disk size will increase to 2,048 bytes. The 478 bytes that exists between the end of file marker and the block boundary is slack space.

On my Fedora Core 4 system, the same slack space experiment was performed. I opened a HelloWorld.txt file using the vi editor and added the words “Hello World” then saved and closed the

file. A listing of the file using the ls command shows that it is 12 bytes in length (the 1 byte difference between this file and the Windows XP file is the End Of File marker that was inserted into the file). However, the disk usage tool du shows that it occupies 4KB and the stat utility confirms the information while providing a more detailed informational dump including the number of blocks, device, inode, and access information (see Figure 7).

```
[root@linus ResearchPaper]# ll
total 12
drwxr-xr-x  2 root root  4096 Mar  3 08:02 ./
drwxrwxrwx  6 root root  4096 Mar  3 08:02 ../
-rw-r--r--  1 root root   12 Mar  3 08:02 HelloWorld.txt
[root@linus ResearchPaper]# du -h HelloWorld.txt
4.0K HelloWorld.txt
[root@linus ResearchPaper]# stat HelloWorld.txt
File: HelloWorld.txt
Size: 12  Blocks: 8  IO Block: 4096  regular file
Device: fd00h/64768d  Inode: 36634668  Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)  Gid: (0/root)
Access: 2006-03-03 08:02:23.000000000 -0500
Modify: 2006-03-03 08:02:23.000000000 -0500
Change: 2006-03-03 08:02:23.000000000 -0500
```

Figure 7. Ext2fs HelloWorld.txt slack space.

The file slack space sizes from both the NTFS experiment and the Ext2fs test are nearly identical. In fact, they only differ by the single byte given to the HelloWorld.txt file produced on the Ext2fs-based system. Therefore, 4,084 bytes of slack space exists on the Ext2fs produced test. Though similar in size, the Linux RAM slack is zeroed before being written; the lack of zeroing is often cited as a security concern with Windows operating systems.

## 4.2 Alternate Data Streams

Alternate Data Streams (ADSs) are confined to the NTFS file system, however, due to Microsoft’s widespread dominance in both the work and home environments, a forensics examiner needs to be aware of this capability. The ADS functionality was added to the NTFS file system in the early 1990’s to allow an improved interoperability with Macintosh systems that utilize Resource Forks. The Macintosh Hierarchical File System (HFS) uses Resource Forks to store icons and other metadata associated with particular files [26]. Microsoft operating systems provide mechanisms for the creation of ADSs as well as an ability to execute code hidden within the streams while simultaneously warning of inconsistent support. These discrepancies are most prominent in the facilities provided for detection.

An ADS can be created by simply separating the default stream name from the ADS name with a colon (e.g. HelloWorld.txt:hidden\_stream). The ADS stream can be written to using an editor (such as NotePad) or a file redirection mechanism:

```
type input.txt > HelloWorld.txt:hidden_stream
```

An ADS can also be created and associated with a directory listing using the same methods:

```
echo Hide This Text > :hidden_stream
```

Any binary information capable of being streamed can be hidden in ADSs just as easily:

```
type c:\winnt\system32\sol.exe > HelloWorld.txt:hidden.exe
```

Additionally, the Properties GUI from the Windows Explorer can be used to manipulate fields and insert information. ADSs assume the access rights assigned to the default unnamed stream since they have no attributes of their own; that is, users without an ability to write a file will not be able to add anything to the ADS of that file.

The start command can be used to execute code within an ADS. However, this functionality is inconsistent as well and may require the full path to launch without error generation:

```
start c:\test\HelloWorld.txt:hidden.exe
```

Observation of the running executable may show a process named HelloWorld.txt with a valid PID. Hidden executables can also be launched through a shortcut, the start menu’s Run box, Perl, through utilization of the Run key in the Registry, and via the Windows Scripting Host (WSH) functionality [6].

## 4.3 Other Methods

### 4.3.1 Disguising Files

Sometimes hiding in plain site can be a useful trick employed by an attacker. For example, using a file name that is associated with operating system files (e.g. start.com, start.exe, or start.bat on a Windows OS). Such a file may initially be overlooked or assumed to be an essential system component. File name extensions are used to help users quickly identify specific file types. An attacker could easily rename the file extension in an attempt to disguise the true nature of a file. For example, a JPEG file extension of .jpg could be renamed to .doc or .txt [29].

### 4.3.2 Hidden Attributes

On Windows systems, a file can be hidden using the Properties GUI from within Windows Explorer as well as the attrib command, which tells the operating system to hide the file. While this will not keep a forensics examiner from finding the data, it can potentially provide a runtime cloak. On Unix systems, placing a dot in front of a file name or a directory gives a similar capability. In fact, classic hidden directories used by hackers on Unix systems are the “...” and “..” (dot-dot-space) directories.

### 4.3.3 Deleting Files

File deletion provides one of the most basic methods of hiding data. The specifics of deleting data will be covered in the sections below, however, it can be said that deletion is an effective hiding mechanism. Depending on several factors, enough time may exist between the period required for the retrieval of information (from the portion of the disk where it was deleted) and the reclamation by another file.

The Ext2fs, as well as all major Unix-like file systems, will keep a lock on resources used by a file descriptor until the process exits or the file descriptor is closed. As such, an attacker can use this ability to continue to write to a file even after calling the unlink function on the file. The file name will not appear on any listings from the file system while at the same time its data will be afforded overwrite protection [31].

## 4.4 Why a Forensics Examiner Cares

If the original file remains unchanged, then the data hidden in its associated slack space will be invulnerable to normal disk usage, invisible from the file system, and undetectable by file integrity checkers that utilize checksumming algorithms and MAC times. Slack space can be used to hide secrets and tools and plant evidence [8].

The ability to hide executable code such as VBS, EXE, CMD, and BAT files inside ADSs make viruses difficult to detect. Traditionally, the stance taken by virus detection scanners has been that real-time scanning will detect the code before it is executed since it must be loaded into memory. However, this approach does not take into account administrators that do not perform real-time scanning due to performance reasons. Therefore, scheduled virus scans will never detect the existence of an ADS. Additionally, writing data to an ADS will not change the timestamp of a file or its listed size [24].

For a forensics examiner to completely identify all possible information sources, awareness of hidden data is crucial. The correct cadre of tools will most assuredly assist in this effort; however, the value of knowledge and experience can not be overlooked (especially in cases where the trickery employed goes beyond the capabilities of the tools).

## 5. DELETING DATA

### 5.1 Why Deleted Files Aren't Removed

The operating system's file management system generally leaves data on the physical storage media when it deletes files.

Following a logical deletion of a file, the file management system updates file allocation entries to indicate that the space that was once reserved is now available. During this procedure, the file name contained within the directory structure is altered in a manner that can be recognized by the file management system and filtered out of the files shown to users; users no longer have access to the deleted files. Secure removal is an inherently slow operation and most users are not willing to wait for the overwriting process to complete. Deletion of large files (those with megabytes of data) would be intolerably slow unless the user was able to resume normal operations immediately.

There are several additional problematic areas associated with file deletion that the operating system would have to overcome to provide secure deletion. For example, file meta-data cannot be completely overwritten at the user level. User level programs simply overwrite the file data, but important file meta-data such as user and group ownership, the number of blocks the file contained, and the deletion time could not be removed. Another issue is that user-level secure deletion mechanisms would have to be wrapped with functionality that would first securely overwrite an existing file before proceeding with a normal file operation such as the copy command (in cases where a file is copied over an existing file). While tools that utilize dynamically linked libraries could be modified, statically linked binaries would remain a problem. File truncation is also a concern since applications that have truncation capabilities do not normally have enough information to know if blocks must be overwritten. Attempting to correct this would mean that all applications that can invoke file truncation would have to be modified. Finally, a user's disk quota

could allow a user to quickly allocate and securely delete large files while starving other users of disk resources if the quotas are not properly maintained during the period where data is being securely overwritten [1, 5].

### 5.2 FAT32 and NTFS Data Deletion

When a file is deleted from a FAT32 file system, the file allocation table entry for the first cluster assigned to the file is updated to indicate that the space reserved for the deleted file has been made available for future use. On NTFS, the File Record Header portion of the MFT contains a Flags field, which is used to maintain file state (e.g. deleted or in-use.) If the Flags field is set to one, then the file is in-use. The file deletion operation will set the Flags field to a value of zero. When a file is deleted on a FAT file system, the first letter of the filename is set to the sigma character ( $\sigma$ ), or, in hex, 0xE5. The data that makes up the file stored on disk is not altered and remains intact during the deletion operation on both NTFS and the FAT32 file systems. In this way, the data is never removed from the hard drive; its location is simply moved back to the pool of available disk space that can be written to as needed. In fact, depending on the disk cluster size versus the amount of data to be written, the data may never be overwritten. The importance of disk slack space can not be underestimated where data deletion is concerned. Since the data is never physically removed, it can still exist in disk slack space under the right conditions. For example, if the cluster size of a given disk was 16KB and the data in a cluster only 5KB, then the remainder of the cluster (11KB) will be disk slack space. The smaller cluster size available in NTFS helps reduce this problem [5, 25, 35].

### 5.3 Ext2fs Data Deletion

The file system's role is to provide an abstraction of the physical structure associated with the storage media. As previously mentioned, most hard drives sold in the United States consist of 512 byte sectors that are addressable from 0 to  $n-1$ . The organization of the sectors into files and directories used by applications through operating system facilities is the responsibility of the file system. Ext2fs collects sectors into blocks that are then organized into block groups. Block groups consist of a superblock, block allocation bitmap, inode allocation bitmap, inode table, and the data blocks. The block groups are normally organized into blocks that are eight times the block size (e.g. a system that utilizes 1,024 byte block sizes would form a block group consisting of 8,192 blocks). The block allocation bitmap is used to maintain the allocation status of blocks within a block group (i.e. allocated or free). A system that uses a 1,024 byte block size, has 1,024 bytes responsible for keeping track of the 8,192 blocks. As such, each block is mapped to one bit in the bitmap. A value of one indicates that the block has been allocated while a zero value signifies that the block is free.

The inode allocation bitmap is similar to the block allocation bitmap. Inodes are special data structures that are 128 bytes in length and are used to represent a file. Inodes contain file information (e.g. modification, access, creation, and deletion times, ownership, permissions, size, and pointers to the actual data blocks of the file). The inode also maintains a link count that keeps track of the number of links to a file. When a file is unlinked, the kernel decrements the link count. If the link count

reaches zero, then the kernel frees up the data blocks associated with the file unless the file is open. Indeed, this is a technique often utilized by an intruder; that is, open a file and then unlink it (the file can still be read from and written to) [3, 12].

The Ext2fs also contains an inode flags field that can be used to indicate the preference for secure deletion. However, the mechanism required to provide secure deletion for most Linux operating system releases has not been implemented; most likely due to the performance penalty. The secure deletion flag can be controlled by using the change file attributes command known as `chattr`. The flag can be turned on by submitting the `+s` flag for a directory or file name and turned off by issuing the `-s` flag [1].

## 5.4 Disk Wiping and Deletion Tools

Disk wiping is a broad term used to identify the method for overwriting a file until the file is rendered unrecoverable. There are many methods of performing a wiping operation (e.g. the Department of Defense's U.S. DoD 5200.28-STD and the Peter Gutmann scheme). These techniques normally overwrite a file a specified number of times using a particular pattern or procedure believed to be effective in altering the magnetic structure of the data. Disk wiping tools are available both commercially and as freeware and, naturally, have a varying degree of effectiveness. The most significant issue that disk wiping tools face is that the majority of them simply do not adequately clean a hard drive of all remnants of data. In fact, even if a tool was able to completely wipe a drive, some exotic techniques (see below) provide investigators with the ability to retrieve data from media that has been completely overwritten. File shredding is a similar concept though it has the connotation of overwriting a file upon deletion vice cleaning up deleted files on a schedule or at a specified time. Other deletion tools simply overwrite files or empty space by writing zeros to the target location [8, 10, 17, 24].

### 5.4.1 Windows-based Tools

It has been said that most disk wiping tools work the same way, however, based on ease of use, understandability, features, and wiping options, some standout products emerged. Both Eraser [37] and WipePro+ [38] (freeware with a small fee for commercial use) provide single pass, DoD, and Gutmann wiping options. Both tools include a DOS utility that can be used to wipe the swap file (something that is not easily done within Windows since the file can not be cleaned while in use). The WipePro+ utility also includes an Information Center that provides a resource concerning disk wiping concepts. Another recommended tool is Msweep [39], which is said to be a very efficient DOS-based tool that will overwrite slack space as well as unallocated space.

A file shredder known as Shredder is available with certain packaged products from McAfee (e.g. QuickClean and McAfee Utilities) as well as a standalone program. Shredder can delete files with 7-pass, US Government standard wiping, or a custom selectable number of passes [5, 17, 24].

### 5.4.2 Linux-based Tools

Several Linux file cleansing tools are available and the same effectiveness caution applies equally here as well. Most of the Linux tools can only be used to wipe files and can not be used to clean empty disk space. Some use multiple random passes while

others provide only a single pass that overwrites the file with zeros. The list of tools include: GNU `shred` (by Colin Plumb), `srm` (by Todd Burgess), `wipe` (by Tom Vier), and `srm` (as part of the THC kit) from The Hacker's Choice (THC) group. The `shred` utility requires specific file system characteristics (i.e. file overwrites must be done in place). If this condition is not met, then the secure deletion operation is not performed and no error message is displayed to the user. The Linux `dd` utility can be used to zero out empty space on a partition. For example, to write zeros to the empty space on the `/home` partition, the following commands could be used:

```
# dd if = /dev/zero of = /home/bigfile
# sync
# rm /home/bigfile
# sync
```

Additionally, the `secure_delete` package available from THC provides a utility known as Secure Fill (`sfill`) that can be used to overwrite empty space. The tool offers more sophisticated overwrite algorithms than what the simple `dd` example above provides. For example, by default `sfill` will perform 38 writes utilizing the Gutmann method. The tool also contains command line options that allow a user to select less secure overwrites (a pass that writes random characters followed by another pass using the `0xff` value) as well as faster, less random passes (i.e. that exclude the use of `/dev/urandom` and don't sync between writes) and the ability to write zeros on the final pass [8, 36].

## 5.5 Why a Forensics Examiner Cares

A forensics examiner will want to recover any and all relevant information. It is their responsibility to search through a forensic image in order to undelete or recover as many files or file fragments as possible. There are many types of files that can potentially contain valuable evidence (e.g. temporary, system, and application data). Many operating systems and computer programs utilize swap files to temporarily store information. For example, Windows NT uses a file known as `pagefile.sys` for its swap space. UNIX systems have dedicated swap partitions (where either a portion or an entire disk is set aside specifically for file swapping activities) that can be searched through for strings. Hibernation files can provide another gold mine of information since they contain all of the data necessary to restore a previous session (though difficult in practice, it is possible to reconstruct the full session using only this data).

Additionally, many programs store binary data both during their execution and upon termination. Examples of programs that write temporary data while operational include Internet browsers, email programs, compression applications, and word processors. For instance, Netscape, Mozilla, Firefox, and IE maintain history databases that contain deleted entries that can be recovered. Similarly, Microsoft Outlook stores email content in the `outlook.pst` file. While special processing is required to read the Outlook file, it may hold deleted emails. Documents produced by Microsoft Word can include images and other media that may be of interest to an investigator and, it too, writes temporary files (through the use of an auto-save mechanism) for history editing purposes. Deleted files can also contain hidden data that was placed there by an offender or an intruder. Indeed, files can be deleted by malicious users and programs or simply erased by those attempting to conceal their offenses. Once removed, a

suspect may believe that the data no longer exists; these deleted files can often make or break an investigation [7, 25].

### 5.5.1 Relevance of Drive Slack

Depending on the file system in use, the file size, and other factors previously mentioned, deleted files may be partially overwritten. In some cases, that may mean a portion of the deleted file can be found in slack space. As such, a forensics examiner might be able to extract and reconstitute the file fragments thereby enabling them to be viewed in their near original state. Since this is often a manually intensive process, this type of recovery is easier for file types that are constructed from human readable elements. For instance, an examiner can often infer the order and importance of components that make up Microsoft Word documents. When header information has been destroyed or damaged, locating and restoring file fragments can be considerably more difficult (though still possible). For example, if the header from a Word document was overwritten, the fragment could be compared against a complete document (or set of documents) that had been created by the same version of the software. This inspection may lead to a determination of the amount of header that was lost and allow a type of digital surgery to take place; that is, a portion of a donor document's header can be used to replace the missing data. Following the unification of the files, Word may be able to recognize and display the repaired file. Image and audio/visual files present a more difficult problem if their header information is overwritten since they often contain important information such as image specifications and other data required for display or playback. In such cases, it may be possible to obtain a portion of the original file utilizing digital surgery and a header from another file [7].

## 6. TRUE DELETION AND DESTRUCTION

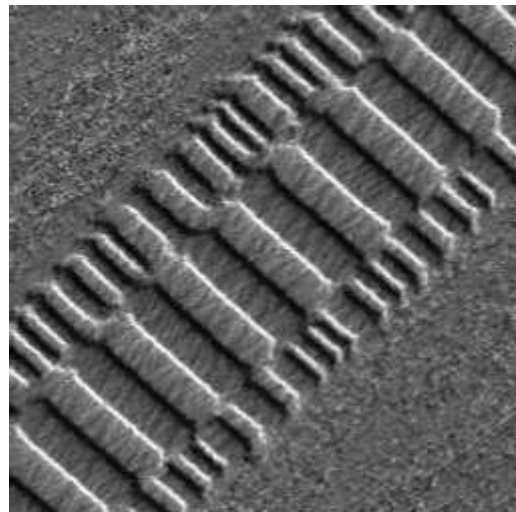
### 6.1 Is It Possible?

While there are various standards for deletion of data (see below), there also appears to be some controversy regarding the recoverability of overwritten data. For instance, as part of a project, twenty commercial data recovery companies were contacted and asked if they could recover a 100KB file that had been completely overwritten once with random data. Only one of the respondents indicated that they possessed the ability to recover the data, but estimated their chance of successfully being able to retrieve the information at less than one percent [1]. Like the response received from many other questions concerning different areas of computing, the answer to the question of whether or not data can truly be destroyed is, "it depends." The single most important factor in this consideration appears to be related to the value of recovering what was believed to be deleted. That is, if the data to be recovered is valuable enough, techniques from the relatively inexpensive to the time consuming and costly can be used to recover the data.

There are several reasons why true data deletion is a very difficult problem to solve. For example, when data is being written to the medium, the polarity of most, but not all, of the magnetic domains is being set by the write head. In part, this is due to the fact that the writing device does not have the ability to write precisely in the same location each time data is written. It can also be attributed to variations in media sensitivity and field strength variability over time. Being binary machines, we like to believe

that zeros and ones are being written to the disk, but Gutmann proposes that when a zero is overwritten with a one, the value is actually closer to 0.95 and when a one is overwritten with a one, the value is approximately 1.05. The circuitry used by the disk is configured so that both values are interpreted as having a value of one, but by using more sophisticated electronics it is possible to differentiate between the different layers and determine the original values [14, 18].

When overwriting data, drive head position deviations from the exact location of the previous information may leave significant portions of the original data along the edge of a track untouched. This has been compared to a three-lane highway where the majority of the overwriting is taking place in the middle lane (see Figure 8). Track width modulation can also be dependent on the phase relationship between the old and new patterns. The track edge problem is being ameliorated as platter densities continue to increase in an effort to boost the maximum drive capacity (i.e. the width of a track on a fixed sized platter must shrink to accommodate the higher capacity) [11, 18]. To fully understand this implication, a more in-depth look at the recording methods used is required.



**Figure 8. Overwritten, residual information along the sides of a track on a magnetic disk [14].**

As bit density increases, the peaks in the analog signal can appear to overlap, which causes what is referred to as intersymbol interference and can lead to data errors. To reduce the possibility of intersymbol interference, signal peaks are separated as far as possible from each other. The read head detects transitions in magnetization (i.e. a transition is encoded as a value of one and the lack of a transition is encoded as a zero), which is then represented as a peak in the head output voltage. To read the data, the output is differentiated and searched through for "zero crossings". Long strings of zero values make accurate clocking difficult; therefore, a limit on the maximum number of zeros is imposed. These factors are accounted for and implemented through a form of run-length-limited (RLL) coding. The parameters selected for use in RLL coding are tailored such that adjacent one values are far enough apart to avoid intersymbol interference and close enough to each other to prevent

synchronization loss. A more computationally expensive technique known as Partial-Response Maximum-Likelihood (PRML) encoding is used on the latest high-density drives. PRML codes are actually RLL codes that have additional, more sophisticated, constraints. The PRML codes avoid intersymbol interference errors by employing a digital filtering technique that shapes the read signal to display the desired frequency and timing characteristics. While using the same heads and media, PRML channels can achieve the same low bit error rate as a standard peak-detection method on media with much higher recording densities (density increases of 30-40% over standard RLL channels can be achieved). A side benefit of the PRML coding is that data is encoded with much smaller magnetic domains (i.e. closer to the physical capacity of the magnetic media and nearly double that of the nearest RLL equivalent). Drives that employ PRML encoding require sophisticated signal processing in order to read the most recently written data, which makes overwritten layers that much more difficult to decipher. In fact, where PRML encoding is used, Gutmann suggests that a good scrubbing of random data should be all that is required to clean previously written data [18].

The concept of data destruction using an overwriting scheme is similar to the basic idea behind degaussing; flip the magnetic domains on the disk back and forth as much as possible while trying not to write a duplicate pattern twice in a row. The use of alternating patterns coupled with many overwrites should produce the desired magnetic domain flipping. This is based on the conclusion that each track contains an image of everything that was ever written to it and the individual contribution from each layer is progressively more difficult to read the further back it was made; hence more overwrites mean that the previous data is continuously pushed farther away. Gutmann developed a set of 22 overwrite patterns that he believes should be able to erase everything (regardless of what encoding was used). He suggests that up to 35 consecutive writes be utilized to destroy the data and that a random order be used to make it more difficult to guess which of the known data passes were made at any given point during the process. The 35-pass overwrite technique covers a blend of scenarios that go back more than thirty years and encompass all types of normally used encoding technologies (i.e. modern high-density drives will fair no better using 35 passes than by using a good random scrubbing). Finally, he feels that even if 10KB of data remained after erasing a modern, high-density, 80GB drive, the chances of someone finding it are close to zero [18]. Essentially, overwriting the data only ensures that the difficulty level of recovering the data has been increased; it does not mean that data recovery is impossible or that the storage locations have been completely sanitized [24].

Someone aware of the existing analysis techniques that allow data to be read from a disk even after it has supposedly been erased might believe that destroying the hard drive with a hammer may offer a better option. Suppose they took even more precaution and decided to physically destroy the platters by breaking them into pieces. While they may succeed in creating a puzzle that is very difficult to piece back together, once reassembled, the techniques they feared could be used against them would still be available. Such radical measures would only reasonably be used in cases of extreme importance, but the possibility exists [11]. Even if all of the pieces could not be put back together, tests

conducted by the NSA offer proof that 1mm x 1mm pieces of hard disk media contain retrievable data [15].

Another potential data destruction method is degaussing. Unfortunately, due to data persistence, it appears that for degaussing to be successful, the source would need to be quite powerful. Demonstrations have shown that a magnetic force of at least five times greater than the coercivity of the medium is required to sufficiently destroy the data. Measured in Oersted, coercivity is the amount of applied magnetic field required to reduce magnetic induction to zero. The higher the coercivity, the harder the data is to erase. For instance, a study conducted using commercial degaussers reported that at a ½ inch above the platform of a platter with a peak horizontal gauss of 1,700, the data signal erasure was no better than 50 percent and fell to lower than 10 percent. This implies that even a commercial grade degausser will leave up to 50 percent of the data on a drive intact. This is likely due to the fact that degausser ratings are based on their capacity to erase media to the point of providing a clean recording surface rather than the elimination of all data (or making retrieval impossible). A degausser with enough power to provide the required data destruction capability is well beyond what the average consumer can afford [11, 15].

Absolute assurance of data annihilation requires extreme measures be taken with the hard drive or the disk platters themselves. For example, incineration, sanding the surface of the platters into tiny particles, using acid to strip the surface of the platters away, or converting the platters into a liquid by melting them down. Once again, the data content will be the deciding factor in which destruction method is appropriate. In cases of national security or data that has been classified as being at the Sensitive Compartmented Information (SCI) or Top Secret levels, these extremes make sense [11, 35].

## 6.2 Legal Requirement and Standards

There are legal requirements as well as government standards for properly disposing of data whether it exists on paper, tape, floppy disk, hard drives, optical drives, or other forms of long-term storage. The Health Insurance Portability and Accountability Act (HIPAA), Gramm-Leach-Bliley Act (GLBA), and Department of Defense (DoD) all have data disposal requirements. In addition, the National Security Agency (NSA) provides its own set of guidelines and guidance for data destruction. There are, however, some issues concerning the government standards. For instance, the standards are often too old and do not account for newer techniques available for recording and retrieving data. As previously mentioned, the latest high density drives use the PRML encoding scheme, but the standards are not current with regard to the more sophisticated channel coding techniques. Additionally, the use of Magnetic Force Microscopy (MFM) to recover data does not appear to be considered in the documents. Another consideration is that the standards may actually be understated in an attempt to fool or confuse opposing intelligence agencies [9, 15].

There are two relevant HIPAA requirements. The first is concerned with data disposal and requires that policies and procedures be implemented to address the final disposition of electronically protected health information (especially with regard to the hardware or electronic media on which it is stored). The second requirement speaks to media re-use. It states that

procedures must be implemented for the removal of electronically protected health information from electronic media before it can be made available for re-use.

The GLBA has similar mandates concerned with data erasure and destruction. Data is to be erased from all computers, diskettes, magnetic tapes, hard drives or any other electronic media that contains customer information when it is being disposed of. Further, the requirement is to effectively destroy the hardware and promptly dispose of outdated customer information [15].

The DoD has several areas regarding data disposition. For example, DoD 8500.1 Section 5.8 assigns responsibility to the NSA to develop sanitization policy for all classified information [15]. The DoD 5220.22-M (also known as the National Industrial Security Program Operating Manual or NISPOM) outlines steps to clear and sanitize rigid non-removable disks. To clear a disk, all addressable locations must be overwritten by a single character. To sanitize a disk, one of the following techniques must be used:

- Degauss with a Type I degausser.
- Degauss with a Type II degausser.
- Overwrite all addressable locations with a character, its complement, then a random character followed by a verification procedure (this is not an approved method for media containing Top Secret information).
- Destruction, which includes the following: disintegrate, incinerate, pulverize, shred, or smelt [24].

The NSA Media Technology Center (MTC) has been assigned responsibility for the determination of appropriate products and procedures to ensure that classified information cannot be recovered (which corresponds to the previously mentioned DoD 8500.1 Section 5.8). They also create and publish the Evaluated Products List that agencies can reference to ensure that they are operating within the prescribed compliancy guidelines.

Several methods have been approved at various governmental levels (i.e. may not be acceptable for all classifications of data or approved by the NSA) and published. The Overwrite Disk Method (not NSA approved) must be one of the multiple pass techniques: Gutmann Method, Schneier Method, or DoD 5220.22-M. The Destruction Method consists of three different levels (i.e. minimal barrier, barrier, and complete barrier). The minimal barrier devices include a hammer, nail, and portable crushing device. The barrier devices include disintegrators, chippers, and a hammer mill. The complete barrier list includes incineration and sanding the hard disk drive's platters. The Encryption Key Destruction Method (not NSA approved) requires that all data be encrypted before being written to disk. The encryption keys are then securely stored and are usually encrypted in a key management system. The key management system will prompt security officers to destroy keys when their retention period expires and provide a "Certificate Of Destruction" following their demolition. To reduce the risk of cracking the encryption, use of FIPS approved algorithms such as AES 256 is encouraged. The Degaussing Method utilizes the Degausser Evaluated Products List (DEPL), which is an NSA approved degausser list. The list specifies model identification information of degauss equipment that has been evaluated by the NSA and found to satisfy their requirements for erasure of magnetic storage devices that retain

classified or sensitive data. It should be noted that a product on the DEPL has not been endorsed by the U.S. Government or the NSA/CSS; making the list simply means that it has met the requirements [15].

The NSA does not approve of the overwrite method for disk sanitization for the following reasons:

- Crashed drives cannot be overwritten.
- Bad Sectors can retain classified information that can be retrieved.
- Write heads can shift thereby hindering the overwrite program's ability to effectively overwrite data and allow tracks the ability to retain retrievable classified information.
- G-List sectors. Defective record areas (blocks) are marked as unusable in some drives and moved to what is known as the g-list due to excessive errors. These areas can not be reached by erasure programs and leave the original data in place [15].

## 7. DATA RECOVERY

### 7.1 Discovering and Recovering Data

There are so many programs available for data discovery and recovery that providing a comprehensive list is beyond the scope of this paper. Here, and in subsequent sections, only the most popular tools (or those cited the most in my research) will be discussed. These software utilities range in price from free (open source) to a cost well beyond what a non-professional would feel compelled to pay. Similarly, the scope of their abilities, specialties, and functionality is equally disparate in some cases and surprising similar in others.

Data detection and retrieval are the natural counterpoints to hiding and deleting data. Recall the aforementioned techniques (see section 4 Hiding Data) that a forensics examiner could use to uncover data during an investigation. For instance, while searching through an NTFS image, the examiner may want to use LADS (List Alternate Data Streams) or the LNS (List NTFS Streams) tool to locate normally hidden ADS files [6]. The Linux operating system provides some handy utilities for discovering files that an intruder may have tried to ensconce. For example, files hidden through the use of camouflage can be unearthed using the file command. The file utility has the ability to determine the type of a file by employing (in order of precedence) file system tests, magic number tests, and language tests. The command will terminate upon the first successful test and report the file type it believes matches the file in question. However, it is possible to trick the file command by using a hex editor to modify the file identification bytes thereby causing it to incorrectly classify a file. Hidden directories in Linux can easily be found by using the find command. The find tool has many options that allow for specific search and filtering functionality, which can be used to reduce the number of potential candidates the examiner must look through. By submitting the `-type d` and `-name ".*"` flags to find, only directories that start with a dot character will be returned.

Where the Ext2fs file systems are in use, an open source tool known as debugfs can be used to find deleted files. To utilize the command effectively, the investigator must have an

understanding of inodes and an idea of which inode matches the file they wish to locate. A list of deleted inodes can be displayed from within the tool by typing `lsdel` at the `debugfs` prompt. The content of a given inode can be viewed by entering `stat` followed by the inode number at the `debugfs` prompt.

Finding unlinked, open files can be accomplished through the use of the `ils` tool, which is part of the suite of utilities available with the Sleuth Kit. By default, the `ils` tool will find and display the inodes of deleted files. However, `ils` can be given the `-o` flag to limit the output to only files that are still open or executing [31].

Since the FAT32, NTFS, and Ext2fs file systems all have varying amounts of slack space, finding hidden data in those areas is simply a matter of knowing where to look. This can be accomplished by hand by physically examining the raw data using a disk editor such as Norton DiskEdit or a hex tool (e.g. WinHex or KHexEdit) [7].

Deleted files can also be recovered using software that has the ability to search unallocated space, swap files, and other digital objects for specific class characteristics such as file headers and footers. File metadata need not be relied on when employing this technique. Essentially, the process is akin to carving files out of a blob-like amalgam of data in unallocated space; hence the name file carving. If a file is fragmented, carving methods will only be able to locate the first position of the file since other fragments will not contain the required signature information [7, 28].

The current front-running, professional level, tool that can be used to find and recover data (and much more) appears to be EnCase. It has been recommended by several authors, papers, presentations, and web sites cited in this paper. EnCase runs on Windows operating systems (e.g. 2000, XP, and 2003) and supports many file systems including Palm (it can even be used to dump the contents of RAM and ROM from a Palm device) [7, 25].

When the recovery power available through software alone is insufficient, other techniques are available. For example, recovery of at least one or two layers of data that has been overwritten is not too difficult to perform. The signal from a hard drive's analog head electronics can be sampled using a high quality digital oscilloscope. The sample can then be downloaded as a waveform to a PC and analyzed in software to recover the previously recorded signal. The software generates an ideal read signal and then subtracts the value actually read from that, which leaves the previous signal as a remnant. The ability to recover this extra information is not exploited by the hard drive electronics, but the circuitry in the oscilloscope is far superior to that employed by the read head. This capability is not possible with newer channel coding techniques such as PRML that require extensive amounts of signal processing. In cases where an oscilloscope just won't do, there are more advanced approaches available. For instance, MFM or STM can be used to read and interpret the overwritten data that still exists beside the new data that can be found along track edges. MFM and STM are particularly efficient techniques since they are used to officially evaluate the effectiveness of disk drive servo-positioning mechanisms [18].

## 7.2 Recovery Technologies

### 7.2.1 Windows-based Tools

The Forensic Toolkit (FTK) is another multi-purpose tool (similar to EnCase and recommended by several sources as well). It appears to be a Windows only program, but provides support for the following file systems: FAT 12/16/32, NTFS, NTFS compressed, Ext2fs, and Ext3fs [7]. Drive Rescue is a freeware utility that can be used to find and recover deleted files as well as files that have been lost due to formatting or corruption. In addition, it has the ability to reclaim files from a drive that can no longer be mounted. The tool's interface navigation is similar to Windows Explorer, which aids inexperienced users by employing a familiar environment. The tool also supports searching for deleted files by their name and provides information regarding the condition of a deleted file; the file's condition is directly related to the amount of data that can be recovered [35]. File Scavenger is a file undeletion tool for the NTFS file system. This utility requires that the originally deleted file has not had any of its space overwritten by more recent I/O storage operations. However, it may be able to recover files that survive a disk reformat [25].

DataLifter and Ontrack's Easy-Recovery Pro are file carvers that can recover many types of files including graphics, word processing, and executable files. The "File Recovery by Type" feature provided by WinHex and E-scripts (available from the EnScript library as part of the EnCase suite) can both be used to carve user defined files. Specialized tools that can be used to extract specific types of files also exist, such as NTI's Graphics Image File Extractor. Some of the aforementioned tools are sophisticated enough that they can be used to extract images from other files (e.g. images inserted into Word documents) [7].

### 7.2.2 Linux-based Tools

FatBack is a file recovery program written by Nick Harbour of the U.S. Department of Defense Computer Forensics Laboratory. It can be used to recover files from FAT12, FAT16, and FAT32 file systems. The tool also supports long filenames, recursive undeletion (i.e. directories), lost cluster chain recovery, and the ability to operate on a single partition or an entire disk [25].

The Sleuth Kit (formerly known as TASK) is an open-source forensic toolkit used to analyze Microsoft and Unix file systems including: FAT, FAT12, FAT16, FAT32, FreeBSD, EXT2, EXT3, OpenBSD, and UFS. It also works on binary images as long as there are no embedded checksums. A suite of tools is provided with the kit that perform various important tasks related to file recovery. Autopsy is a graphical interface (front-end) to the utilities found in The Sleuth Kit. Autopsy is HTML-based and uses a client/server model, which allows the client to be run on any platform that can support an HTML browser [25].

The `unrm` and `lazarus` tools are part of a collection of utilities packaged with The Coroner's Toolkit (TCT). The `unrm` tool can be utilized to access unallocated portions of a UNIX file system and will write any discovered raw data to an output file. The `lazarus` program attempts to reconstruct files from their raw data. Normally, the raw data is gathered and provided by the `unrm` tool. Additionally, `lazarus` can rebuild data objects from other input sources, such as system memory and swap space. It has been claimed that the tool can be used on just about any file system

(with the caveat that success varies depending on the way the data is stored) including FAT32, NTFS, Ext2fs, and UFS [3, 30, 33].

The Recover tool automates direct inode recovery in a manner similar to that found in the unrm and lazarus tools. A script is provided that inquires as to approximately when the file in question was deleted. A list of potential candidate files is then produced as output [3, 30, 35].

The previously mentioned debugfs can also be used to recover deleted files. Recall that the inode information must be known and can be examined by entering lsdel and the stat commands at the tool's prompt. Once the inode number belonging to the file to recover has been located, the file can be reclaimed by using the dump command followed by the inode number and a file name location at the prompt. For example:

```
debugfs: dump <36150> /tmp/reclaim.out.
```

The command above would dump the contents of the inode that exists at location 36150 to the /tmp/reclaim.out file [31].

Scalpel is a file carver that was designed with high performance, frugality, and support for distributed implementation as priorities. Steps were taken to ensure that the amount of search time, number of memory-to-memory copies, and write time were all reduced. The target platform was a bootable Linux distribution (e.g. Knoppix or Helix) and the basic techniques used to perform the carving were made to be easily adaptable to a distributed or cluster-based system. The tool has a benchmark of 5,000 files carved in only 19 seconds from a 1.2GB FAT32 image running on a Pentium II 350MHZ machine [28, 30]. Foremost is another file carving tool that can be used on any digital object (e.g. an evidence file, unallocated space, or a swap file). This tool also supports user defined file types through the addition of the appropriate header and footer information to its configuration file "foremost.conf" [7].

### 7.2.3 Exotic Techniques

#### 7.2.3.1 Magnetic Force Microscopy

Magnetic Force Microscopy is a technique used to provide a high resolution image of resident magnetization patterns with minimal sample preparation. MFM was derived from Scanning Probe Microscopy (SPM) and utilizes a sharp magnetic tip that is attached to a flexible cantilever. The tip is placed close to the surface to be analyzed (e.g. a few nanometers) where an interaction with a sample's stray field emanation occurs [18, 35]. While moving the tip across the surface and measuring the force (or force gradient), an image of the field strength can be formed by monitoring the cantilever position using an optical interferometer or a tunneling sensor [18].

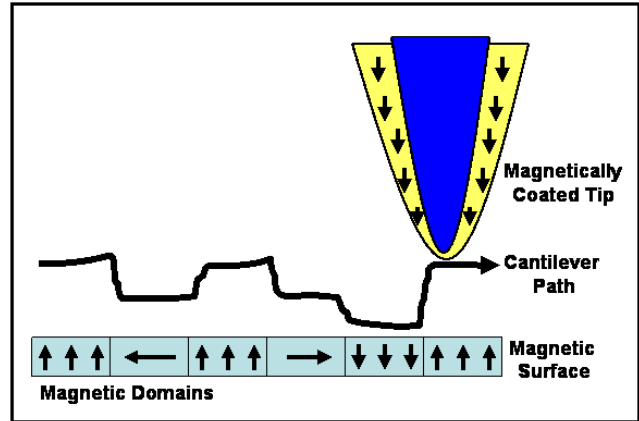


Figure 9. Magnetic Force Microscopy illustration [9].

#### 7.2.3.2 Scanning Tunneling Microscopy

Magnetic Force Scanning Tunneling Microscopy (STM) is a more recent version of the aforementioned MFM technique. The tip used for STM is first plated with nickel (onto a pre-patterned surface), peeled from the substrate, then plated with gold to minimize corrosion. The tip is then mounted in a probe at some small bias potential (a few tenths of a nanoamp using a few volts of DC power) such that electrons from the test surface can tunnel across the gap to the probe's tip. While the probe is being scanned across the surface, a feedback system continuously makes adjustments to the vertical position so that a constant current is maintained. Image generation is then accomplished in the same manner as that used by the MFM technique [18]. The STM technique is a time consuming process since it only reads approximately 50 bits of data at a time and each scan takes nearly five minutes to complete [35]. In contrast, MFM requires very little sample preparation and can provide results fairly quickly. For example, on older drives that utilize a well documented disk format, an image can be produced in less than ten minutes [1, 18].

#### 7.2.3.3 Others

By manufacturers' sales figures, there are several thousand SPM's still in service. Some of the models currently in the field have special analysis features, such as vacuum chucks for standard platters as well as specialized magnetic media analysis modes. These machines can be paired with sophisticated programmable controllers and analysis software, which can help automate the recovery procedures. The use of ferrofluid combined with optical microscopes was a technique employed before gigabit per square inch storage capacities was common. With the high density rates in use today, it is no longer feasible to utilize this method since the magnetic features are smaller than the wavelengths of visible light [18].

## 8. SUMMARY

Part of the body of knowledge that a forensics examiner requires is concerned with discovering and recovering hidden and deleted data. To successfully perform their duties, they must be aware of the potential methods that intruders, attackers, viruses, and worms use during their illicit actions. In order to form a more fundamental understanding of these issues, a comprehension of the basic principles of hard drives, such as physical characteristics, use of magnetization, platter composition, track

and sector organization, disk formatting, and the read/write mechanism is imperative. Equally important is an appreciation of file system characteristics and their structures with regard to hiding and deleting data.

There are several techniques that can be used to hide data including different forms of slack space, ADS, disguising files, using hidden attributes, and even data deletion. File systems play an important role in slack space, and a quick experiment demonstrated the relative ease with which it can be created and brought home the reality of its existence on both NTFS and Ext2fs.

Clicking the delete button doesn't really remove the data from the hard drive for various reasons including performance considerations. Similar to hiding data, differences between the FAT32, NTFS, and Ext2fs file systems concerning their deletion mechanisms are an important consideration for the digital detective. While there is some controversy concerning the recoverability of data after being subjected to disk wiping and deletion, platform specific tools are available that (at a minimum) have the ability to increase the difficulty level of retrieving information. True data deletion does not seem likely without physically destroying the disk drive (or at least the platters), which was considered along with the effectiveness of potential destruction methods. The legal requirements and government standards for data destruction (especially considering HIPAA, GLBA, DoD, and the NSA) were also discussed.

Finally, the ability to locate hidden data and reclaim deleted information is absolutely essential to the digital evidence collection process. Tools that can provide automation and assistance to the forensics examiner for both the Windows and Linux operating systems are available; however, knowledge of how to extract data by hand may be needed in cases that require abilities beyond those currently offered by the best tools. Exotic recovery techniques are most likely only utilized in cases of vital importance.

## 9. REFERENCES

- [1] Bauer, S., and Priyantha, N. B. Secure Data Deletion for Linux File Systems. In *Proceedings of the 10<sup>th</sup> USENIX Security Symposium*, (August, 2001), 153-164.
- [2] Brain, M. *How Hard Disks Work*, Retrieved February, 2006 from: <http://computer.howstuffworks.com/hard-disk7.htm>
- [3] Buckeye, B. and Liston, K. *Recovering Deleted Files In Linux*, Retrieved February, 2006 from: <http://www.samag.com/documents/s=7033/sam0204g/sam0204g.htm>
- [4] Card, R., Ts'o, T., and Tweedie, S., *Design and Implementation of the Second Extended Filesystem*, Retrieved February, 2006 from: <http://e2fsprogs.sourceforge.net/ext2intro.html>
- [5] Carlton, G. H. A Critical Evaluation of the Treatment of Deleted Files in Microsoft Windows Operating Systems. In *Proceedings of the 38<sup>th</sup> Hawaii International Conference on System Sciences*. IEEE, 2005.
- [6] Carvey, H. *The Dark Side of NTFS (Microsoft's Scarlet Letter)*, Retrieved February, 2006 from: <http://www.infosecwriters.com/texts.php?op=display&id=53>
- [7] Casey, E. *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet, 2<sup>nd</sup> Edition*. Academic Press, San Diego, CA, 2004
- [8] Chuvakin, A. *Linux Data Hiding and Recovery*, Retrieved February, 2006 from: <http://www.linuxsecurity.com/content/view/117638/49/>
- [9] Dawson, M., Davis, J., Forgie C., and Tauber S. *Data Recovery*, Retrieved February, 2006 from: <http://staff.washington.edu/dittrich/misc/Data%20Recovery.ppt>
- [10] Dear, D. *An Exploration of Future Anti-Forensic Techniques*, Retrieved February, 2006 from: <http://www.assuremind.com/antiForensics.pdf>
- [11] Devera, D. *The Difficulty of Data Annihilation from Disk Drives: or Exnihilation Made Easy*, Retrieved February, 2006 from: [http://www.thuktun.org/cs574/papers/cs574\\_ddevera.pdf](http://www.thuktun.org/cs574/papers/cs574_ddevera.pdf)
- [12] Dominus, M. J., *Internals of ext2fs*, Retrieved February, 2006 from: <http://perl.plover.com/yak/ext2fs/samples/slide001.html>
- [13] Elmasri, R. and Navathe, S. *Fundamentals of Database Systems, Fourth Edition*. Pearson Education Inc., Boston, MA, 2004.
- [14] Farmer, D., and Venema, W. *Forensic Discovery*, Retrieved January, 2006 from: <http://fish2.com/forensics/pipe/forensic-discovery-book.zip>
- [15] Federspiel, D. *Data Disposal – Gone For Good*, Retrieved February, 2006 from: [http://www.snia.org/education/tutorials/fall2005/security/Data\\_Disposal\\_Gone\\_for\\_Good\\_Rev3.pdf](http://www.snia.org/education/tutorials/fall2005/security/Data_Disposal_Gone_for_Good_Rev3.pdf)
- [16] NTFS.com, "File Storage Hardware and Disk Organization", Retrieved February, 2006 from: <http://www.ntfs.com/hard-disk-basics.htm#Hard>
- [17] Gittings, K. *Where Data Hides and Resides Understanding Hidden Data in Windows*, Retrieved February, 2006 from: [http://www.giac.org/certified\\_professionals/practicals/gsec/3839.php](http://www.giac.org/certified_professionals/practicals/gsec/3839.php)
- [18] Gutmann, P. *Secure Deletion of Data from Magnetic and Solid-State Memory*, Retrieved February, 2006 from: [http://www.cs.auckland.ac.nz/~pgut001/pubs/secure\\_del.html](http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html)
- [19] Wikipedia, "Hard disk," Retrieved February, 2006 from: [http://en.wikipedia.org/wiki/Hard\\_disk](http://en.wikipedia.org/wiki/Hard_disk)
- [20] Microsoft Corporation, "How NTFS Works," Retrieved February, 2006 from: <http://technet2.microsoft.com/WindowsServer/en/Library/81cc8a8a-bd32-4786-a849-03245d68d8e41033.mspx>
- [21] Kuepper, B. *What You Don't See On Your Hard Drive*, Retrieved February, 2006 from: <http://www.sans.org/rr/whitepapers/incident/653.php>
- [22] Microsoft Corporation, "Limitations of FAT32 File System," Retrieved February, 2006 from:

- <http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q184006&>
- [23] eMag Solutions, “Magnetic Hard Disk Drive,” Retrieved February, 2006 from: <http://www.usbyte.com/common/HDD.htm>
- [24] Mallery, J. R. *Secure File Deletion, Fact or Fiction?*, Retrieved February, 2006 from: <http://www.sans.org/rr/whitepapers/incident/631.php>
- [25] Mandia, K., Prorise, C., and Pepe, M. *Incident Response & Computer Forensics, Second Edition*. McGraw-Hill/Osborne, Emeryville, CA, 2003
- [26] Martin, D. *Windows, NTFS and Alternate Data Streams*, Retrieved February, 2006 from: [http://www.giac.org/certified\\_professionals/practicals/gsec/0715.php](http://www.giac.org/certified_professionals/practicals/gsec/0715.php)
- [27] Microsoft Corporation, “Microsoft Extensible Firmware Initiative FAT32 File System Specification, FAT: General Overview of On-Disk Format, Version 1.03,” Retrieved February, 2006 from: <http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc>
- [28] Pan, L. *Data Recovery Software Tools: Today and the Future*, Retrieved February, 2006 from: <http://www.deakin.edu.au/~ln/PhD/Talks/DataRecovery.pdf>
- [29] Pendyala, K. S., *Cyber Forensics*, Retrieved March, 2006 from: <http://www.csi-pune.org/downloads/cyberpune.ppt>
- [30] Richard, G. G., and Rousev, V. *Scalpel: A Frugal, High Performance File Carver*, Digital Forensic Research Workshop (DFRWS), New Orleans, LA, 2005
- [31] Robertson, G. *An Introduction to Hiding and Finding Data on Linux*, Retrieved March, 2006 from: [http://www.giac.com/practical/GSEC/Gary\\_Robertson\\_GSEC.pdf](http://www.giac.com/practical/GSEC/Gary_Robertson_GSEC.pdf)
- [32] Stallings, W. *Operating Systems: Internals and Design Principles, Fourth Edition*. Prentice-Hall Inc., Upper Saddle River, NJ, 2001.
- [33] Carnegie Mellon University, “Using The Coroner’s Toolkit: Rescuing Files with Lazarus”, Retrieved February, 2006 from: <http://www.cert.org/security-improvement/implementations/i046.03.html>
- [34] Microsoft Corporation, “Windows XP Professional Resource Kit, Working with File Systems,” Retrieved February, 2006 from: <http://www.microsoft.com/technet/prodtechnol/winxppro/reskit/c13621675.msp>
- [35] Zetterstrom, H. *Deleting Sensitive Information Why Hitting Delete Isn’t Enough*, Retrieved February, 2006 from: <http://www.sans.org/rr/whitepapers/privacy/691.php>
- [36] Van Hauser. *Secure Delete, Secure FILL*, Retrieved July, 2006 from: <http://linux.softpedia.com/get/Security/THC-SecureDelete-10390.shtml>
- [37] Eraser: <http://www.tolvanen.com/eraser/>
- [38] WipePro+: <http://www.marcompress.com/AboutWipePro.htm>
- [39] Msweep: <http://www.secure-data.com/ms.html>