Breaking Forensics Software: Weaknesses in Critical Evidence Collection

Tim Newsham - <tim[at]isecpartners[dot]com>
Chris Palmer - <chris[at]isecpartners[dot]com>
Alex Stamos - <alex[at]isecpartners[dot]com>
Jesse Burns - <jesse[at]isecpartners[dot]com>

iSEC Partners, Inc
115 Sansome Street, Suite 1005
San Francisco, CA 94104
http://www.isecpartners.com

Version 1.1

August 1, 2007

Abstract

This article presents specific vulnerabilities in common forensics tools that were not previously known to the public, many of which were found through simple fuzzing techniques. It discusses security analysis techniques for finding vulnerabilities in forensic software, and suggests additional security-specific acceptance criteria for consumers of these products and their forensic output. Traditional testing of forensics software has focused on robustness against data hiding techniques and accurate reproduction of evidence. We also provide an analysis of a network forensic acqusition protocol, and discuss the issues with remotely acquiring forensic images. This article argues that more security focused testing, such as that performed against security-sensitive commercial software, is warranted when dealing with such critical products.

1 Introduction

This article presents specific vulnerabilities in common forensics tools that were not previously known to the public, discusses techniques for finding vulnerabilities in forensic software, and recommends additional security-specific acceptance criteria buyers should apply. The primary contribution of this work is to take an adversarial look at forensics software and apply fuzzing and vulnerability assessment common to analysis of other products, such as operating systems or office suites, to forensic software. Our findings raise some interesting legal and societal questions[1] although these issues are not explored in this paper.

Two popular software packages for performing forensic investigations on computer evidence, Guidance EnCase and Brian Carrier's The Sleuth Kit (TSK), are vulnerable to attack. The most common problems

[1] See http://www.cs.columbia.edu/~smb/blog/2007-07/2007-07-26.html for example.

are "crashers", that is, damaged data files, storage volumes, and filesystems which cause the software to crash before the forensic analyst can interpret the evidence.

We performed some random and targeted fault injection testing against the two products and uncovered several bugs, including data hiding, crashes that result in denial of service, and infinite loops that cause the program to become unresponsive.

2 Prior Art

While blind fuzzing and targeted fault injection are not new techniques, there has not been much public research into the relatively small niche of forensics software.

There is not much on EnCase or TSK in the Common Vulnerabilities and Exposures database[2], for example. Searching on "encase" in the CVE search engine returns only one result (at the time of writing, 28 June 2007), http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1578 ("EnCase Forensic Edition 4.18a does not support Device Configuration Overlays (DCO), which allows attackers to hide information without detection"). Searching CVE for "sleuth" (http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sleuth) and "sleuthkit" (http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sleuthkit) return 0 results, and a searching on "sleuth kit" returns results containing the word "kit" but not "sleuth".

There is significantly more prior art related to hiding data from forensics software or making forensics analysis more difficult. Garfinkle[6] provides an excellent overview of previous works.

3 Classes of Attacks Against Forensic Software

Forensic software is especially difficult to secure, yet must be especially robust against attack. It must acquire data from any type of device, in any format; it must parse, render, and search as many data formats as possible; and it must do all this with acceptable performance without sacrificing correctness or accuracy — in the presence both of malicious tampering and of accidental faults in the evidence.

In a forensic investigation, denial of service (DoS) vulnerabilities are no longer merely annoyances, but can impede the investigation, making evidence difficult or impossible to examine. "Crasher" bugs (often the result of the overflow of buffers on the stack or heap) are sometimes exploitable, leading to a situation in which a maliciously crafted evidence file might allow an attacker to frustrate analysis or possibly execute code on the investigator's workstation[3]. This may compromise the integrity of investigations performed on the machine or allow a lawyer to argue the point.

3.1 Data Hiding

The purpose of forensic software is to discover and analyze evidence stored on digital media. If the software fails to detect information on the medium for some reason, an attacker could exploit the weakness to hide evidence. For example, a data acquisition tool is vulnerable to a data hiding attack if it fails to

[2] http://cve.mitre.org/

[3] To be clear, we have not found any code execution vulnerabilities.

acquire the host protected area of a hard disk. Note that we do not consider cryptography a data hiding attack because while it may be uninterpretable, encrypted information is still visible.

Another type of data hiding attack is the "attack by tedium": where there exist asymmetries such that an attacker can obfuscate data more easily than the forensic investigator can unobfuscate it, the effect can be similar to a steganographic attack. This method of attack is relatively weak since it relies on the usability affordances of a particular forensic software kit. A determined investigator could defeat the attack by using multiple software kits or scripting the toolkit for automated evidence analysis, for example.

3.2 Code Execution, Evidence Corruption

Code execution vulnerabilities result from particular implementation flaws such as stack and heap overflows. Programming errors in native code may allow an attacker to specify data that overwrite control flow information in the program and provide new code of the attacker's choice to be executed in the context of the vulnerable process. When successfully executed, the attacker's code has access to system resources such as files and networking.

If an attacker succeeds in executing arbitrary code on the forensic workstation, he can cause the forensic image to become corrupted, hiding or destroying evidence. One subtle attack would be to cause the forensic toolkit not to alter the forensic image, but simply to ignore the evidence the attacker wishes to hide. There would be no obvious tip-off that an attack has occured, such as if the checksums of the evidence files changed, yet the toolkit would be instructed by the attacker never to reveal the incriminating evidence.

All the usual "mundane" attacks are of course also possible, such as planting spyware or other malware on the forensic workstation.

Bugs that allow an attacker to overwrite memory, even without arbitrary code execution, could also allow an attacker to spoil or hide evidence[4].

3.3 Denial of Service, Blocking Analysis

Denial of service vulnerabilities, which cause the program to crash or hang, frustrate forensic analysis. If an attacker hides the incriminating evidence in a file that crashes the forensic software but does not crash the attacker's own file viewer[5], the analyst must perform extra work to discover the evidence. If the analyst is diligent, the attack is a mere service downgrade — the analyst loses the rich search and parse functionality of their forensic tool. If the analyst is overworked, on a tight deadline, or lazy, they might miss important evidence.

This might seem minor, but note that the sophisticated searching, parsing, key recovery, and file carving features are the entire reason forensic software toolkits exist as a distinct class of software. Forensic analysis can of course be done with nothing but a write-blocker and the standard Unix tools, but few professional analysts would be satisfied with such a limited toolbox. Until forensic toolkits become more robust, analysts may be in the dark without knowing it.

[4] We do not know of any such defects.

[5] We observed many instances of this type of bug.

4 Techniques Used to Find Vulnerabilities

Because one of the most obvious attack surfaces on forensic software is the filesystem metadata parsing code and the rich data file parsing and rendering code, we attacked it by generating fuzzed filesystems and data files. We also made attempts to hide data by making disks with many partitions, deeply-nested archive files (TAR, ZIP, etc.), and filesystems with directory loops.

4.1 Fuzzing Data Formats

We did not need sophisticated fuzzing to discover many of the vulnerabilities. We used simple random fuzzing with no special provision for the particulars of any given data format. The fuzzer is instantiated with a random seed and provides a set of mutator functions; when fed source data, one of the mutators is chosen and invoked on the source. iSEC has a library of mutators that can

- randomly choose a substring (of random length) of the source and replace it with a random string of the same size;
- replace a random number of single bytes in the source with random bytes;
- increment or decrement a random number of single bytes in the source;
- replace a randomly-selected NUL byte or sequence of two NULs in the source with a given value of the same size;
- replace the entire source with a random string of the same size;
- overwrite 32-bit values in the source with some other given 32-bit value, advancing the replacement position on successive calls;
- delete or insert a randomly-selected substring (of random size) from the source; and
- randomly choose any two mutators and perform both mutations on the source.

When fuzzing disk and volume images we did not use the mutators that change the size of the object, since filesystems are based on fixed-size blocks with many structures expected to be aligned at particular offsets. Perturbing a filesystem too drastically tends to cause implementations to reject it completely, with no chance of bug discovery.

We performed fuzzing on several different scales. At the smallest scale we fuzzed individual files such as JPEGs, PDFs and MS Word documents. This testing was performed by creating a filesystem and populating it with many fuzzed copies of files drawn from a large collection of source files representing many common file formats. Each of these filesystem was then analyzed using the forensic analysis tools and the files were browsed and searched using any built-in viewers. The primary goal of this testing was to identify issues in the built-in file viewers. When issues were discovered, they were analyzed using GDB or WinDBG.

We performed fuzzing of filesystem structures. To perform these tests we started with a filesystem that was previously constructed and populated with a variety of files and file formats. We chose to test MSDOS, Ext2FS and NTFS filesystems since they are widely used and because of the complexity of NTFS. Then we fuzzed the entire filesystem image. This testing corrupted both file data and filesystem meta-data. The fuzzed filesystem images were then packed together as separate partitions on a disk and the entire disk

analyzed. When issues were observed, the filesystem causing the issue was isolated by analyzing subsets of partitions until a single partition was identified. Issues were further isolated on a partition by radix search: We compared the fault-causing filesystem against the unmodified filesystem image to enumerate all of the changes. We then made a disk full of partitions with each partition containing only a subset of the changes that were in the faulting image. These partitions were then analyzed and the entire process repeated until we were able to identify a minimal set of edits that still caused the issue to occur. These issues were then analyzed manually or in GDB or WinDBG.

Finally, we performed fuzzing of an entire disk. This testing was capable of corrupting disk metadata, such as partition tables, as well as filesystem metadata and file data. We performed all of our disk fuzzing using the FDISK/MBR partition scheme. Disks were created with a large number of partitions and then populated with many identical filesystem images. The entire disk image was then fuzzed and the resulting disks analyzed with the forensic tools. Analysis of any resulting defects was performed by isolating changes, manual analysis and analysis with the GDB and WinDBG debuggers.

4.2 Manual, Targeted Manipulation of Data Formats

We also performed targeted, manual mangling of data formats, such as by creating directory loops in filesystems, creating loops in MBR partition tables, creating disk images with very many partitions, tweaking data objects in JPEG files that influence memory management, and so on.

MBR partition tables. We wrote code to identify all of the MBR records and performed fuzzing on only those blocks. The fuzzing was again simplistic. The reason we did this was to increase the amount of mutations in each test case (since we can only test one disk at a time, whereas we can test many filesystems at a time, for example) without causing other issues (i.e. in the filesystem code) that might mask findings.

Directory loops. We manually edited ext2fs and NTFS filesystems so that a directory became a child subdirectory in its own directory, and then analyzed the resulting image.

Long file names. We generated filesystems with very long file names. These were created both inside a single directory and in a chain of deeply nested directories.

Large directories. We generated filesystems with a directory containing large numbers of files. These were filled with files having really short names, medium length names and long filenames.

Deeply nested directories. We generated filesystems with deeply nested directories. The directory names were very short and very long.

5 Defects Found in The Sleuth Kit

Brian Carrier's The Sleuth Kit (TSK) is a tool-oriented forensic software suite in the Unix tradition (although it does run on Windows in addition to Linux, Mac OS X, and other Unix variants). Individual programs with command-line interfaces each do one task — extracting inode metadata from a disk image, copying data referenced by an inode to a file — and to work together by reading and writing streams of text. It is implemented in C and tied together by a web interface implemented as Perl CGIs (Autopsy).

In contrast to EnCase, TSK almost completely relegates evidence display to third-party software. TSK

consists of 23 separate single-purpose programs, but we found vulnerabilities in only a few:

fls lists the file and directory names in a give filesystem image, including deleted files;

fsstat displays the metadata for the filesystem, including inode numbers and mount times;

icat copies files in the disk image by inode number (Unix filesystems) or MFT entry number (NTFS), as discovered and chosen by the investigator using e.g. the istat tool; and

istat displays the metadata stored in an inode or MFT entry.

Simple fuzzing raised several issues, and the inherent programmability of Unix-type tools allowed us to easily isolate the particular spot in damaged files that was causing the problem. Since TSK is an open source program we were able to build TSK to include symbols and use GDB to perform source level debugging. This eased the analysis of the issues we uncovered. We were also able to fix some of the minor issues and continue testing. This allowed us to reach deeper issues that may have been masked by some of our earlier findings.

In general it appears that the implementation is sufficiently careful about keeping buffer writes in bounds, but it places too much trust in the data from the disk image when reading from buffers. Most issues involve out-of-bounds reads that can lead to incorrect data, or crashes. There were also some issues that may lead to denial of service.

5.1 Data Dereferenced After Free

A crash can occur when processing a corrupted ext2fs image because the error processing code dereferences data after it frees it. In ext2fs.c:

```
1230      for (n = 0; length > 0 && n < inode->direct_count; n++) {
1231          read_b = ext2fs_file_walk_direct(fs, buf, length,
1232              inode->direct_addr[n], flags, action, ptr);
1233
1234          if (read_b == -1) {
1235              free(buf);
1236              data_buf_free(buf[0]);
1237              return 1;
1238          }
```

If *read_b* is -1 (line 1234) then *buf* is freed (line 1235) before data in *buf*[0] is freed (line 1236). This leads to a crash on some systems.

5.1.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 7616332 \x01
$ icat Bad.dsk 56-128-3
```

5.2 Corrupted NTFS image Causes icat to Run Indefinitely

icat runs virtually forever when run on some altered NTFS images. It appears that a 64-bit value was read off the disk and used as a byte count. We observed the following in *gdb*:

```
#9  0x080892ef in ntfs_data_walk (ntfs=0x80ee0c0, inum=56, fs_data=0x80f0400,
    flags=0, action=0x80a45d0 <icat_action>, ptr=0x0) at ntfs.c:1639
1639                retval = action(fs, addr, buf, bufsize, myflags, ptr);
```

```
(gdb) p/x fsize
\$3 = 0xffffffffff8ecd42
(gdb) p/x fs_data->size
\$4 = 0x9342
(gdb) p/x fs_data->runlen
\$5 = 0xffffff0600009200
```

5.2.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 7616332 \x01
$ icat Bad.dsk 56-128-3
```

5.3 NTFS Image Causes icat to Crash

icat crashes while processing a file on a corrupted NTFS filesystem image. The crash occurs when dereferencing *fs_data_run* at line 1570 of ntfs.c:

```
1543         /* cycle through the number of runs we have */
1544         while (fs_data_run) {
1545
1546             /* We may get a FILLER entry at the beginning of the run
1547              * if we are processing a non-base file record because
1548              * this $DATA attribute could not be the first in the bigger
1549              * attribute. Therefore, do not error if it starts at 0
1550              */
1551             if (fs_data_run->flags & FS_DATA_FILLER) {
     [... code elided ...]
1564                 }
1565                 else {
1566                     fs_data_run = fs_data_run->next;
1567                 }
1568             }
1569
1570             addr = fs_data_run->addr;
```

The check at line 1544 ensures that *fs_data_run* is non-NULL, however line 1566 updates the variable without returning to the check (with a continue) or performing the check again. This leads to a NULL dereference at line 1570.

5.3.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 7567157 AA
$ icat Bad.dsk 8-128-1
```

5.4 NTFS Image Causes fls to Crash (1)

fls crashes while listing files from a corrupted image. The crash occurs when copying data in *fs_data_put_str*. The calling code is in ntfs.c:

```
1778         /* Add this resident stream to the fs_inode->attr list */
1779         fs_inode->attr =
1780         fs_data_put_str(fs_inode->attr, name, type,
1781         getu16(fs->endian, attr->id),
1782         (void *) ((uintptr_t) attr +
1783             getu16(fs->endian,
1784             attr->c.r.soff)), getu32(fs->endian,
1785             attr->c.r.ssize));
```

The read buffer is *attr* plus an arbitrary 16-bit offset and the read length is an arbitrary 32-bit length. No bounds checking is performed to ensure that the buffer contains as many bytes as are requested.

5.4.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 7652939 \x01
$ fls -rlp Bad.dsk
```

5.5 NTFS Image Causes fls to Crash (2)

fls crashes while listing files from a corrupted image. The crash occurs at line 208 of ntfs_dent.c while dereferencing *idxe*:

```
205         /* perform some sanity checks on index buffer head
206          * and advance by 4-bytes if invalid
207          */
208         if ((getu48(fs->endian, idxe->file_ref) > fs->last_inum) ||
209             (getu48(fs->endian, idxe->file_ref) < fs->first_inum) ||
210             (getu16(fs->endian, idxe->idxlen) <= getu16(fs->endian,
211                     idxe->strlen))
212             || (getu16(fs->endian, idxe->idxlen) % 4)
213             || (getu16(fs->endian, idxe->idxlen) > size)) {
214             idxe = (ntfs_idxentry *) ((uintptr_t) idxe + 4);
215             continue;
216         }
```

This code is executed in a loop while walking a table and advancing *idxe*. The variable is initially in range, but is eventually moved out of range because the caller passes in a large size (line 735) which is an arbitrary 32-bit integer taken from the filesystem image. No bounds checking is performed to ensure that the value is in range.

```
734     retval = ntfs_dent_idxentry(ntfs, dinfo, list_seen, idxe,
735         getu32(fs->endian,
736             idxelist->buf_off) -
737         getu32(fs->endian, idxelist->begin_off),
738         getu32(fs->endian,
739             idxelist->end_off) -
740         getu32(fs->endian, idxelist->begin_off), flags, action, ptr);
```

5.5.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 7653298 d
$ fls -rlp Bad.dsk
```

5.6 NTFS Image Causes fsstat to Crash

fstat crashes while processing a corrupted filesystem image. The crash happens when dereferencing *sid* in line 2714 of ntfs.c:

```
2703     unsigned int owner_offset =
2704         getu32(fs->endian, sds->self_rel_sec_desc.owner);
2705     ntfs_sid *sid =
2706         (ntfs_sid *) ((uint8_t *) & sds->self_rel_sec_desc +
                          owner_offset);
2707
2708     // "1-"
2709     sid_str_len += 2;
2710     //tsk_fprintf(stderr, "Revision: %i\n", sid->revision);
2711
2712     // This check helps not process invalid data, which was noticed
           while testing
2713     // a failing harddrive
2714     if (sid->revision == 1) {
```

This variable is computed with an arbitrary 32-bit offset (line 2704) from an existing buffer (line 2706) without any bounds checking.

5.6.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 10667177 \x9b
$ fstat Bad.dsk
```

5.7 NTFS Image Causes fsstat to Crash

fstat crashes while processing a corrupted filesystem image. The crash happens while copying data from the *sds* variable (line 2831) in ntfs.c. Care is taken to ensure that no out of bounds reads occur with checks at line 2817. However, these checks are based on the offset computed in terms of 32-bit values (line 2809) not in terms of bytes. The check assumes that the *current_offset* is actually a byte count, which would be four times larger.

```
2808      while (total_bytes_processed < sds_buffer->size) {
2809          current_offset =
2810              (uintptr_t *) sds - (uintptr_t *) sds_buffer->buffer;
2811
2812          offset = getu32(fs->endian, sds->ent_size);
2813          if (offset % 16) {
2814              offset = ((offset / 16) + 1) * 16;
2815          }
2816
2817          if ((offset != 0) && (offset < (sds_buffer->size -
                                              current_offset))
2818              && (getu64(fs->endian, sds->file_off) <
                                       sds_buffer->size)) {
2819
2820              NTFS_SDS_ENTRY *sds_entry;
2821
2822              if ((sds_entry =
2823                       (NTFS_SDS_ENTRY *)
                         mymalloc(sizeof(NTFS_SDS_ENTRY))) ==
2824                  NULL) {
2825                  return 1;
2826              }
2827              if ((sds_entry->data = (uint8_t *)
                                         mymalloc(offset)) == NULL) {
2828                  free(sds_entry);
2829                  return 1;
2830              }
2831              memcpy(sds_entry->data, sds, offset);
```

5.7.1 Reproduction

```
$ patch.py NtfsPart.dsk Bad.dsk 10666450 \x01
$ fsstat Bad.dsk
```

6 Defects Found in Guidance EnCase

EnCase from Guidance Software is a very different beast from TSK. It runs only on Windows[6] features a sophisticated GUI, and incorporates features for browsing, searching and displaying devices, filesystems, and data files. For programmability, it incorporates its own programming language, Enscript, that resembles C++ and Java.

[6]Although it comes with a remote acquisition component for linux.

As with TSK, EnCase showed numerous defects with relatively simple fuzzing techniques, although we also created targeted, domain-specific faults in test data, such as carefully crafted partition tables and NTFS directory structures.[7]

We tested EnCase versions 6.2 and 6.5. Guidance has stated that many of the issues we have identified will be addressed in EnCase 6.7.

When issues were found they were analyzed by isolating the minimal set of changes to that caused the issue, manually identifying the semantics of those changes, and analyzing any resulting crashes in WinDBG. Because EnCase is a closed, commercial system, analysis was performed without access to source code or debugging symbols. EnCase's use of code obfuscation and anti-debugging features also made analysis more difficult than it might otherwise have been. At times we noticed that EnCase caught and ignored runtime exceptions that were causing crashes and we had to attach a debugger while performing testing in order to analyze defects rather than rely on post mortem debugging. Without access to source code we were not easily able to patch defects as we uncovered them. This may result in some unfound defects being masked by other defects that occur earlier during analysis.

6.1 Disk Image With Corrupted MBR Partition Table Cannot Be Acquired

EnCase cannot properly acquire disks with certain corrupted MBR partition tables. When running linen on a system with a disk with a carefully crafted partition table (including many partition table entries), linen won't start up properly. If linen is started prior to corrupting the image, it will start up, but EnCase will hang indefinitely while acquiring the image. While the acquisition is hung, it is possible to cancel out of the import from the GUI.

If a disk image is manually captured and transferred to the EnCase workstation and acquired as a raw disk image, EnCase will hang indefinitely while attempting to acquire the image. There is no way to cancel out of this process — the GUI becomes unresponsive. We have not identified the root cause of this issue, but it appears to be due to the overly large values in the 29th partition table entry. We were unable to reproduce this issue in similar situations with a small number of partitions.

The partition table for MbrInfiniteLoop.dsk, as reported by the Linux fdisk utility, is:

```
Disk /dev/hdb: 1073 MB, 1073741824 bytes
16 heads, 63 sectors/track, 2080 cylinders
Units = cylinders of 1008 * 512 = 516096 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hdb1               1        2080     1048288+   5  Extended
/dev/hdb5               1          44       22113    b  W95 FAT32
/dev/hdb6              45          88       22144+   b  W95 FAT32
/dev/hdb7              89         132       22144+   b  W95 FAT32
/dev/hdb8             133         176       22144+   b  W95 FAT32
/dev/hdb9             177         220       22144+   b  W95 FAT32
/dev/hdb10            221         264       22144+   b  W95 FAT32
/dev/hdb11            265         308       22144+   b  W95 FAT32
/dev/hdb12            309         352       22144+   b  W95 FAT32
/dev/hdb13            353         396       22144+   b  W95 FAT32
/dev/hdb14            397         440       22144+   b  W95 FAT32
/dev/hdb15            441         484       22144+   b  W95 FAT32
/dev/hdb16            485         528       22144+   b  W95 FAT32
/dev/hdb17            529         572       22144+   b  W95 FAT32
/dev/hdb18            573         616       22144+   b  W95 FAT32
/dev/hdb19            617         660       22144+   b  W95 FAT32
/dev/hdb20            661         704       22144+   b  W95 FAT32
```

[7] We noticed instances where EnCase's remote acquisition tool for Linux, linen, could not process a corrupted disk image if linen was started up after the filesystem was corrupted. Since our primary goal was to test EnCase, not linen, we worked around these issues by running linen prior to corrupting a disk image without further analyzing the issues in linen.

```
/dev/hdb21             705          748       22144+   b  W95 FAT32
/dev/hdb22             749          792       22144+   b  W95 FAT32
/dev/hdb23             793          836       22144+   b  W95 FAT32
/dev/hdb24             837          880       22144+   b  W95 FAT32
/dev/hdb25             881          924       22144+   b  W95 FAT32
/dev/hdb26             925          968       22144+   b  W95 FAT32
/dev/hdb27             969         1012       22144+   b  W95 FAT32
/dev/hdb28            1013         1056       22144+   b  W95 FAT32
/dev/hdb29         2098209      2098252       22144+   b  W95 FAT32
/dev/hdb30            1101         1144       22144+   b  W95 FAT32
/dev/hdb31            1145         1188       22144+   b  W95 FAT32
/dev/hdb32            1189         1232       22144+   b  W95 FAT32
/dev/hdb33            1233         1276       22144+   b  W95 FAT32
/dev/hdb34            1277         1320       22144+   b  W95 FAT32
/dev/hdb35            1321         1364       22144+   b  W95 FAT32
/dev/hdb36            1365         1408       22144+   b  W95 FAT32
/dev/hdb37            1409         1452       22144+   b  W95 FAT32
/dev/hdb38            1453         1496       22144+   b  W95 FAT32
/dev/hdb39            1497         1540       22144+   b  W95 FAT32
/dev/hdb40            1541         1584       22144+   b  W95 FAT32
/dev/hdb41            1585         1628       22144+   b  W95 FAT32
/dev/hdb42            1629         1672       22144+   b  W95 FAT32
/dev/hdb43            1673         1716       22144+   b  W95 FAT32
/dev/hdb44            1717         1760       22144+   b  W95 FAT32
/dev/hdb45            1761         1804       22144+   b  W95 FAT32
/dev/hdb46            1805         1848       22144+   b  W95 FAT32
/dev/hdb47            1849         1892       22144+   b  W95 FAT32
/dev/hdb48            1893         1936       22144+   b  W95 FAT32
/dev/hdb49            1937         1980       22144+   b  W95 FAT32
/dev/hdb50            1981         2024       22144+   b  W95 FAT32
/dev/hdb51            2025         2068       22144+   b  W95 FAT32
```

The corrupted partition table entry corresponding to hdb29 is:

```
Block 10200961 slot 1
      flag=0 shead=15 scs=(255, 255) type=0x05 ehead=15 ecs=(255, 255)
      start=1064385 len=44352
Block 10200961 slot 2
      flag=0 shead=0 scs=(0, 0) type=0x00 ehead=0 ecs=(0, 0)
      start=0 len=0
Block 10200961 slot 3
      flag=0 shead=0 scs=(0, 0) type=0x00 ehead=0 ecs=(0, 0)
      start=0 len=0
Block 10200961 slot 4
      flag=0 shead=15 scs=(255, 255) type=0x0b ehead=15 ecs=(255, 255)
      start=2113929279 len=44289
```

6.2 Corrupted NTFS Filesystem Crashes EnCase During Acquisition

EnCase crashes while acquiring certain corrupted NTFS partitions. The crash occurs when EnCase processes FILE records that contain a larger-than-expected offset to update sequence value, causing it to read past the end of a buffer, resulting in a read access violation. Here is an example FILE record that causes the crash.

```
0073B600 46 49 4C 45 30 61 03 00 D9 11 10 00 00 00 00 00 FILE0a..........
0073B610 01 00 01 00 38 00 0D 00 70 02 00 00 00 04 00 00 ....8...p.......
0073B620 00 00 00 00 00 00 00 00 04 00 00 00 18 00 00 00 ................
```

The modified byte, is 0x61 (“a”). The original value was 0x00. This makes the offset 0x6130 rather than the normal value of 0x0030. Figure 1 shows WinDbg just after the crash occured.

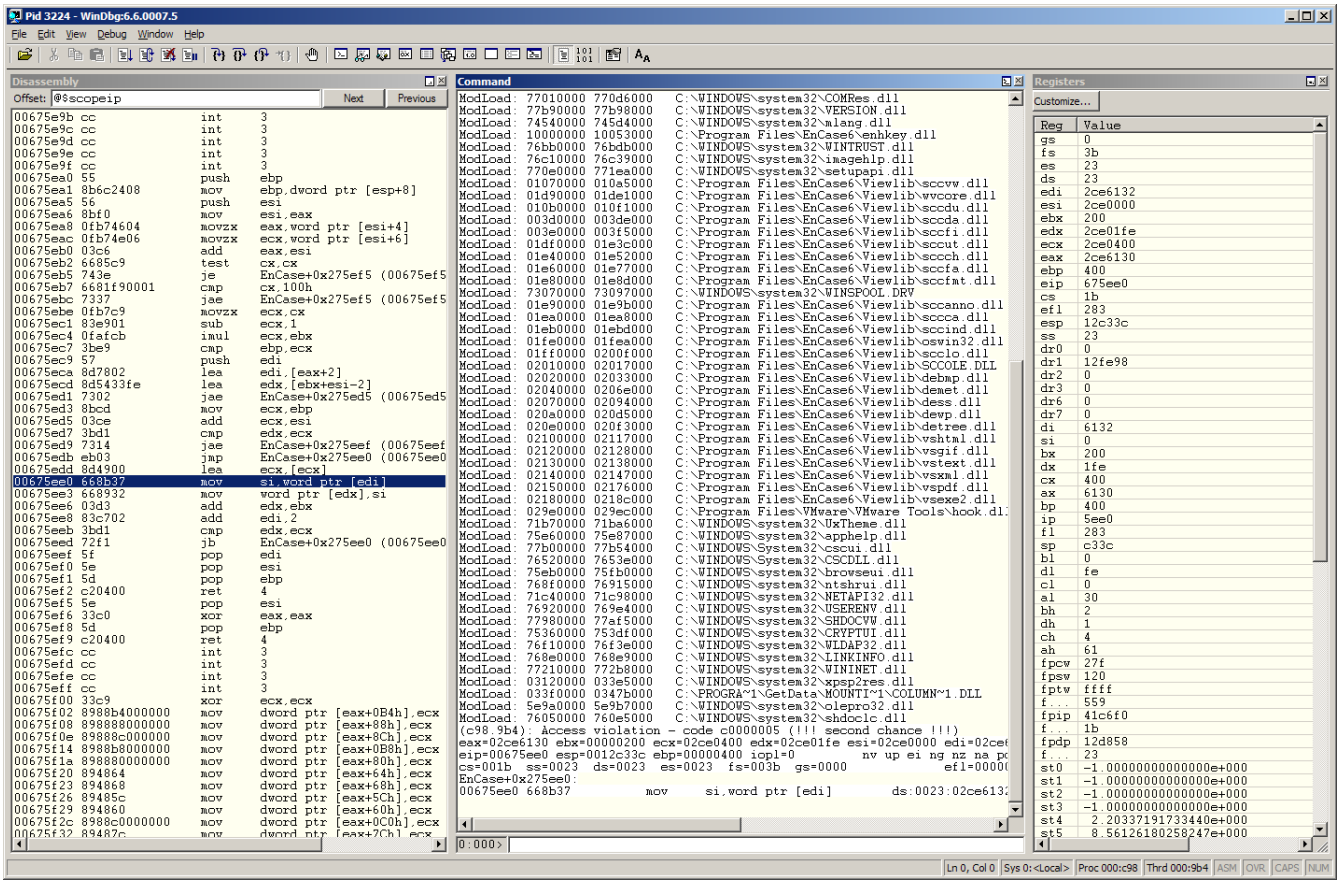

Figure 1: The postmortem debugger after EnCase crashed while acquiring a corrupted NTFS filesystem. The access violation occurs when the *mov* instruction at eip = 0x00675ee0 ($EnCase$ + 00275ee0) attempts to dereference *edi*, which is under partial control of the attacker (the base 0x02ce0000 + 0x6130 from the FILE record).

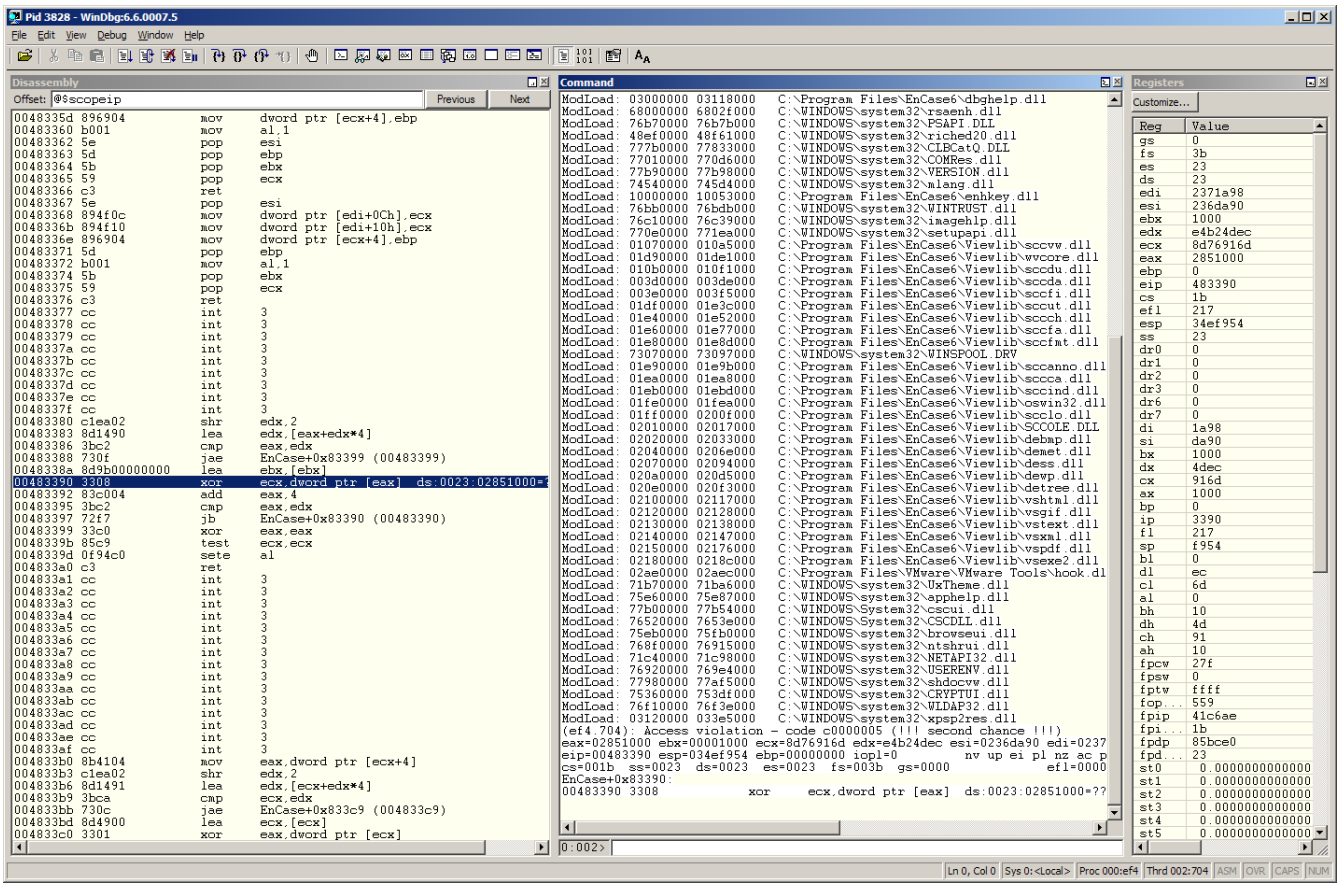

Figure 2: The postmortem debugger after EnCase crashed while searching/analysing a filesystem containing corrupted Microsoft Exchange database. *eax* varies from run to run.

6.3 Corrupted Microsoft Exchange Database Crashes EnCase During Search and Analysis

EnCase crashes while searching/analyzing a filesystem containing a corrupted Microsoft Exchange database, as seen in Figure 2. The crash occurs during the searching phase of an acquisition in which all Search, Hash and Signature Analysis options were enabled. The crash appears to be a read access violation with a bad value in *eax* that is dereferenced, but the exact value in *eax* appears to change from run to run. We have not determined the cause of or full implications of this problem.

6.4 Corrupted NTFS Filesystem Causes Memory Allocation Error

EnCase reports memory allocation errors when acquiring corrupted NTFS images. The size of memory being allocated is under the control of the attacker. iSEC has not found any ill effects caused by this error condition other than an error being displayed and corrupted records not being displayed.

The following shows the hex dump of the corrupted FILE record. The bytes in blue are the attribute types (four bytes) and lengths (four bytes). The byte in red is the byte that has been altered.

```
0073CA00 46 49 4C 45 30 00 03 00 35 40 11 00 00 00 00 00 FILE0...5@......
0073CA10 01 00 01 00 38 00 03 00 58 02 00 00 00 04 00 00 ....8...X.......
0073CA20 00 00 00 00 00 00 00 00 06 00 00 00 1D 00 00 00 ................
0073CA30 02 00 00 00 00 00 00 00 10 00 00 00 60 00 00 00 ............`...
0073CA40 00 00 00 00 00 00 00 00 48 00 00 00 18 00 00 00 ........H.......
0073CA50 38 EC 47 80 6D 5B C7 01 AE 50 1D 85 6D 5B C7 01 8.G.m[...P..m[..
0073CA60 AE 50 1D 85 6D 5B C7 01 AE 50 1D 85 6D 5B C7 01 .P..m[...P..m[..
0073CA70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0073CA80 00 00 00 00 04 01 00 00 00 00 00 00 00 00 00 00 ................
0073CA90 00 00 00 00 00 00 00 00 30 00 00 00 68 00 00 00 ........0...h...
0073CAA0 00 00 00 00 00 00 02 00 4C 00 00 00 18 00 01 00 ........L.......
0073CAB0 05 00 00 00 00 00 05 00 38 EC 47 80 6D 5B C7 01 ........8.G.m[..
0073CAC0 38 EC 47 80 6D 5B C7 01 38 EC 47 80 6D 5B C7 01 8.G.m[..8.G.m[..
0073CAD0 38 EC 47 80 6D 5B C7 01 00 00 00 00 00 00 00 00 8.G.m[..........
0073CAE0 00 00 00 00 00 00 00 00 00 00 00 00 10 00 00 00 ................
0073CAF0 05 03 66 00 69 00 6C 00 65 00 73 00 00 00 00 00 ..f.i.l.e.s.....
0073CB00 90 00 00 00 D0 00 00 00 00 00 04 18 00 00 00 05 ................
0073CB10 B0 00 00 00 20 00 00 00 24 00 49 00 33 00 30 00 ....  ...$.I.3.0.
0073CB20 30 00 00 00 01 00 00 00 00 00 10 00 00 08 00 00 0...............
0073CB30 10 00 00 00 A0 00 00 00 A0 00 00 00 01 00 00 00 ................
0073CB40 41 00 00 00 00 00 01 00 78 00 5A 00 01 00 00 00 A.......x.Z.....
0073CB50 1D 00 00 00 00 00 01 00 00 F2 9D 3C D5 D6 C6 01 ...........<....
0073CB60 00 F2 9D 3C D5 D6 C6 01 44 36 59 82 6D 5B C7 01 ...<....D6Y.m[..
0073CB70 00 F2 9D 3C D5 D6 C6 01 00 0E 00 00 00 00 00 00 ...<............
0073CB80 25 0D 00 00 00 00 00 00 20 00 00 00 00 00 00 00 %.......  .......
0073CB90 0C 02 48 00 45 00 4C 00 4C 00 4F 00 57 00 7E 00 ..H.E.L.L.O.W. .
0073CBA0 31 00 2E 00 58 00 4D 00 4C 00 00 00 00 00 00 00 1...X.M.L.......
0073CBB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0073CBC0 18 00 00 00 03 00 00 00 08 00 00 00 00 00 00 00 ................
0073CBD0 A0 00 00 00 59 00 00 00 01 04 40 00 00 00 03 00 ....Y.....@.....
0073CBE0 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
0073CBF0 48 00 00 00 00 00 00 00 00 00 20 00 00 00 00 02 00 H.......  ......
0073CC00 00 20 00 00 00 00 00 00 00 00 20 00 00 00 00 00 00 .  .......  ......
0073CC10 24 00 49 00 33 00 30 00 21 08 B0 56 21 08 DF 29 $.I.3.0.!..V!..)
0073CC20 00 30 58 E1 E0 8E E1 F7 B0 00 00 00 28 00 00 00 .0X.........(...
0073CC30 00 04 18 00 00 00 04 00 08 00 00 00 20 00 00 00 ............  ...
0073CC40 24 00 49 00 33 00 30 00 03 00 00 00 00 00 00 00 $.I.3.0.........
0073CC50 FF FF FF FF 82 79 47 11 2E 00 61 00 72 00 6A 00 .....yG...a.r.j.
0073CC60 22 00 00 00 00 00 01 00 68 00 56 00 00 00 00 00 ".......h.V.....
0073CC70 1D 00 00 00 00 00 01 00 00 30 B2 30 D5 D6 C6 01 .........0.0....
0073CC80 00 30 B2 30 D5 D6 C6 01 FC 0E AC 80 6D 5B C7 01 .0.0........m[..
```

The modified byte changes the attribute length of the 0xA0 attribute from its original value of 0x58 to its new value of 0x59. If the value was left at its original value, the bytes in green would start a new attribute of type 0xB0 and length 0x28. Instead the attribute starts one byte later and is attribute 0x28000000 with length 0x00. The values in purple, 00 61 00 72, form the size that will be allocated by EnCase, 0x72006100. EnCase will report a memory allocation error since it cannot allocate this much memory. These values can be changed arbitrarily and EnCase will allocate that size, if small enough, or report a memory allocation error with that size, if it is too large.

We are not familiar with what meaning EnCase is attempting to give to these bytes in this context, and

we are unaware of any exploitable condition that arises from this error.

6.5 EnCase and Linux Interpret NTFS Filesystems Differently

EnCase and Linux appear to use different NTFS metadata when parsing directory structures. We created an NTFS image with a directory loop in it by modifying an NTFS filesystem and replacing a directory entry for a file with a reference to the directory's parent directory. When mounting this directory in Linux[8], the modification was as expected and a directory loop was present. When importing this image into EnCase, the loop was not present and the original file was still present in the directory — but EnCase did not make other files in the directory visible.

This difference in behavior can be used by an attacker to hide data on a disk. An NTFS image can be constructed that has one interpretation on Linux and another in EnCase.

We manually edited an NTFS image to create a directory loop. This directory loop was visible in Linux (when using the NTFS-3g driver) but to our surprise, EnCase did not see the edits we made. Instead it displayed the unedited file. This indicates that EnCase and Linux give different interpretation to NTFS images, probably by using different parts of the redundant information stored in the filesystem. An attacker could abuse this inconsistency to hide data that could only be viewed in Linux and not in EnCase.

To create the directory loop, we started with a normal NTFS disk image. We located the directory entry for one of the files (readme.txt) in one of the subdirectories and replaced the directory entry with a directory entry for the directory itself (which can be found in the parent directory). The original, unedited, directory entry for readme.txt was:

```
0128CA00 49 4E 44 58 28 00 09 00 75 F4 10 00 00 00 00 00 INDX(...u.......
0128CA10 00 00 00 00 00 00 00 00 28 00 00 00 C8 04 00 00 ........(.......
0128CA20 E8 0F 00 00 00 00 00 00 02 00 01 00 74 00 00 00 ............t...
0128CA30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0128CA40 5D 00 00 00 00 00 01 00 68 00 56 00 00 00 00 00 ].......h.V.....
0128CA50 5C 00 00 00 00 00 01 00 00 79 31 40 D5 D6 C6 01 \........y1@....
0128CA60 00 79 31 40 D5 D6 C6 01 8E 94 5E 84 6D 5B C7 01 .y1@......^.m[..
0128CA70 00 79 31 40 D5 D6 C6 01 00 04 00 00 00 00 00 00 .y1@............
0128CA80 13 03 00 00 00 00 00 00 20 00 00 00 00 00 00 00 ........  .......
0128CA90 0A 03 72 00 65 00 61 00 64 00 6D 00 65 00 2E 00 ..r.e.a.d.m.e...
0128CAA0 74 00 78 00 74 00 00 00 5E 00 00 00 00 00 01 00 t.x.t...^.......
0128CAB0 68 00 58 00 00 00 00 00 5C 00 00 00 00 00 01 00 h.X.....\.......
0128CAC0 00 A6 62 41 D5 D6 C6 01 00 A6 62 41 D5 D6 C6 01 ..bA......bA....
0128CAD0 E8 F6 60 84 6D 5B C7 01 00 A6 62 41 D5 D6 C6 01 ..‘.m[....bA....
0128CAE0 00 34 01 00 00 00 00 00 80 33 01 00 00 00 00 00 .4.......3......
0128CAF0 20 00 00 00 00 00 00 00 0B 03 53 00 47 00 49 00  .........S.G.I.
0128CB00 55 00 4C 00 39 00 38 00 2E 00 64 00 72 00 76 00 U.L.9.8...d.r.v.
0128CB10 5F 00 00 00 00 00 01 00 68 00 58 00 00 00 00 00 _.......h.X.....
```

The colored sections represent the directory entry. The red sections are the edited fields which are the file reference (5d 00 00 00 00 00 01 00) the actual size and real size (0x0400 and 0x0313 respectively) and the flags (20 00 00 00 00 00 00 00). These values were changed to match the values for the parent directory: file reference of 5c 00 00 00 00 00 01 00, actual and real sizes of zero and flags of 00 00 00 01 00 00 00 00.

[8] iSEC used the NTFS-3g Linux driver: http://www.ntfs-3g.org/.

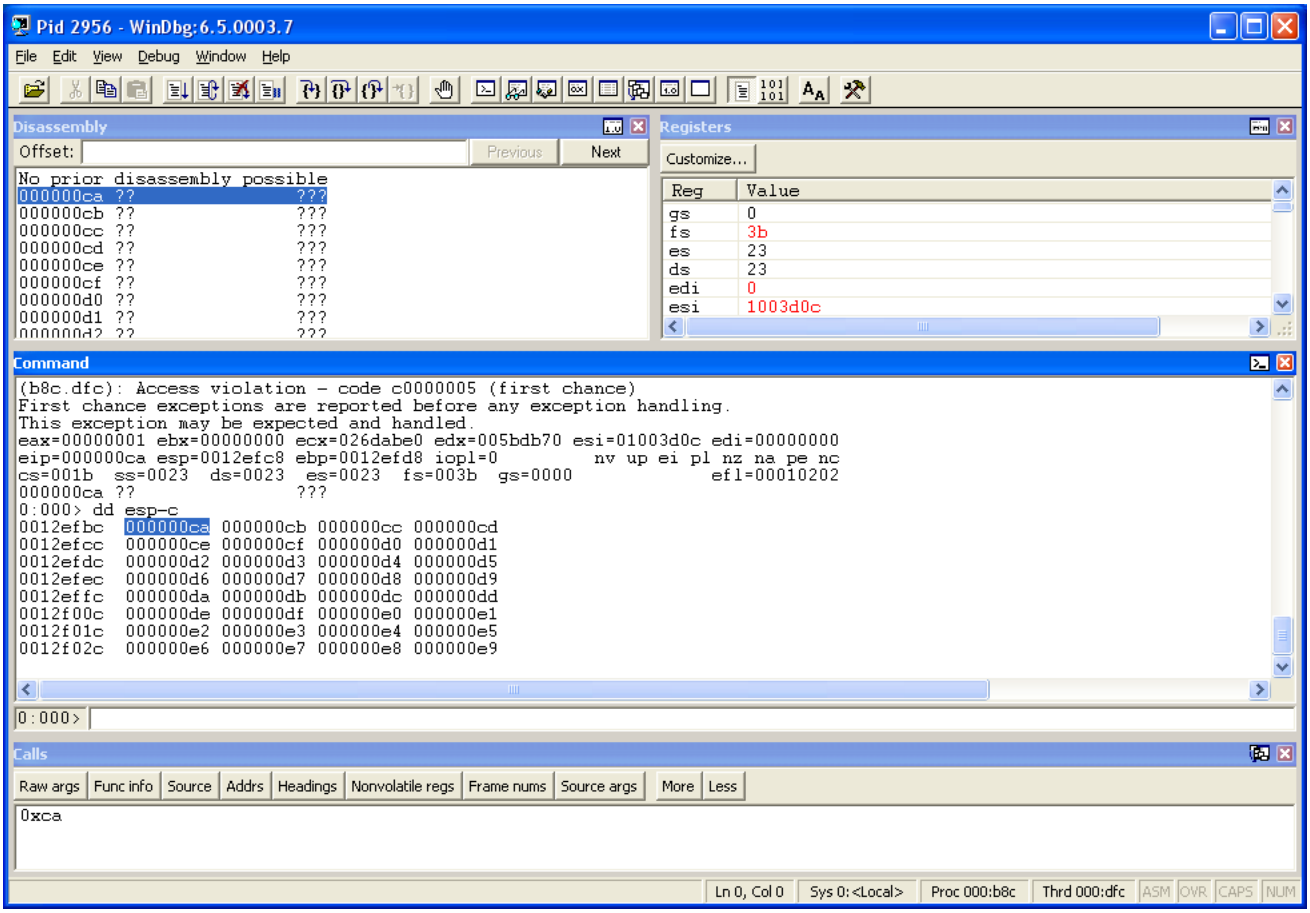

Figure 3: This screenshot shows the program context after executing the expand all action on the tree view for the files in a deeply nested NTFS directory. Notice that the current *eip* contains the value that appears below the stack pointer. This most likely occurred because a return address was executed after the return address on the stack had been overwritten.

6.6 EnCase Crashes When Viewing Certain Deeply Nested Directories

We created NTFS images with very deeply nested directories and observed that EnCase would crash in different ways after the image was acquired when performing the Expand All action, or when manually expanding the subdirectory views in the file browsing GUI. Some of these crashes were caused when the program used a return address on the stack that had been overwritten. The values being written to the stack were small integers. While were able to manipulate the value of these integers to some degree, we were unable to exploit this flaw for arbitrary code execution.

We suspect that the buffer overflow occurs because of an optimization that avoids using recursion to descend into subdirectories if there is only one more subdirectory to visit in a directory. It appears that when many subdirectories are present in a directory, this condition cannot occur. When there is long chain of directories each containing only one subdirectory, this condition can occur. By manipulating the number of directories that have one child and the number of directories that have more than one child, an attacker can gain some limited control over what integers are being written to the stack. See Figure 3.

6.7 Simple Data Hiding With Too Many Disk Partitions

We found that EnCase only shows the first 25 partitions on a volume, naming them either "`C:`" through "`Z:`", and with an additional partition named "["; or "hdb5" through "hdb25" (as shown in Figures 4, and 5). Additional partitions are not listed in the device window, and are only searchable with EnCase's various search features, but not browseable. Linux, however, allows us to create disks with 56 extended partitions and to use them all.

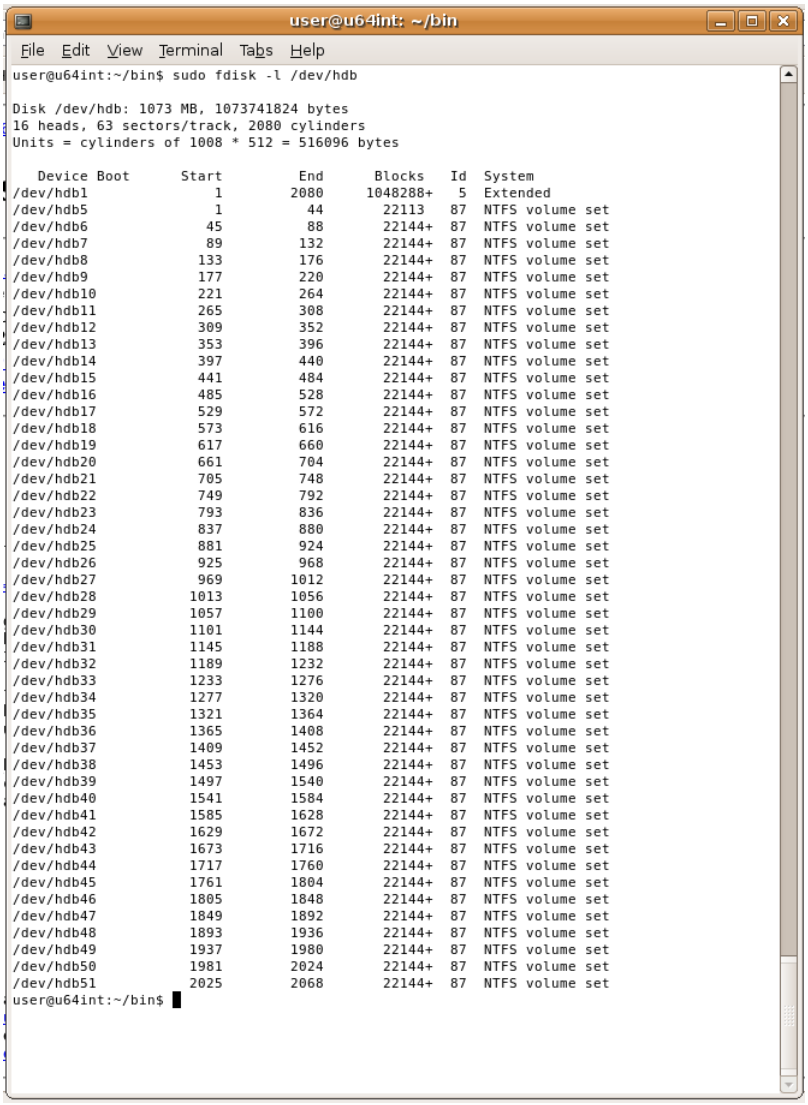

Figure 4: The output of fdisk on Linux shows that /dev/hdb has many (more than 25) partitions.

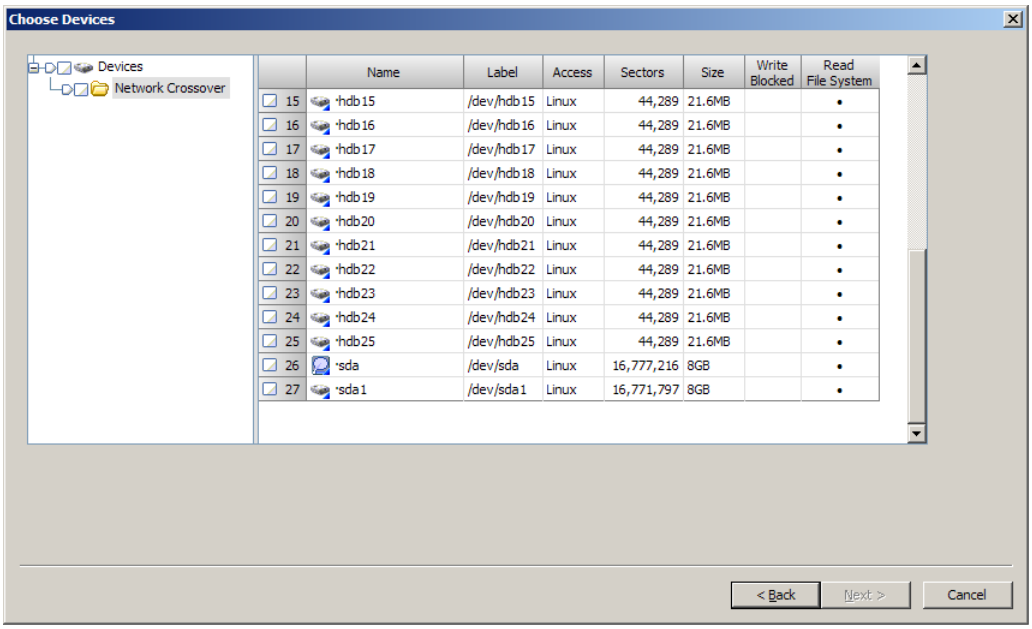

Figure 5: EnCase cannot see more than 21 of the partitions on /dev/hdb. In this example, the device is added to the case with the "network crossover" feature, using linen on the Linux computer. Similar results are seen when acquiring from a raw image.

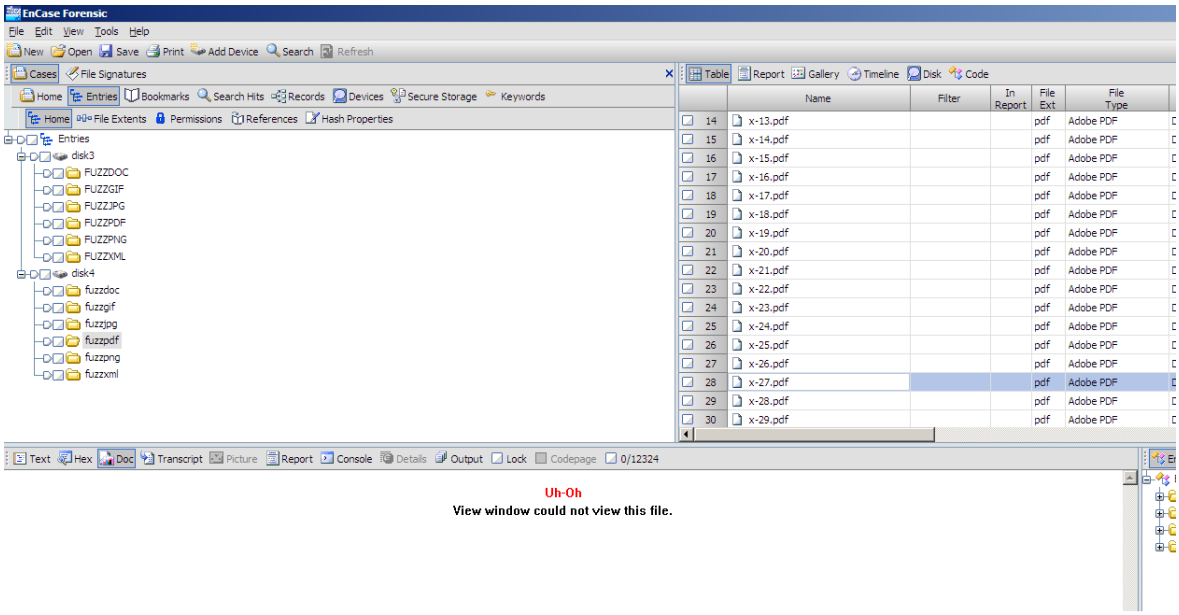

Figure 6: EnCase cannot display x-27.pdf, a fuzzed copy of a software manual file.

6.8 Problems With the Internal File Viewers

EnCase makes use of DLLs from Stellent[9] to render various file types inside the EnCase view pane. This feature is not reliable, resulting in occasional crashes and the occasional inability to read files that third-party tools are able to display.

6.8.1 EnCase Cannot Display Certain Files that Third-Party Tools Can

This problem, while not a security flaw, is indicative of the unreliability of the file viewing feature. We found that EnCase was sometimes able to display fuzzed PDFs, was sometimes unable to, and sometimes crashed. (The crashes were not always in the same place in the code, even on the same input files.) Figure 6 illustrates this problem. Note that simply double-clicking on the file in EnCase's upper-right window pane will cause the operating system's default viewer for the given file type to be launched, making this problem minor.

6.8.2 EnCase Occasionally Crashes on Fuzzed PDFs

EnCase sometimes can display fuzzed PDFs, sometimes cannot display them, and sometimes crashes trying to display them. Figure 7 illustrates this problem.

[9] http://www.stellent.com/en/index.htm

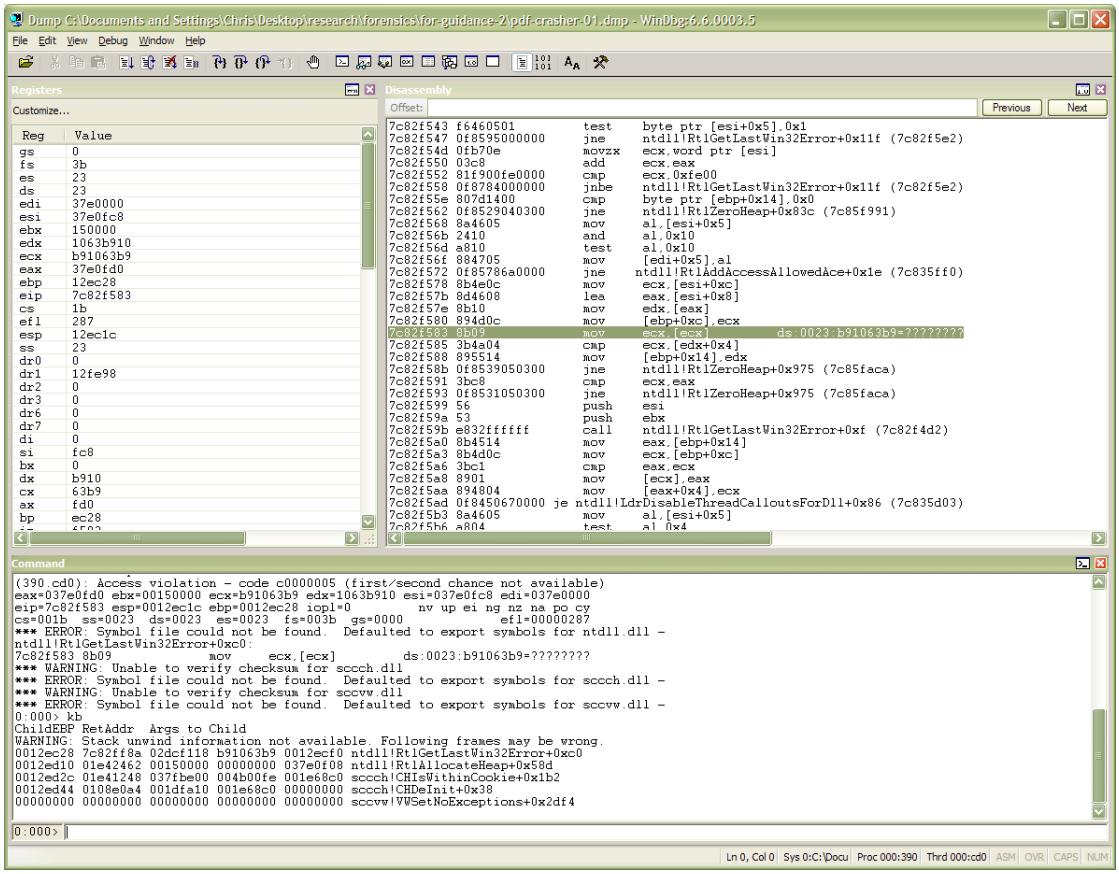

Figure 7: An example crash dump from when EnCase tried to render a PDF.

7 Attacks against EnCase Enterprise Evidence Collection

Guidance Software sells several versions of its EnCase product, with one of the more extensive packages being the EnCase Enterprise Edition (EEE). EnCase Enterprise uses the core EnCase Forensic product as the basis for the examination and analysis of captured hard disk and memory images. The added value in the Enterprise package comes from the addition of remote network imaging of systems, a feature which is especially valuable to large enterprises that perform many investigations of their production or employee systems during the year.

Although the use model for EEE varies considerably from the typical "seize, image, and analyze" work flow of a law enforcement forensics examiner, the same goals for the integrity and admissibility of the evidence collected, as well as robustness against attack, should apply to EEE. Guidance Software publishes a document entitled "Evidentiary Authentication within the EnCase Enterprise Process"[10], which addresses the legal issues involved in the use of evidence gathered with EEE. This document, which briefly outlines the EEE security schema and analyzes relevant case law, makes this conclusion:

> "EnCase Enterprise is ideally suited to recover and authenticate data over a local or wide area network. In many cases, the use of EnCase Enterprise is more that merely acceptable, but in fact constitutes best practices. EnCase Enterprise maintains the integrity of the files being examined, and cannot write to the target drive. Its robust security infrastructure disallows unauthorized access and securely logs and identifies all users and activity throughout the course of the examination through a secure server. This framework establishes EnCase Enterprise as an unparalleled process for maintaining and documenting data acquired in a remote and live network environment."

7.1 Description of EnCase Enterprise

EnCase Enterprise is comprised of these three components:

EnCase Examiner This is the application used by examiners to capture and analyze computer systems. According to Guidance Software[11], it is based upon the EnCase Forensic Edition product with the addition of the network capture functionality. Each licensed copy of EnCase Examiner contains a certificate and private key tied to the individual user.

EnCase SAFE The Secure Authentication for EnCase (SAFE) product is a network server that provides authentication and access control services, and which mediates encrypted connections between the Examiner and Servlet. The SAFE is tied to the customer's enterprise license, and contains a keypair signed by Guidance's root keypair. The SAFE provides an interface for authorizing Examiner keypairs and defining the rights granted to authenticated users. During the system acquisition process, the SAFE acts as a trusted introducer, and provides a symmetric session key to both the Examiner and the targeted servlet.

EnCase Servlet The Servlet runs on the machine targeted for acquisition, and provides evidence acquisition service to authenticated, authorized Examiners. It is comprised of two binaries, the user-mode program enstart.exe and the enstart_.sys driver. The servlet generally listens on TCP port 4445, although another port can be provided as a command line option. The servlet also includes a "stealth" functionality that putatively cloaks the running process from local detection, although it has been

[10] http://guidancesoftware.com/downloads/EEEauthentication.pdf
[11] http://guidancesoftware.com/products/ee_works.aspx

reported that the servlet is detectable by commercial rootkit detection software[12]

The Enterprise remote capture functionality uses a custom network protocol, which is described in two white papers available from Guidance Software: "Inside EnCase Enterprise: Review of Security Schema"[13] and "Digital Privacy Considerations with the Introduction of EnCase Enterprise"[14]. Guidance additionally provided us with a Microsoft PowerPoint presentation describing the protocol, entitled "EnCase Enterprise Edition Security Protocols"[15].

These documents seem to be the only officially available information describing the cryptographic scheme used by EEE to authenticate participants in a capture and protect the integrity and confidentiality of the resultant system image, and they omit many important details necessary for a proper security assessment of the protocol. The information excluded includes details on how EEE "bootstraps" trust relationships between these items, although the mention of Examiner and SAFE PGP keypairs signed by Guidance Software imply that trusted root public keys or fingerprints are included in the servlet binaries.

Guidance Software was granted U.S. Patents #6792545 and #7168000 for an "Enterprise computer investigation system" and "Automatic reconnect and reacquisition in a computer investigation system" respectively[16]. The descriptions and figures of the network acquisition process found in these patents are significantly more detailed than the other documents available on the protocol, and iSEC used the patent applications to help understand the low-level cryptographic workings of EEE. Although the patents do not specifically claim that their described methods match those of the shipping version of EnCase Enterprise Edition 6, the information in the patent applications is corroborated by the other documents specifically claiming to describe the EEE process.

The available documentation omits a clear technical enunciation of the goals of the Enterprise cryptographic scheme, although the following paragraph seems to serve this purpose best:

> "Guidance Software incorporates functionality within the Enterprise Edition of EnCase to ensure protection from electronic eavesdropping and prevent misuse of the product by unauthorized personnel."[17]

It should be noted that while authentication of the data captured is hinted at as a goal in other public documents, it is not specifically called out as a goal in either the "Review of Security Schema" or the description of the patented process. In both cases, authentication and authorization of the examiner's workstation is the primary goal.

7.2 Target Authentication Flaw in EnCase Enterprise

iSEC examined the publicly available data regarding the network acquisition process of EnCase Enterprise Edition, and observed the network traffic and local system operations of EEE during the acquisition process. Our analysis demonstrated that EnCase Enterprise Edition 6 did not cryptographically verify the identity of the acquisition target. This creates the opportunity for attackers to use simple network level hijacking techniques to respond to an acquisition request meant for another system and supply the attacker's

[12] Detected by AVG Anti-Rootkit according to an unverified report on the EnCase user message boards.
[13] http://guidancesoftware.com/downloads/Review_Security_Schema.pdf
[14] http://guidancesoftware.com/downloads/DigitalPrivacy.pdf
[15] This document was provided to us via email and does not seem to be available from Guidance's public website
[16] http://www.google.com/patents?as_q=&num=10&btnG=Google+Search&as_epq=&as_oq=&as_eq=&as_pnum=&as_vt=&as_pinvent=&as_pasgnee=guidance+software&as_pusc=&as_pintlc=&as_drrb_is=q&as_minm_is=1&as_miny_is=2007&as_maxm_is=1&as_maxy_is=2007&as_drrb_ap=q&as_minm_ap=1&as_miny_ap=2007&as_maxm_ap=1&as_maxy_ap=2007
[17] "Inside EnCase Enterprise: Review of Security Schema", Page 2

alternate disk image. This could be used in an attempt to "frame" an innocent investigation target if their system is acquired by EnCase Enterprise Edition.

Unlike the software failures discussed earlier, this issue only affects over-the-network acquisitions by EnCase Enterprise Edition, and does not impact the trustworthiness of drive images obtained through physical seizure and direct imaging of that drive through a hardware write blocker, nor network acquisitions performed over an Ethernet cross-over cable[18].

Our analysis rests on these findings obtained from analysis of the EnCase Enterprise Edition protocol and observations of EEE in use on production corporate networks:

1. The EEE servlet which is installed on an acquisition target may be customized for each EEE client, but iSEC found that the servlet is bit-wise identical for each system on which it is installed. Some EEE customers include this identical servlet in every desktop and server build, to insure that they can perform incident response or forensics on any of their systems without the step of installing new software.

2. The EEE servlet does not have a persistent cryptographic identity across reboots or installs. Various symmetric cipher keys are negotiated from time to time, but these keys are transient and the negotiation process is not signed by a persistent system-specific private key.

3. The EEE servlet does not utilize any pre-existing trust relationships that may exist in an Active Directory environment, such as the Kerberos identity of the machine account, to bootstrap trust between the examiner, the SAFE and the servlet.

4. During the network exchange that establishes a symmetric session key between the examiner and the servlet, the SAFE generates a random number and sends it, signed by the SAFE's private key, to the servlet. The servlet responds with a newly generated temporary session key and returns it, along with the random number it received, to the SAFE. This response is encrypted with the SAFE's public key, but is not signed with any private key belonging to the servlet, nor does it contain a token or shared secret specific to this instance of the servlet. As a result, the exchange that establishes a session key between the SAFE and the servlet serves only to verify the identity of the SAFE, but provides no verification of the servlet's identity.

5. iSEC observed that the EEE servlet queries the local operating system for information such as its domain name and IP address, which is verified locally against the IP address expected by the examiner's workstation. This means that any impersonation attacks will require the machine doing the impersonation to either be configured with the same IP address as the victim, or the attacker must modify the EnCase servlet (perhaps through a memory patch) to avoid this check or perform the check with the victim's IP address.

6. iSEC observed that acquisitions of different machines residing on the same IP address at different times does not trigger any obvious warnings to the examiner. This observation supports our belief that EEE does not contain a cryptographic mapping between IP addresses and servlet installs.

The end result of these findings is that neither the SAFE nor the examiner's workstation can verify the identity of an acquisition target utilizing cryptographic methods, and must rely on a lack of adversaries on the network or the existence of a perfectly secure network over which to communicate. These are two assumptions that cannot be safely made on a normal corporate Ethernet network. The use of EEE to

[18]These types of acquisitions can actually be performed by EnCase Forensic as well, and use a completely different and much simpler network protocol

acquire a system image over the network relies on the use of several network protocols, such as DNS[19] and ARP[20], which are known be be insecure and for which exist many freely available attack tools. In a normal network environment, it is incorrect to assume that the mapping of a client system's domain name such as "bob_workstation.company.com" to an actual, physical machine is accurate in the presence of minimally skilled network attackers on the network. This is especially true when reliance on this mapping underlies civil or criminal legal action, as may result from a EEE acquisition.

7.2.1 Attack Scenarios Against EEE Acquisition

The following are scenarios by which a malicious employee of an EnCase Enterprise Edition customer could cloak their activities from review or intercept EEE acquisition requests to "frame" or protect other users in the environment. iSEC did not carry out these attacks, but they are conceivable under our understanding of the cryptographic protocol and consistent with our observations of the workings of the product. These attacks rely on specific configuration details of the network on which EnCase Enterprise is used. These network-level attacks are well documented and tools are available that will work on many large corporate networks.

In our scenarios, Alice is GoatCorp's forensic examiner, Bob is the target of an internal investigation by Alice and is innocent of wrong-doing, and Malice is the one who is actually guilty of the crime and intends to get away with it.

1. **IP Address Takeover** - The simplest attack method Malice can use to frame Bob for her crime is to build a virtual or physical machine containing the incriminating evidence and to assign it to Bob's IP address. If Bob's computer is on, then it will inform him that there is an IP address conflict and lose connectivity to the network. In an environment where DNS resolution to user's desktops is simple, or where the examiner enters static IP addresses into the workstation for targeting, this may be a sufficient method to replace Bob's system image in the view of the examiner.

2. **ARP Spoofing** - A more complex attack would be for Malice to use an ARP Spoofing attack to intercept all network traffic between Bob, the SAFE, and Alice[21]. Malice would still need a machine resembling Bob's that contains the incriminating evidence, and would need to configure her ARP spoofing tool to forward the Ethernet frames containing EnCase traffic to her virtual machine while allowing all normal traffic to and from Bob. This is a slightly more complicated attack than a "standard" ARP Spoof, since the incriminating virtual machine needs to believe that it's IP address is the same as Bob's, but is practical with small modifications to the ARP spoofer and a proper firewall configuration on Malice's system. A similar attack could be carried out via DHCP spoofing, by assigning Bob a new IP address with Malice as his gateway. Malice would perform 1:1 network address translation on all of his traffic to grant him the illusion of direct access to the network, except for forwarding traffic meant for the servlet port to her incriminating virtual machine.

3. **DNS Attacks** - Malice could attack the corporate DNS system to reassign resolutions of bob.corp.goatcorp.com to the IP address of the incriminating machine. This attack could take many forms, such as attacking an insecure DNS update mechanism[22] or by attacking the DNS resolution on the SAFE or Alice's machines[23].

[19] An overview of DNS spoofing attacks is available here: http://www.securesphere.net/download/papers/dnsspoof.htm

[20] Popular attack tools include Ettercap (http://ettercap.sourceforge.net/), dsniff (http://monkey.org/~dugsong/dsniff/) and Cain & Abel (http://www.oxid.it/)

[21] This would more likely be an attack between Bob and the gateway in a large enterprise, since Alice and the SAFE would need to route their traffic at layer-3 to get to Bob

[22] This is seen in companies utilizing dynamic DNS updates dependent on DHCP leases

[23] It is unclear from the available documentation whether the resolution of a target's DNS name is performed on the SAFE,

4. **Modifying the Servlet or Operating System** - The integrity of the EnCase servlet or the underlying operating system can not be determined remotely by the EnCase examiner or SAFE. As a result, if Alice attempts to examine Malice's workstation for evidence of her attacks framing Bob, Malice can modify the servlet or kernel to provide a version of her system lacking the incriminating evidence. It is conceivable that currently available Windows rootkits could be modified to detect the EnCase servlet and provide it with incorrect data for the captured image, such as hiding the existence of a second disk in the machine and cloaking disallowed programs from the list of running processes.

7.2.2 Improvements to EnCase Enterprise and Recommendations for Users

The concept of remote acquisition of evidence is troubled by the general problem of trusting a computer to provide an accurate description of its own state. Such trust would have to be based upon a remote attestation of the integrity of the operating system and the EnCase servlet, which is unavailable without the existence of a hardware Trusted Computing Base (TCB). The method of physically seizing the computer allows for a much higher degree of confidence that the correct drive has been acquired as the investigator visually validates the correct workstation was seized. For now, we recommend that remote forensic acquisitions only be used for initial incident response and preliminary investigations. It is in the best interest of the EEE customer to also obtain drive images in the traditional physical manner, and to compare those images to the original network captures to verify that the correct system was captured. Additionally, EEE users need to be aware of the possibility that users with local administrator access to the target machine could modify the EnCase servlet in memory or tamper with the operating system to disguise their crimes. This should be especially taken into account when acquiring the systems of advanced computer users.

Although there are methods by which Guidance Software could make the EEE security schema less vulnerable to attack, the overall problem of secure communication with a remote system under an attacker's control and the verification of the integrity of the remote computer's software, including the EnCase servlet, is not solvable and is reminiscent of the struggles undertaken by the vendors of Digital Rights Management (DRM) software. We have considered two methods of servlet authentication which could reduce the risk of servlet impersonation attacks, but which would provide no protections against local administrators falsifying the information they provide to examiners. The methods we have considered are:

Utilizing Existing Trust Relationships - EnCase Enterprise Edition currently only supports the acquisition of Microsoft Windows based servers and desktops. Most large enterprises deploy Windows Active Directory services to provide authentication and management services across their Windows machines. In an Active Directory environment, each machine in the directory has a "machine account" with a Kerberos-based cryptographic identity. The SAFE-Servlet key negotiation could be modified to use Active Directory's identification facilities, which could provide mutual authentication between the target machine account and the SAFE machine account.

Servlet Registration - The EnCase servlet could be modified to perform a "registration" procedure upon its first use on the corporate network, which in many companies could occur on the first boot of a system after it is imaged. This registration procedure would include the generation of a unique keypair and the creation of a "hardware ID" based upon the specific configuration of the machine. Although it is impractical that a definite mapping between this keypair and the machine's domain name and IP address be kept in a large corporate network, the existence of such a keypair will enable EEE to keep an audit trail of servlets "calling home" that can be compared to the key fingerprints generated during an acquisition. This would make it easier for the examiner to detect fraud during an investigation.

examiner's workstation or both. Further research into this would be necessary before attempting this attack via subverting DNS resolution

The recent addition of hardware-secured Trusted Computing Bases (TCB) into select computer systems provides a possibility for a greater level of confidence in the future, but such software and hardware technology is not yet deployed in enterprise environments at the level necessary to provide any improvement to this process. Although we only examined the leading product in the "remote enterprise forensics" category, due to the technological and legal constraints placed upon all products that perform this type of remote acquisition we suspect that the competing products may suffer from similar issues.

8 Conclusion

We performed focused, shallow, and narrow testing of EnCase and The Sleuth Kit, yet immediately found security flaws with simple attack techniques.

We believe these vulnerabilities exist for several reasons. Forensic software vendors are not paranoid enough. Vendors must operate under the assumption that their software is the target of attacks. After all, the software is often used to examine evidence seized from suspected computer criminals and from computers suspected to have been compromised by an attacker — that is, the evidence has been under the control of someone capable and motivated to frustrate an investigation against them, or to attack again.

Vendors do not take advantage of the all of the protections for native code that platforms provide. For example, stack overflow protection, memory page protection (e.g. ensuring that the write bit is unset whenever the execute bit is set on a page) and safe exception handling are provided by Microsoft Operating Systems and compilers to mitigate problems like the ones we've identified. We did not observe the use of these protection mechanisms in the products we tested (See Figure 8 for example). The use of higher level programming languages such as C# or Java can eliminate many of the attacks we've identified altogether.

```
seg000:0041F900 sub_41F900      proc near               ; CODE XREF: seg000:004C006E↓p
seg000:0041F900                                         ; sub_4C0170+A1↓p ...
seg000:0041F900
seg000:0041F900 var_228         = dword ptr -228h
seg000:0041F900 var_224         = dword ptr -224h
seg000:0041F900 var_220         = dword ptr -220h
seg000:0041F900 var_21C         = dword ptr -21Ch
seg000:0041F900 var_218         = dword ptr -218h
seg000:0041F900 var_214         = dword ptr -214h
seg000:0041F900 var_210         = dword ptr -210h
seg000:0041F900 arg_0           = dword ptr  4
seg000:0041F900 arg_4           = dword ptr  8
seg000:0041F900 arg_8           = dword ptr  0Ch
seg000:0041F900 arg_C           = dword ptr  10h
seg000:0041F900
seg000:0041F900                 sub     esp, 228h       Prologue
seg000:0041F906                 push    ebx
seg000:0041F907                 push    esi
seg000:0041F908                 push    edi
seg000:0041F909                 lea     eax, [esp+234h+var_210]
seg000:0041F90D                 mov     [esp+234h+var_220], eax
     ...                        ...
seg000:0041FA1B                 call    sub_41A3A0
seg000:0041FA20                 pop     edi
seg000:0041FA21                 pop     esi
seg000:0041FA22                 mov     al, bl
seg000:0041FA24                 pop     ebx             Epilogue
seg000:0041FA25                 add     esp, 228h
seg000:0041FA2B                 retn    10h
seg000:0041FA2B sub_41F900      endp
```

Figure 8: The prologue and epilogue of a function in the EnCase binary does not include any stack protection checks (/GS). We did not observe any functions with stack protection checks in them.

Forensic software customers use insufficient acceptance criteria when evaluating software packages. Criteria typically address only functional correctness during evidence acquisition (*not* analysis) when no attacker is present,[24] yet forensic investigations are adversarial. Therefore, customers should pressure vendors to observe the practices in (2) and to perform negative testing against the product (discussed further below).

The software and methods for testing the quality of forensic software should be public. Carrier notes[25] that sufficient public testing tools, results, and methodologies either don't exist or are not public. Making these public will help customers know what they are getting and where they may be vulnerable, and may even raise the standard of testing and improve the quality of the software.

Acquisition of system images over a corporate network is an inherently dangerous procedure and should be avoided whenever possible. In the case of EnCase Enterprise Edition, network acquisition allows attackers on the network to subvert analysis of their systems or falsely provide drive images for other users on the network.

8.1 Future Work

We have only scratched the broad attack surface of the products we investigated. We performed testing on a wide range of popular file formats and several popular filesystem and partition formats but there are

[24] see http://www.cftt.nist.gov/ and [1], specifically lines 43 – 46 ("The two critical measurable attributes of the digital source acquisition process are accuracy and completeness. Accuracy is a qualitative measure to determine if each bit of the acquisition is equal to the corresponding bit of the source. Completeness is a quantitative measure to determine if each accessible bit of the source is acquired.") and 86 – 172. Although the NIST document focuses strictly on the acquisition of evidence, it is not enough to standardize only acquisition. Most forensic toolkits also include functionality for evidence analysis, and it is at the analysis stage where security, not just accuracy and completeness, is crucial.

[25] http://dftt.sourceforge.net/: "To fill the gap between extensive tests from NIST and no public tests, I have been developing small test cases."

many formats that we did not test. Most fuzz testing was simplistic and did not take advantage of the specific formats of the files, filesystem or partition types that were being fuzzed. We tested two popular forensic suites, but there are several others in wide use which we did not test. We did not perform an in-depth audit of the source code or analysis of the binaries of either product. There's a wide range of options for future investigation into forensic software.

8.2 Vendor Responses

Brian Carrier, the lead developer of TSK, corrected all of the flaws discussed here in version 2.09.

Guidance Software responded to a pre-release draft of this report, which did not contain our analysis of EnCase Enterprise, via a posting[26] to the BUGTRAQ mailing list. In this response they state:

> All of the testing involved intentionally corrupted target data that highlighted a few relatively minor bugs. The issues raised do not identify errors affecting the integrity of the evidence collection or authentication process, or the EnCase Enterprise process (i.e., the operation of the servlet code or the operation of the SAFE server). Moreover, the issues raised have nothing to do with the security of the product. Therefore, we strongly dispute any media reports or commentary that imply that there are any "vulnerabilities" or "denials of service" exposed by this report.

Guidance has informed iSEC that all of the EnCase Forensic issues described in section 6 of this report will be addressed in the next maintenance release of the product, version 6.7. In private correspondence, Guidance has denied that the analysis contained herein of EnCase Enterprise Edition constitutes a vulnerability or weakness in the EEE acquisition process, and promised a public response to our findings. When such a response is released, we will cite it in this document.

8.3 Acknowledgments

We thank Brian Carrier, lead developer of The Sleuth Kit, for his fast and helpful responses to our defect reports. Thanks also go to iSEC's summer interns, Justine Osborne and Jigar Shah, for their help analyzing and documenting software crashes.

References

[1] http://www.cftt.nist.gov/DA-ATP-pc-01.pdf. 28

[2] http://dftt.sourceforge.net/.

[3] http://www.seccuris.com/documents/papers/Seccuris-Antiforensics.pdf.

[4] http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-foster-liu-update.pdf.

[5] http://metasploit.com/projects/antiforensics/.

[26] http://www.securityfocus.com/archive/1/474727

[6] http://www.simson.net/clips/academic/2007.ICIW.AntiForensics.pdf. 2

[7] B. Carrier. File system forensic analysis. Addison Wesley, 2005.