Chapter 7

COUNTERING HOSTILE FORENSIC TECHNIQUES

Scott Piper, Mark Davis and Sujeet Shenoi

Abstract Digital forensic investigations can be subverted by hostile forensic techniques and tools. This paper examines various hostile forensic techniques, including the exploitation of vulnerabilities in standard forensic procedures and denial of service attacks on forensic tools during imaging and analysis. Several techniques for concealing evidence within file systems and external to file systems are highlighted. In addition, strategies for countering hostile forensic techniques and tools are discussed.

Keywords: Hostile forensics, subversion, denial of service, evidence concealment

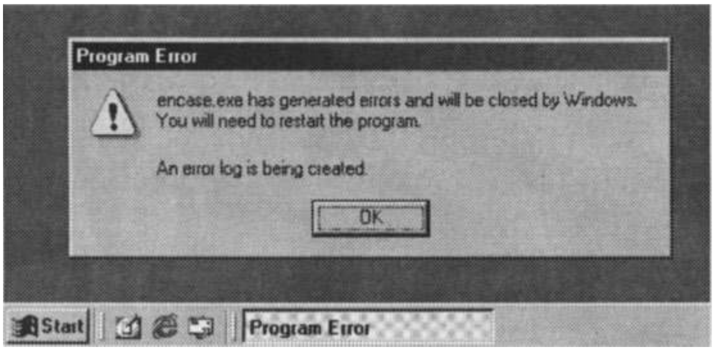
1. Introduction

The emerging discipline of digital forensics brings advanced scientific and engineering techniques to bear on the tasks of detecting, recovering and analyzing electronic evidence [12, 14]. However, certain elements of the hacker community are engaged in developing "anti-forensic" or "hostile forensic" techniques and tools to subvert digital forensic investigations [8, 13, 15]. Efforts have been undertaken to exploit flaws in digital forensic techniques and tools. The holy grail of these efforts is to find an exploit, e.g., a buffer overflow, that would result in the execution of malicious code in forensic tools used by law enforcement agencies. Such an exploit could make the tools unable to display certain data, make them delete evidence, or simply prevent them from operating. Fortunately, such an exploit has not yet been created, but hostile forensic techniques and tools abound [8, 13, 15].

Some hostile forensic techniques hinder investigations by hiding evidence, destroying evidence or by ensuring that little or no evidence is created. Others exploit vulnerabilities in forensic procedures and tools

Please use the following format when citing this chapter:

Piper, S., Davis, M., Shenoi, S., 2006 in International Federation for Information Processing, Volume 222, Advances in Digital Forensics II, eds. Olivier, M., Shenoi, S., (Boston: Springer), pp. 79–90.


Program Error

encase.exe has generated errors and will be closed by Windows. You will need to restart the program.

An error log is being created.

Figure 1. Result of a denial of service attack on EnCase v4.15.

to prevent evidence from being discovered. More insidious are hostile techniques that perpetrate denial of service attacks on forensic tools to prevent them from imaging media or analyzing evidence.

Nothing can be done when no digital evidence is created. When digital evidence is destroyed, not much can be done to recover it without the use of specialized equipment. However, if evidence does exist on a computer system or network, it can be found and analyzed, even when hostile forensic techniques have been employed.

This paper discusses the state of the art in hostile forensics and presents strategies for countering hostile forensic techniques and tools. The next section focuses on techniques for subverting investigations and perpetrating denial of service attacks on forensic tools. Following this, techniques for concealing evidence within file systems are discussed. Finally, strategies for hiding data in devices that are not normally seized and in devices that are not easily imaged are evaluated.

2. Subversion and Denial of Service

Investigations cannot proceed if the forensic tools themselves are incapable of detecting, recovering or analyzing evidence from computer systems. One hostile forensic strategy is to exploit a vulnerability in a forensic technique or procedure to prevent the discovery of evidence. Another is to launch a denial of service attack on a forensic tool during imaging or analysis to simply prevent it from operating properly.

Many file systems permit the creation of arbitrarily deep directory structures. However, EnCase v4.15 is not designed to traverse a directory tree more than 253 directories deep. When an investigator attempts to use EnCase v4.15 to examine such a directory tree, the tool exhibits a fatal error (Figure 1).

This section discusses techniques for defeating forensic procedures and tools. These techniques, involving the manipulation of timestamps, insertion of compression bombs, and use of sparse files and magic numbers, are described along with strategies for countering them.

2.1 Timestamps

A computer incident, e.g., a network intrusion, occurs within some period of time. Forensic investigators focus the majority of their efforts on discovering what happened during that time frame. The analysis typically involves reviewing the modification, access and creation (MAC) times of files to determine what data may have been accessed during the intrusion. Many forensic tools provide functionality for filtering files that meet the temporal criteria pertaining to an incident. Malicious individuals have attempted to subvert investigations by changing the timestamps associated with incriminating files.

The Linux `touch` command can be used to change file timestamps to the current time, and to reset MAC times to arbitrary values. Several utilities (e.g., fileTweak) are available for changing timestamps on Linux, FAT and NTFS file systems. The Metasploit Project [13], which produced a framework for developing exploit code, created the Timestomp utility that targets NTFS files. In addition to MAC times, NTFS files have an "entry modified" time. Timestomp is the first tool that permits the modification of all four timestamps associated with NTFS files [13].

File timestamps cannot be trusted. This fact should be taken into account when filtering files during the examination of evidence recovered from a computer system or network. Other evidence, such as access logs, should be considered when searching for files that are relevant to an incident.

2.2 Compression Bombs

A compression bomb is a file that expands massively when it is uncompressed, causing the system to crash [2]. For example, the `42.zip` [1] compression bomb is a mere 42KB. When uncompressed completely, it expands to an astounding 4.5PB (Petabytes).

The technique for creating a compression bomb is relatively simple. First, a large file containing zeroes is compressed (Figure 2). Multiple copies of this compressed file are combined into a new file, which is compressed. Copies of the new compressed file are then made, and the copies are compressed again into a single file. This procedure is repeated to produce a small – but extremely potent – compression bomb.

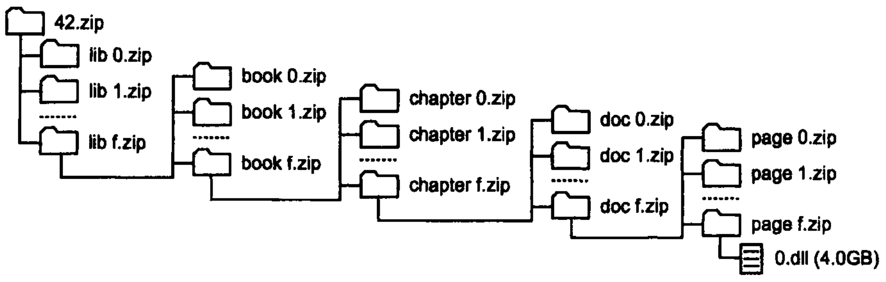


Figure 2. Uncompressed version of the 42.zip compression bomb.

Compression bombs were originally used to disable anti-virus filters. Typically, an attacker would email a compression bomb to a targeted computer system. The anti-virus software on the email server would attempt to scan the compression bomb for viruses by uncompressing it. In the process, the anti-virus software would exhaust its memory resources and crash, exposing an open portal for the attacker to send the real virus. Similarly, an individual wishing to disrupt an investigation might store several compression bombs on a hard drive or other computer media, causing digital forensic tools to crash.

The commonly used digital forensic tools react differently to compression bombs. The Forensic Toolkit (FTK) (v1.42 build 03.12.05) freezes and becomes unusable when it attempts to acquire the image of a drive containing a compression bomb. EnCase (v4.15) is able to acquire an image, but it freezes when it burrows too deep into the compression tree. ILook (v7.0.35), on the other hand, is unable to traverse more then one compression level; therefore, a file that is compressed two or more times is inaccessible to ILook.

Once a compression bomb has been identified, it can be ignored for the purpose of gathering evidence. High compression ratios are achieved by compressing files composed entirely of zeroes. Consequently, it is unlikely that useful data is stored within a compression bomb.

2.3 Sparse Files

Sparse files are an obscure feature of Ext2/Ext3 and other file systems (e.g., NTFS) [5]. These files allow data to be written to any location within the file. A sparse file has few data items, and the locations that do not hold data are assigned zero values. It is inefficient to have many blocks on a disk that contain identical data values, especially when the values are all zeroes. Consequently, sparse files map all blocks with zero values to a single block on the disk (e.g., Block 0 for Ext2/Ext3).

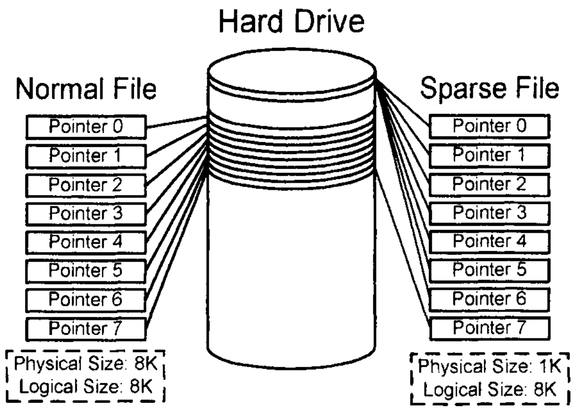


Figure 3. Comparison of normal file and sparse file storage.

Figure 3 shows how normal and sparse files are stored in a file system with 1K blocks. Pointers 0 through 6 of the sparse file (corresponding to blocks with zero values) point to the same block. Consequently, while the logical size of the sparse file is 8K, its physical size is only 1K.

```
# echo "This is a 1GB file" | dd of=sparse.txt bs=1K seek=1048576
0+1 records in
0+1 records out

// Now show the file sizes; logical, then physical
# ls sparse.txt
-rw-r--r--    1 root     root          1.0G Jun  3 16:19 sparse.txt
# du sparse.txt -h
4.0K     sparse.txt
```

Figure 4. Sparse file creation procedure.

Figure 4 presents a Linux console procedure for creating a 1GB sparse file on an Ext2 formatted floppy disk. A 1TB sparse file can also be created on a floppy disk. However, when formatting the disk, it is necessary to set the number of inodes to a value greater than the default value.

Digital forensic tools react to sparse files in unpredictable ways depending on the size of the file. FTK breaks a 1GB sparse file into smaller files; all the contents are stored in the last file, which may be viewed using FTK. On the other hand, while FTK will show that a 1TB sparse file exists, it cannot display its contents. EnCase crashes when used to view a 1GB sparse file. Like FTK, EnCase will show that a 1TB sparse file exists, but it cannot display its contents. Regardless of its size, ILook will show that a sparse file exists but it cannot display its contents.

Note that a sparse file may contain important evidence anywhere within the file, not just at the end. Therefore, after determining that a sparse file exists, an investigator must review a hex dump of the file system to analyze the file.

2.4 Magic Numbers

Every file system has a signature, called a "magic number," that allows the operating system to determine its format [5]. The Ext2/Ext3 file system has a 2-byte magic number of `53 EF`; FAT has a magic number of `55 AA`. File systems continue to function normally even when the magic number is corrupted. However, most software, including digital forensic tools, cannot determine the correct file system, which prevents them from functioning properly. For example, a forensic tool would not be able to parse the data structures in an imaged file, although it would still permit hex dumps of the image.

The magic number on an Ext2 formatted floppy disk can be overwritten by issuing the Linux command:

```
# dd if=/dev/zero of=/dev/fd0 bs=1 count=2 seek=1080.
```

If the magic number (or the beginning) of a partition is overwritten, it is first necessary to determine the file system. Next, the magic number in the image must be corrected to permit analysis using forensic tools.

A related denial of service attack involves overwriting a partition table so that forensic tools cannot determine where partitions begin and end. The tools assume that the entire drive is one giant partition and do not parse the real partitions correctly. The Linux utility **gpart** [3] can be used to reconstruct the partition table in such a situation.

3. Data Concealment within File Systems

This section describes how secret information may be hidden within file systems to evade detection by traditional digital forensic tools. Three data hiding techniques, involving alternate data streams, file slack space and reserved locations of file systems, are discussed along with strategies for detecting and recovering hidden data.

3.1 Alternate Data Streams

Microsoft Windows is commonly installed on a hard disk using the NTFS file system. NTFS provides more functionality than FAT, which was used in earlier versions of Windows. Alternate data streams, one of the new features of NTFS, can be used to conceal data [12].

NTFS files are interpreted by the operating system as streams of data associated with a filename. Typically, files only have one stream of data, but additional data streams may be added to store information about the file, e.g., summary information about the file, keywords and comments. Within the Windows XP GUI, an alternate data stream can be created or viewed by right-clicking on a file, selecting `Properties` and then selecting the `Summary` tab.

```
C:\>echo hello world > file.txt
C:\>echo this data is hidden > file.txt:secret
C:\>more < file.txt
hello world
C:\>more < file.txt:secret
this data is hidden
```

Figure 5. Alternate data stream creation procedure.

The DOS command prompt can also be used to create alternate data streams. Alternate data streams are created in the same way as normal files, but the file is referenced using the file name and the alternate data stream name separated by a colon.

Figure 5 shows the procedure for creating a file with contents `hello world`. An alternate data stream called `secret` is associated with the file; this alternate data stream contains `this data is hidden`. Figure 5 also shows that when file contents are displayed, only the data associated with the default stream (`hello world`) appears. The alternate data stream is displayed using the command: `more < file.txt:secret`. This demonstrates that the contents of the alternate data stream are distinct from the default stream.

Alternate data streams are useful for concealing data because the Windows operating system lacks the functionality to access them. Indeed, Windows ignores alternate data streams when reporting file sizes and free space on a disk. For example, the file `file.txt`, which contains both `hello world` and `this data is hidden`, is listed as being only 14 bytes. Alternate data streams are not listed when viewing directory listings or browsing folders using Windows Explorer. In fact, the only way to discover an alternate data stream is to use third-party software, e.g., `Streams` [19].

Even more astonishing is the fact that, in Windows, the only way to delete an alternate data stream is to delete the entire file. Since alternate data streams can be associated with files as well as directories (including the root directory), the removal of an alternate data stream is somewhat

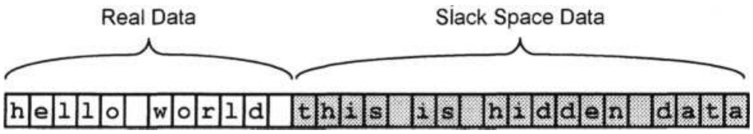


Figure 6. Hidden data in file slack space.

problematic. The **Streams** utility [19] can be used to selectively delete alternate data streams.

Until a few years ago, data concealed using alternate data streams could be discovered only by string searches and hex dump analyses. Most forensic tools are now able to detect the presence of alternate data streams.

3.2 Slack Space

Most file systems divide their partitions into blocks of equal size. Instead of allocating just enough bytes to store a particular file, complete blocks are reserved for the file. For example, on a file system with 512-byte blocks, a 14-byte file takes up 512 bytes of storage, and a 526-byte file uses 1024 bytes. Thus, files can grow within their allocated blocks; when they outgrow them, additional blocks can be allocated from elsewhere on the disk. Slack space is the unused space within a block [5, 12].

File slack space is not overwritten unless the size of the file increases. If the file shrinks, old data residing in the slack space could be retained indefinitely.

Data may be hidden in slack space (Figure 6), for example, by using the **bmap** tool that was originally created to read slack space. Many files, especially those associated with the operating system and applications, are updated rarely, if ever. The slack space of these files is a good place to hide data. Most forensic tools can be used to examine slack space, but investigators must know which files are most appropriate for hiding data and search their slack space for concealed evidence.

3.3 Reserved Locations

File systems have reserved locations that are used to support upgrades and new features. Since the reserved locations are unused until a file system is updated, data written to these locations neither overwrites useful data nor affects system operation.

The reserved locations of the Ext2/Ext3 file systems can be identified by reviewing Linux kernel source code in **./include/linux/ext2_fs.h**. Figure 7 shows the source code for one of the structures in the Ext2 file

```
struct ext2_group_desc
{
        __u32   bg_block_bitmap;        /* Blocks bitmap block */
        __u32   bg_inode_bitmap;        /* Inodes bitmap block */
        __u32   bg_inode_table;         /* Inodes table block  */
        __u16   bg_free_blocks_count;   /* Free blocks count   */
        __u16   bg_free_inodes_count;   /* Free inodes count   */
        __u16   bg_used_dirs_count;     /* Directories count   */
        __u16   bg_pad;
        __u32   bg_reserved[3];
};
```

Figure 7. Source code for the group descriptor.

system. A total of 14 bytes of data can be hidden within this structure, 2 bytes in **bg_pad** and 12 bytes in the **bg_reserved** variable.

The Data Mule FS tool [8] was designed to hide data in Ext2/Ext3 file systems. This tool breaks up a large file into small fragments, which are stored in reserved locations throughout a file system. To counter this tool, we developed the `rfinder` utility [17] that detects and extracts hidden data in Ext2/Ext3 file systems in a forensically sound manner.

4. Data Concealment outside File Systems

Standard forensic procedures involve seizing and imaging storage media. Individuals seeking to conceal evidence may hide data in devices that are not normally seized or in devices that are not easily imaged. This section describes how data may be concealed within random access memory, obscure hard drive locations and BIOS chips. Also, techniques for detecting and recovering hidden data are discussed.

4.1 Random Access Memory

The question of whether or not a running computer should be turned off upon seizure is a subject of debate [12, 14]. One side recommends pulling out the power cord. Another side, concerned that this procedure may damage the drive or stop the machine from completing a write operation, insists that the machine be shut down properly using the operating system. Yet another side, recognizing that valuable data might be lost during machine shutdown, recommends that information pertaining to open ports, running processes, etc. be collected while the machine is running. Each procedure has its advantages and disadvantages. However, the third procedure is based on an important observation – some key evidence may not be stored on disk.

To reduce the amount of evidence potentially recoverable from a hard drive, a malicious individual might attempt to perform most, if not all, actions in memory. A remote user could use a root kit that remains persistent in memory, and attach to a currently running process or use common utilities already on the machine to perform actions. Individuals desiring to minimize evidence of their actions on a local machine could use Knoppix [11] or other CD-bootable operating systems that do not require a hard drive or other permanent memory storage. The Tinfoil Hat Linux operating system is designed to leave no evidence pertaining to user actions: it encrypts all data written to persistent memory.

4.2 Hard Drives

A hard drive has more memory than is accessible by imaging the drive. For example, the Host Protected Area or ATA-Protected Area at the end of a hard drive cannot be read from or written to using standard operating system calls because the drive reports that it is smaller than its true capacity [5]. Forensic examiners should be aware that important evidence might be concealed in these locations. While standard tools (FTK, EnCase and ILook) cannot access this evidence, special tools, e.g., X-Ways Replica [20], are capable of detecting and recovering the hidden data.

SMART technology [16], another obscure hard drive feature, could be used by a remote hacker to determine if a victim machine has been the subject of an investigation. This technology, which is used to monitor the health of hard drives, provides information about how long a drive has been in operation. Suppose a hacker loses a connection to a victim computer for a period of time. Upon regaining the connection, the hacker could determine that the length of time that the drive has been in operation does not match the time elapsed since it was mounted. The hacker might infer that drive was imaged, and then attempt to subvert the investigation by wiping incriminating evidence on other computers.

4.3 BIOS Chips

Every computer and embedded device has a Basic Input/Output System (BIOS) chip, which is required to boot the system. A BIOS chip typically has 128K to 512K of flash memory that holds code and data. However, the chip may contain between 25K to 100K of unused space that can be used to store data without affecting the operation of the BIOS. This unused space has been exploited by virus writers and computer game enthusiasts. Malicious individuals can also use this space to hide incriminating evidence [6, 7].

Uniflash [18], a BIOS flashing utility, can be used to read and write data to a BIOS chip. Data may be written to BIOS free space and certain regions of BIOS modules (e.g., those containing error messages) without corrupting the BIOS [6, 7]. Alternatively, the entire BIOS memory may be overwritten, which, of course, renders the BIOS chip unusable [6, 7]. In this case, however, a BIOS Savior device [10] is required to boot the computer. This device provides a backup BIOS chip and a hardware switch that enables the user to select whether the computer will use the backup chip or the original BIOS chip for the booting process.

Forensic investigators must be aware that data may be hidden on a BIOS chip. They should check for utilities (e.g., Uniflash) and tools (e.g., BIOS Savior) that enable BIOS chips to be modified. It may also be necessary to conduct a forensic examination of the BIOS chip itself. Certain segments of BIOS memory can be viewed using the Windows `debug` command [6, 7]. The entire BIOS memory can be extracted using special software (e.g., AwardMod [9]) and analyzed using standard forensic tools (e.g., EnCase, FTK and ILook) [6, 7].

5. Conclusions

As digital evidence becomes increasingly important in judicial proceedings, it is logical to assume that malicious individuals will attempt to subvert investigations by targeting vulnerabilities in digital forensic procedures and tools. They will also endeavor to conceal incriminating evidence in obscure regions of file systems, in devices that are not easily imaged or in devices that are not normally seized.

This paper has two main contributions. The first is the description of the state of the art in hostile forensic techniques and tools. The second, and more important, contribution is the discussion of strategies for countering hostile forensic techniques and tools. Of particular significance are the strategies for combating subversion and denial of service attacks on forensic tools, and techniques for detecting and extracting concealed evidence. This paper has been written to raise awareness about hostile forensic techniques – and countermeasures – within the law enforcement community. We hope it will stimulate efforts within the digital forensics research and development community to ensure that all the evidence – wherever it may reside – is recoverable and presentable in court.

References

[1] `42.zip` (www.unforgettable.dk).

[2] AERAsec, Decompression bomb vulnerabilities (www.aerasec.de/security/advisories/decompression-bomb-vulnerability.html).

[3] M. Brzitwa, **gpart** (www.stud.uni-hannover.de/user/76201/gpart).

[4] R. Card, Cross-referencing Linux (lxr.linux.no/source/include/linux/ext2_fs.h?v=2.6.10).

[5] B. Carrier, *File System Forensic Analysis*, Addison-Wesley, Crawfordsville, Indiana, 2005.

[6] P. Gershteyn, M. Davis, G. Manes and S. Shenoi, Extracting concealed data from BIOS chips, in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi (Eds.), Springer, New York, pp. 217-230, 2005.

[7] P. Gershteyn, M. Davis and S. Shenoi, Forensic Analysis of BIOS chips, in *Advances in Digital Forensics II*, M. Olivier and S. Shenoi (Eds.), Springer, New York, pp. 301-314, 2006.

[8] The grugq, The art of defiling, presented at the *Hack in the Box Conference* (packetstormsecurity.nl/hitb04/hitb04-grugq.pdf), October 8, 2004.

[9] J. Hill, AwardMod (sourceforge.net/projects/awardmod), 2002.

[10] IOSS, RD1 BIOS Savior (www.ioss.com.tw), 2000.

[11] Knoppix (www.knoppix.org).

[12] W. Kruse and J. Heiser, *Computer Forensics: Incident Response Essentials*, Addison-Wesley, Boston, Massachusetts, 2002.

[13] V. Liu, Metasploit Project (www.metasploit.com/projects/antiforensics).

[14] K. Mandia and C. Prosise, *Incident Response and Computer Forensics*, McGraw-Hill/Osborne, Emeryville, California, 2003.

[15] S. McClure, J. Scambray and G. Kurtz, *Hacking Exposed: Network Security Secrets and Solutions*, McGraw-Hill/Osborne, Emeryville, California, 2001.

[16] S. McLeod, Smart anti-forensics (www.forensicfocus.com/index.php?name=Content&pid=53).

[17] S. Piper, M. Davis, G. Manes and S. Shenoi, Detecting misuse in reserved portions of Ext2/3 file systems, in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi (Eds.), Springer, New York, pp. 245-256, 2005.

[18] Rainbow Software, Uniflash (www.uniflash.org), 2005.

[19] M. Russinovich, Streams (www.sysinternals.com/Utilities/Streams.html), 2005.

[20] X-Ways Software Technology, X-Ways Replica: DOS disk cloning and imaging tool (www.x-ways.net/replica.html).