Scalpel: A Frugal, High Performance File Carver

Golden G. Richard III
Vassil Roussev
Department of Computer Science
University of New Orleans
New Orleans, LA 70148
Contact: golden@cs.uno.edu

Abstract

File carving is an important technique for digital forensics investigation and for simple data recovery. By using a database of headers and footers (essentially, strings of bytes at predictable offsets) for specific file types, file carvers can retrieve files from raw disk images, regardless of the type of filesystem on the disk image. Perhaps more importantly, file carving is possible even if the filesystem metadata has been destroyed. This paper presents some requirements for high performance file carving, derived during design and implementation of Scalpel, a new open source file carving application. Scalpel runs on machines with only modest resources and performs carving operations very rapidly, outperforming most, perhaps all, of the current generation of carving tools. The results of a number of experiments are presented to support this assertion.

Keywords: digital forensics, file carving, high performance computing, data recovery

1 Introduction

A wide variety of digital forensics tools, both commercial and open source, are currently available to digital forensics investigators. The purpose of these tools is to provide layers of abstraction that allow investigators to safely make copies of digital evidence and perform routine investigations, without becoming overwhelmed by low level details, such as physical disk organization. One of the most important tools in a digital forensics investigator's toolbox is a file carver.

File carvers read databases of headers and footers, which are strings of bytes at predictable offsets, and search one or more target disk images for occurrences of the headers and footers. The goal is to identify the starting and ending locations of files in the disk images and "carve" (copy) sequences of bytes into regular files. Additional file-type specific rules may apply, for example, to associate the footer closest to a discovered header or farthest away from the header or inclusion or exclusion of the footer in the carved file.

File carving is a particularly powerful technique because files can be retrieved from raw disk images, regardless of the type of filesystem. File retrieval is possible even if the filesystem metadata has been completely destroyed. For example, a file deposited on a FAT partition can often be recovered even if the partition is reformatted as NTFS, then ext2, then FAT again, even if bad block checks (which are generally read-only operations) are applied. While a filesystem's metadata can be quite fragile, file data is much more resilient.

One limitation of the current generation of automatic file carvers is that a file's data must be contiguous to be carved properly. With some manual intervention or additional work [5], even non-contiguous data can be carved, but luckily, modern filesystems, such as ext2/3 (for Linux) and NTFS (for Windows), are actually quite kind to file carvers. This is because they strive to perform disk allocation which minimizes file fragmentation, in order to reduce seek time and improve filesystem performance. Even under legacy filesystems such as FATx, which are prone to fragmentation, the data of many files of modest size is likely to be unfragmented. This is because file fragmentation, if present, is on cluster boundaries and cluster sizes under FATx tend to be rather large.

This paper discusses some issues for optimizing file carving operations and describes

Scalpel, a new open source file carving application. The primary benefits of Scalpel over the current generation of file carvers is that it operates rapidly, even on legacy hardware with limited memory.

The motivation for the development of Scalpel is several-fold. First, current generation file carvers tend to perform poorly unless substantial resources are available on the machine performing carving operations. This is simply unnecessary, as the paper will demonstrate. The second reason is that, as forensic targets grow in size and the backlogs of law enforcement (and private) digital forensics laboratories grow, *we simply must expect our tools to squeeze every bit of performance out of the computing equipment we have available.* Many of the strategies employed in development of high-performance operating systems can be applied to digital forensics tools, including minimizing memory-to-memory copying and elimination of unnecessary disk I/O.

2 File Carving Strategies

In the absence of metadata providing a complete description of the contents of a set of disk images, file carving requires one or more complete passes over each image. Since a file header may appear anywhere in a disk image, a limiting factor on performance is the total number of bytes to read a disk image once, T_{read}. T_{read} is constant for all carving strategies and cannot be substantially reduced through clever optimization. In addition, other factors affect the minimum cost: the time to search for headers, footers, and relevant strings, memory-to-memory copy operations, and the total number of bytes written for carved files, T_{write}. To maximize the performance of a file carving application, a few guiding principles are appropriate. Most are common programming "sense", but are very important in file carving because of the huge amounts of data these applications must move. In a review of existing open source file carvers and in the design of Scalpel, the following principles were compiled:

- *The amount of time searching for headers and footers (once the data is read from the disk image) should be minimized.* This means employing a fast string search algorithm and minimizing unnecessary searches (for example, not searching for footers for which no viable header has been found).
- *Memory-to-memory copies should be minimized.* File carvers copy billions (and soon, trillions) of bytes of data. Even though disk operations are orders of magnitude slower than memory copies, excessive memory-to-memory copies do reduce performance. If possible, data should be written to carved files directly from the buffer used for reads against the disk image. These are lessons learned from design of high-performance operating systems.
- *T_{write} should minimized by carving the smallest number of files that meet the needs of the investigator.* Since write operations on mechanical disks are particularly expensive, only those blocks of data in the disk image which match the search specifications provided to the file carver should be written. This precludes an approach where, for example, carving operations begin whenever a footer is discovered, but the carved "file" is discarded when no suitable footer is found.

Assuming that these principles are applied, then the performance of a file carver will depend heavily on the number of passes (including partial ones) over each disk image. Said another way, the number of bytes read to accomplish the carving must be minimized—the multiplicative factor for T_{read} must be as small as possible. A secondary consideration is that, all other things being equal, sequential passes over the disk image will likely be faster than many randomly ordered reads, for two reasons. First, modern filesystems perform block read-ahead and these read-ahead facilities are interrupted by non-sequential reads. Second, total seek time is reduced.

2.1 A New Carver: Scalpel

In this section, the design and implementation of Scalpel, a new, open source file carver, are presented.

2.1.1 Design Considerations

Scalpel is a high performance file carver with three primary design requirements. The requirements, in decreasing order of importance, are:

- *Frugality.* The file carver should run on machines with minimal resources. The target platform is a Pentium 2 class machine w/ 512MB (or less) of RAM, running a bootable Linux distribution, such as Knoppix [4], Knoppix STD [9], or Helix [10]. On such a machine, Scalpel should be able to carve files of any type and of any maximum size, subject to available disk space.

- *High performance*. Scalpel should perform file carving as quickly as possible, without compromising the accuracy of carving operations.
- *Support for distributed implementation*. The basic techniques in the file carver should be readily adaptable to a distributed, cluster-based digital forensics platform, such as that described in [3]. Such platforms may support caching most or all of the disk image in RAM, which should speed carving operations substantially.

Scalpel's existence, then, is motivated primarily by a desire to quickly carve files of arbitrary size on machines with modest resources. Since file carving is a non-interactive, I/O bound task, there is little reason to dedicate expensive computing resources, which might be used for other purposes, to file carving. A Pentium class machine, equipped with a modest amount of RAM and a modern disk interface should perform adequately. Section 3 shows that Scalpel meets both the first and second goals. The third goal has been met, but discussion is beyond the scope of this paper. The authors are currently incorporating Scalpel into their distributed digital forensics framework and these results will be published in a future paper.

2.1.2 Scalpel Internals

Scalpel reads a configuration file on startup that defines the types of files that should be carved and for each file type, additional information, including the specifications of the header and footer strings and the maximum file size for the type.

To perform file carving, Scalpel makes only two sequential passes over each disk image. The first pass reads the entire disk image in large chunks (of user-definable size, with a default size of 10MB). Each chunk is searched for file headers and a database of the locations of these headers is maintained. Once the current chunk's header indexing is complete, a search is performed for footers. For a particular file type for which a footer is defined, a footer search is conducted against the current chunk only if the footer might potentially match some header in the file. This is possible under two conditions. The first is if a potentially matching header was discovered in the current chunk of the disk image. The second is if a potentially matching header was discovered in a previous chunk of the image file, but close enough to the current position to meet the requirements of the maximum carve size for the file type. The location of each matching footer is stored.

Once the first pass over the disk image is complete, Scalpel has a complete index of header and footer locations, which is used to populate a set of *work queues* that control file carving operations during the second pass over the disk image. For each file header in the index, an attempt is made to match the header with an appropriate footer, subject to rules in the configuration file. One work queue is associated with each chunk of a disk image and for a single file to be carved, each queue contains at most one of the following record types:

- *STARTCARVE*: A file carving operation begins in this chunk. The starting location in the chunk corresponds to the location of the header for the file in the disk image. The file to be carved is opened and the initial portion of the file is written.
- *STARTSTOPCARVE:* A file carving operation begins and ends in this chunk. The file is opened, some portion of the current chunk is written, and the file is closed.
- *CONTINUECARVE:* A file carving operation spans this chunk. The entire contents of the chunk is written to the file being carved and the file remains open.
- *STOPCARVE:* A file carving operation ends in this chunk. Some portion of the chunk is written to the file being carved and the file is closed.

During the second pass over a disk image, Scalpel again processes the disk image in chunks (of the same size used in the first pass). As described above, each chunk has an associated work queue, which describes carving operations

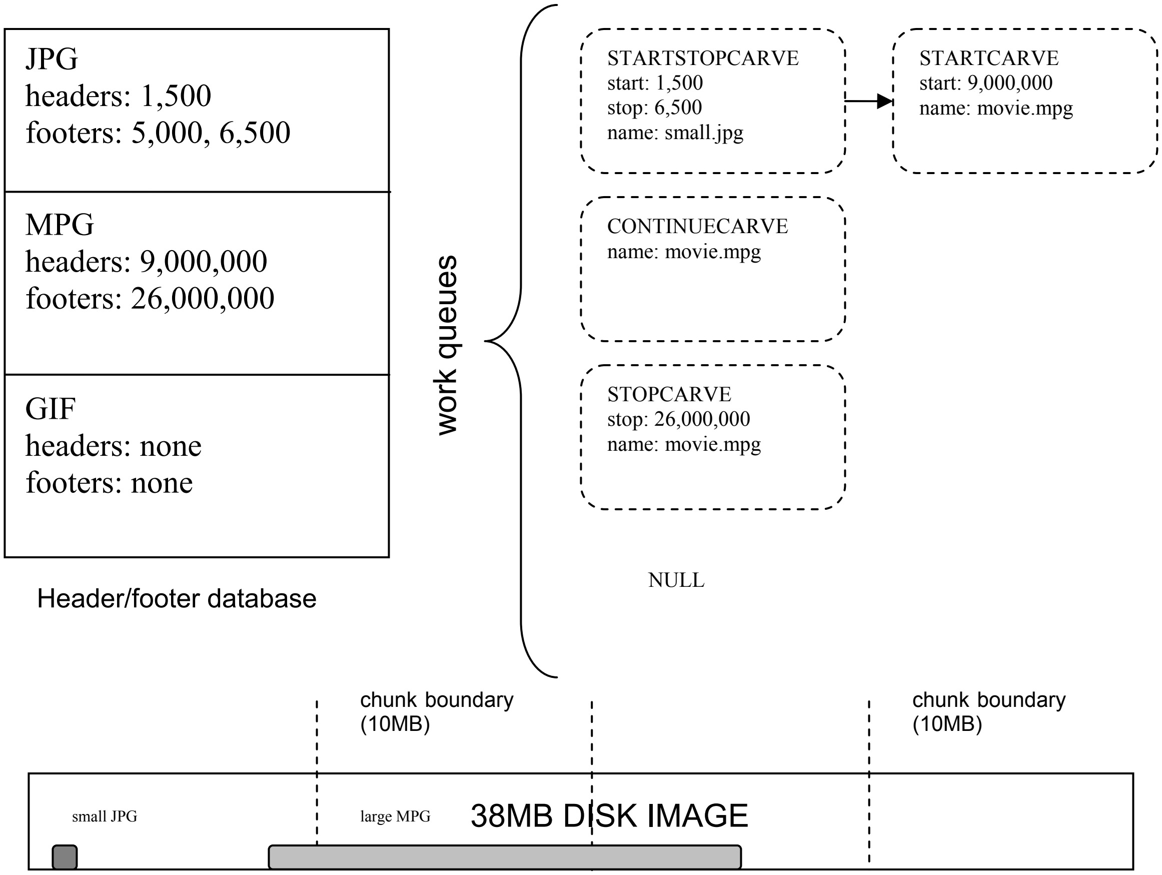

Figure 1. Work queues in Scalpel. Each 10MB chunk of an image file is assigned a work queue, which contains a sequence of records that define carving operations for that chunk during Scalpel's second sequential pass over the image file.

to carry out when that chunk is read. The use of work queues is illustrated with a brief example in Figure 1. In this example, a 38MB disk image is examined, which contains two files, a small JPG and a larger MPG file. The first pass has already built the header and footer databases. For the JPG, the header at byte 1,500 is paired with the footer at byte 6,500. For the MPG, the header at position 9,000,000 is paired with the footer at position 26,000,000. A single STARTSTOPCARVE entry is deposited in the queue for chunk 0 of the disk image corresponding to the complete carving operation for the JPG file. A STARTCARVE entry is placed in the chunk 0 work queue for the MPG file, which will cause carving to begin at byte 9,000,000 during processing of chunk 0. The CONTINUECARVE entry deposited in chunk 1's work queue will result in writing the entire chunk to the MPG file. Finally, when chunk 2 is processed, the STOPCARVE entry will copy the first 6MB of chunk 2 into the MPG file and close the file. The last chunk of the disk image is completely skipped on the second pass, because its work queue is empty.

The motivation for the use of work queues is to make maximal use of the data in each chunk of the disk image when it is read. No extraneous memory-to-memory copies are performed—writes to carved files are performed out of the same buffer used to hold the current chunk of the disk image. To further reduce disk activity during the second pass over the disk image, Scalpel uses seek operations to skip consecutive chunks for which no work is scheduled.

This often results in dramatic speedup over simply reading each chunk in the second pass.

2.1.3 Analysis of Scalpel

The lower bound on the number of bytes read by Scalpel is T_{read}, which corresponds to one pass over the disk image in which no headers are discovered. In this case, the second pass is eliminated entirely, because there is nothing to carve. In the worst case, Scalpel's read performance is 2 * T_{read}, which corresponds to a second pass in which each 10MB chunk is involved in at least one file carving operation. Scalpel minimizes T_{write} by never writing carved files unless the associated data meets all of the requirements imposed by the configuration file for a particular file type.

Considering memory usage, the first design goal for Scalpel dictates that it execute correctly on machines with limited resources. Scalpel's largest dynamic memory allocation is the buffer used to hold "chunks" of an image file as it is carved, by default, 10MB. The work queues used by Scalpel consume only a small amount of memory, even for large carve sets. As such, it operates well on machines with modest amounts of RAM and no swap space.

2.2 Foremost 0.69

One of the widely used open source file carvers is Foremost [6]. Foremost has been in use for several years and was used as a basis for the development of Scalpel. They still share some basic functions, such as the configuration file parser.

Foremost performs all carving operations during a "single" pass over a disk image, reading the disk image in chunks (default size in 0.69 is 10MB). The carving strategy is illustrated in Figure 2. When Foremost discovers a header in the current chunk, it determines if the current chunk contains enough data beyond the header to accommodate the maximum carve size for <u>any</u> file type. If not, an additional disk read is performed to build an in-memory buffer containing the header and enough data to accommodate the maximum carve size for <u>any</u> file type. If the file type corresponding to the header has a defined footer, a search is undertaken to discover a matching footer in the buffer. If a footer is found, then the portion of the buffer between the header and footer is written to a file and the carving operation for the file is complete. For file types with no defined footer, the entire buffer is written. Once this single file carving operation is complete, the buffer is discarded and processing continues sequentially from just past the position of the discovered header.

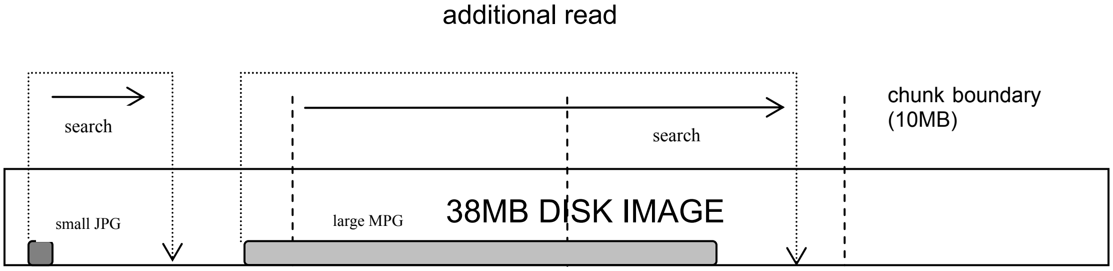

Figure 2. Simplified view of Foremost 0.69 file carving strategy. A disk image is processed in chunks, with the default size being 10MB. When a header is discovered, the maximum carve size is used to calculate how far away from the header the footer (or the maximum file size) might be. There are two possibilities. If the end of the file is completely inside the current chunk (refer to the small file, above), then no additional reads for this file carving operation are needed. If the end of the file might be outside the current chunk (as in the large file, above), Foremost performs an additional read to build a buffer beginning with the header and whose length is the maximum carve size for any file type.

2.2.1 Analysis of Foremost

Foremost reads k * T_{read} bytes to process a disk image, where $k \geq 1$. Beyond a single pass over the disk image, which costs T_{read}, Foremost performs additional reads, with the quantity depending on the number of headers discovered and the frequency of complete files to be carved appearing entirely in the current chunk of the disk image. An analytical derivation of k is difficult to obtain, but some experimental results, obtained by instrumenting a copy of Foremost 0.69, shed some light. In the experiments described in the next section, the factor k was between 1 (for carving an empty disk image) and 45. This means that the amount of data read by Foremost is in some cases equivalent to tens of passes over the disk image, in the worst case observed in the experiments, equivalent to 45 * T_{read} bytes. The reason that the performance differential between Foremost 0.69 and Scalpel is generally only a small constant factor and not on the order of $k / 2$ (since Scalpel performs exactly full two passes, in the worst case) is because many of the reads performed by Foremost are overlapping. These reads are generally served by the filesystem cache, reducing but not eliminating their cost. With large maximum carve sizes, however, Foremost's performance suffers. Because many of the reads are non-sequential, they may also disturb operating system-level read-ahead mechanisms.

Foremost has a substantial memory requirement. In particular, a *malloc()* call at the point of each header discovery allocates a block of dynamic memory equal in size to the maximum carve size for any file type defined by the configuration file. This means that Foremost 0.69 is incapable of carving a file whose size is greater than the available virtual memory at the point of header discovery. While this isn't as problematic on machines with large amounts of RAM, it will be troublesome if very large files are carved. There is also a large memory-to-memory copy at the point of header discovery. Unfortunately, the large memory requirements of Foremost preclude using it to carve even moderate-sized files on modest machines running without swap space (e.g., an older box booted using Knoppix).

A new version of Foremost, 1.0beta, which incorporates some new features, was released as this paper was completed. As this beta version of Foremost appears to be a complete rewrite and in some limited testing does not yet perform all file carving operations correctly, no detailed performance study for 1.0beta was conducted. A comparison between a stable version of Foremost 1.0 and Scalpel is the subject of future work.

3 Experimental Results

3.1 Linux

This section presents the results of a number of experiments designed to validate the correct operation of Scalpel and to test its performance. Foremost 0.69, described in the last section, is used as a basis for comparison, for two reasons. The first is that it is widely used and the author has personal experience with its general accuracy in performing file carving. The second is that Foremost 0.69 is open source. This is a crucial point, because it allowed a thorough investigation of the file carving strategy, which provided some of the insights that fueled development of Scalpel. Carrier [7] and Kenneally [8] each make convincing arguments for the open source case in the digital forensics arena. It's a complicated issue and troublesome economic factors enter the debate for software developers who don't have other means of buying food, but in this case rapid development of a new tool (Scalpel) was made possible because the developers of Foremost open sourced their own carving tool.

The experiments described below include a number of different types of carving operations, varying both the types of files carved and the maximum sizes for carving operations. There are some minor differences in the default behaviors of Scalpel and Foremost that bear mention. The first is that by default, Scalpel does not carve files whose footers are not found within the defined maximum carve size for a file type. Foremost always performs carving, truncating the carve at the maximum file size, even if a matching footer isn't discovered. The "-b" command line option for Scalpel is for Foremost compatibility and results in carves always being performed, with a note in the Scalpel log file that the file is truncated. Another issue is that Foremost 0.69 doesn't detect all headers [footers] when the headers [footers] for a particular file type overlap in the disk image. A Scalpel command line option, “-r”, causes Scalpel to use only the first occurrence of overlapping headers [footers], mimicking Foremost’s behavior. In the tests below, these command line options were used to force Scalpel to emulate Foremost, so that a more accurate performance comparison might be made.

The current version of Scalpel reads exactly the same configuration file format as Foremost 0.69 and so in all cases, exactly the same configuration files were used for the runs of Foremost 0.69 and Scalpel 1.5. MD5 hashes of all files carved were compared. In all cases, the sets of files carved were exactly the same for Scalpel and Foremost 0.69.

Experiments under Linux were conducted on two machines, whose relevant specifications are given below. In the tables, these machines are referred to as “P2-350” and “T40p”:

P2-350:

350MHz Pentium 2 with 512MB of RAM and no swap space.
4 port ATA-133 IDE controller, 7200rpm 80GB drive for holding carve results.
Operating System: Knoppix 3.7.

T40p:

Thinkpad T40p, 1.7GHz Pentium M, with 2GBof RAM and 4GB of swap space.
7200rpm 60GB drive.
Operating System: RedHat 9 with upgraded 2.40.20 kernel.

The T40p was the development machine for Scalpel and the tests on this machine are rather simple. These tests are representative of those used to guide the design of Scalpel’s carving strategy. Most of the substantial carving experiments were performed on the P2-350, which is a much less powerful machine, since this sort of machine provides greater bang for the carving buck.

The following tables illustrate the difference in performance between Scalpel and Foremost under a number of different carving scenarios. Each carving operation was performed multiple times and the median time chosen as representative. In all cases, the machines were idle, except for the carving tests, and no network connections were active.

Scalpel 1.5 (20MB max)	13s
Foremost 0.69 (1MB max)	12s
Foremost 0.69 (5MB max)	42s
Foremost 0.69 (10MB max)	57s
Foremost 0.69 (20MB max)	1m43s

Table 1. Carving results for 512MB USB key image on T40p. Carving parameters: 1MB / 5MB / 10MB / 20MB JPG and DOC. ~1,100 files carved.

Scalpel 1.5	24s
Foremost 0.69	2m0s

Table 2. Carving results for 512MB USB key image on T40p. Carving parameters: 20MB JPG, 20MB DOC, 100K BMP, 4MB AVI, 1MB ZIP. ~1720 files carved.

Scalpel 1.5	34s
Foremost 0.69	34s

Table 3. Carving results for 1.1 GB NULL image (zeroed drive image) on Thinkpad T40p. Carving parameters: 20MB max JPG + Microsoft Office. 0 files total carved.

Scalpel 1.5 (10MB max)	11m27s
Foremost 0.69 (1.2MB max)	8m59s
Foremost 0.69 (5MB max)	12m19s
Foremost 0.69 (10MB max)	12m47s

Table 4. Carving results for 1.2 GB FAT32 (from e-bay) on P2-350. Carving parameters: 1.2/5/10MB JPG. ~2,200 files carved.

Scalpel 1.5	18m36s
Foremost 0.69	23m18s

Table 5. Carving results for 1.2 GB FAT32 (from e-bay) on P2-350. Carving parameters: 10MB GIF, 10MB JPG, 10MB AVI, 10MB MPG, 10MB DOC, 50K HTML, ~5,000 files carved.

Scalpel 1.5	1h33m10s
Foremost 0.69	6h21m54s

Table 6. Carving results for 8GB raw drive (unknown source, no partition table) on P2-350. Carving parameters: 10MB GIF, 10MB JPG, 10MB AVI, 10MB MOV, 10MB MPG, 100K BMP, 5MB DOC, 50MB PST/OST, 50K HTML, 5MB PDF, 200K WAV, 1MB RealAudio, 10MB ZIP. ~52,000 files carved.

Scalpel 1.5	2h40m39s
Foremost 0.69	9h50m31s

Table 7. Carving results for 40GB NTFS (from a UNO laboratory) on P2-350. Carving parameters: 10MB JPG, 50MB AVI, 10MB DOC, 50K HTML, 5MB PDF. ~ 72,000 files carved.

Scalpel 1.5	43m20s
Foremost 0.69	---------

Table 8. Carving results for 80GB drive on P2-350. Carving parameters: 1GB max Outlook.1 files total carved.

Table 1 shows the results of carving a 512MB USB key image on the T40p. In this test, JPG + DOC files are carved, with maximum sizes of 1MB, 5MB, 10MB, and 20MB. Scalpel is insensitive to small differences in user-specified maximum carve sizes, so only the results for the 20MB maximum size are shown. For the 1MB size, Foremost is marginally faster. The speed of Foremost degenerates as the maximum carve size is increased. This is because a larger maximum size increases the likelihood that a footer search must extend beyond the current chunk of the disk image, which causes Foremost to issue additional read operations.

Table 2 includes results for the same USB key image, but carving a number of file types, whose maximum sizes are given in the caption. Scalpel is roughly 5X faster than Foremost 0.69 in this test.

Table 3 is a simple throughput test on a zeroed disk image. This mirrors the case where a drive image contains nothing of interest—for example, it might be new or it might have been forensically sanitized. As expected, Scalpel and Foremost 0.69 have identical performance, since their strategies diverge only when a matching header is found in a disk image.

Table 4 provides results for JPG-only carves on a 1.2GB drive obtained from e-bay. In this experiment, the P2-350 was used. Scalpel is outperformed by Foremost 0.69 only with a maximum carve size of 1.2MB, where Foremost issues only 263MB of additional read operations beyond the 1.2GB required to read the entire disk image once. For the 5MB and 10MB maximum carve sizes, Foremost issues 4.9GB and 21GB of additional reads, respectively.

Table 5 shows results for carving a number of file types from the same 1.2GB drive. The types carved are listed in the caption. Foremost issues 48GB of reads beyond the 1.2GB required to read the disk image a single time.

The results in Table 6 are for carving an 8GB drive of unknown origin using the P2-350. The drive's partition table is wiped out, but the drive operates correctly. A number of file types of various maximum sizes are carved. Scalpel performs the carve operations significantly faster—saving almost 5 hours of processing time. Again, this is possible because Scalpel strives to minimize disk I/O, while Foremost 0.69 performs 238,270,750,000 bytes of reads in addition to its single pass over the 8GB image. As the number of types and maximum sizes for carved files increases, the performance of Foremost falls farther behind that of Scalpel.

In Table 7, results are presented for carving a 40GB drive. In this example, a large 50MB cap on AVI-format video files is allowed. Scalpel performs the carving operation in approximately ¼ the time required by Foremost, saving over 7 hours of processing time. In this experiment, Foremost performs 117,622,357,936 bytes of additional reads in addition to a single pass over the 40GB image.

The final test under Linux, whose results appear in Figure 8, is particularly important. In this test, an 80GB drive image is searched for large (maximum: 1GB) Outlook email archives. On the 512MB RAM P2-350, Foremost 0.69 is unable to complete the operation—it crashes with a segmentation fault when an Outlook file header is discovered, because 1GB of dynamic memory cannot be allocated when the single matching header is discovered. Scalpel carves a single Outlook email box from the 80GB drive image in 43 minutes, 20 seconds. The logical size of the Outlook email box was roughly 600MB, though a full 1GB was carved because no footer was specified for Outlook in the configuration file.

These results indicate that, except in cases where only a small number of modest-sized files are carved, Scalpel performs much faster than Foremost 0.69 and will run on a significantly less powerful machine.

3.2 Windows XP

In order to compare Scalpel to other file carvers, a very limited test was performed under Windows XP. This experiment duplicates the carving operations performed in Table 6 on a 3 GHz Pentium 4 with 2GB RAM and dual 72GB 15K rpm SCSI drives. A Win32 compilation of Scalpel, using the MinGW version of gcc 3.4.2, FTK v1.50b, FTK v1.60, and WinHex 12.1 were used. The results appear in Table 9.

Scalpel 1.5	1h10m15s
WinHex 12.1	1h12m0s
FTK 1.50b	1h36m0s
FTK 1.60	2h10m0s

Table 9. Carving results for 8GB raw drive (unknown source, no partition table) on P4-3GHz. Carving parameters: 10MB GIF, 10MB JPG, 10MB AVI, 10MB MOV, 10MB MPG, 100K BMP, 5MB DOC, 50MB PST/OST, 50K HTML, 5MB PDF, 200K WAV, 1MB RealAudio, 10MB ZIP. ~52,000 files carved.

WinHex's file carving facility is most comparable to Scalpel. In this limited test, Scalpel is only marginally faster than WinHex 12.1. The number of files carved was comparable enough to measure carving performance, but some anomalies were noted. First, for some file types, WinHex modifies user-specified limits on carved file sizes, as described in the WinHex documentation. This makes a direct comparison (e.g., by using MD5 hashes) between the files carved by Scalpel and WinHex difficult, because WinHex's carving strategy is proprietary. However, for most file types, the number of files carved was very close and a visual check indicated that, for image files at least, the same file sets were carved. The performance of WinHex on modern hardware is excellent and the results agree for the most part with Scalpel.

FTK's carver is very limited, supporting only a small number of predefined file types, whose headers/footers are discovered during the "Add Evidence" preprocessing phase. The timing provided in Table 9 for FTK covers only BMP, GIF, JPEG, PDF, HTML, and OLE (Office documents) and includes FTK's "Add Evidence" preprocessing phase (with the minimum number of options permissible for data carving to be supported), data carving, and file export.

Since FTK provides no control over maximum carve sizes, headers, or footers and performs some extra work in the preprocessing phase that isn't performed by the other file carvers in the test, it is impossible to directly compare Scalpel's performance with FTK. Even so, a general comparison is possible. The performance of FTK's file carving facility is poor compared to Scalpel and WinHex, especially in light of the number of file types not being carved.

An alarming situation was noted in carving the 8GB image under FTK 1.50b. FTK carved 3,463 GIF files. WinHex 12.1, Scalpel 1.5 (under both Windows and Linux) and Foremost 0.69 (under Linux) each carved exactly 4,817 GIF files. In some cases, the lengths of the GIFs carved by WinHex were different than those carved by Scalpel and Foremost, but in a thumbnail viewer, it was easy to verify that the sets of files carved were largely the same. Furthermore, the great majority (but not all) of the MD5 hashes between the Scalpel and WinHex carve sets matched. Approximately 400 of the GIFs carved by Scalpel/WinHex/Foremost could not be viewed and were likely not actually GIF format (e.g., they were binary garbage with appropriate "GIF" headers and footers). But of the 3,463 GIF files carved by FTK, 2,442 of the files were exactly 783 bytes in length. None of the files of 783 bytes in length were viewable—all were corrupt. Furthermore, neither Scalpel, Foremost, nor WinHex carved a single GIF file of 783 bytes in length.

This problem appears to be fixed to some degree in FTK v1.60, which carved 4,194 GIF files, approximately 100 of which were corrupt. FTK may be employing a filter to avoid carving corrupted GIFs. It is possible that this filter is overzealous, but a substantial amount of additional testing is required to draw any strong conclusions. What is clear is that an upgrade to v1.60 is critical for anyone using FTK for file carving and the output from FTK should be compared to another file carver.

3.3 Scalpel Performance Summary

Scalpel carves exactly the same set of files, for a given configuration file, under both Linux and Windows XP. While Scalpel has no optimizations for the Windows platform, note that the performance difference between the P2-350 (essentially, a throw-away box, worth less than $200) running Linux and the high-end P4 machine (> $3,000, with 15K SCSI disks) running XP is not substantial.

4 Conclusions and Future Work

File carving is an important digital forensics technique and has applicability beyond digital investigation, e.g., in simple data recovery. Many file carving applications are currently available, but to the author's knowledge, none (except Scalpel, presented in this paper) are able to quickly perform file carving operations of arbitrary size on machines with modest resources.

Scalpel is a new open source file carver, sharing some code with Foremost 0.69, but employing a significantly different, highly optimized carving strategy, which avoids unnecessary memory-to-memory copies and disk I/O. Scalpel performs at most 2 sequential passes over an image file to complete a set of user-defined carving operations and exhibits excellent performance on Pentium class hardware with modest amounts of RAM and no swap. Detailed experiments were conducted to illustrate the performance of Scalpel.

Several enhancements to Scalpel are currently underway. One is more accurate header analysis, in the style of the Unix *file* command. This requires some care, because over-zealous scrutiny of file formats may result in carving operations not being performed when the "incorrect" file data (perhaps due to an overwritten block, for example) may be only slightly damaged and in fact, usable. We are also incorporating Scalpel's file carving techniques into our framework for distributed digital forensics, to speed carving operations on very large images. It may be possible to speed up the Windows version of Scalpel further—this is currently an open issue. Finally, we want to compile and test Scalpel on other Unix platforms, including Mac OS X.

In the future, we hope that collaboration with researchers working on next-generation digital forensics techniques, such as automated reassembly of fragmented documents (e.g., [5]), which are beyond the capabilities of the current generation of "generic" file carvers, will yield even more powerful file carving applications.

The latest version of Scalpel, including source code and a Windows executable, is available at:

http://www.digitalforensicssolutions.com/Scalpel

Comments, feedback, bug reports, and suggestions for additional features are welcome and should be addressed to golden@cs.uno.edu.

5 References

[1] WinHex 12.1, available at http://www.winhex.com

[2] Forensic Toolkit (FTK), available at http://www.accessdata.com.

[3] V. Roussev, G. G. Richard III, "Breaking the Performance Wall: The Case for Distributed Digital Forensics," *Proceedings of the 2004 Digital Forensics Research Workshop (DFRWS 2004)*, Baltimore, MD.

[4] Knoppix: A Live Linux CD based on Debian GNU/Linux, available at http://www.knoppix.net.

[5] K. Shanmugasundaram, "Automated Reassembly of Fragmented Images," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[6] Foremost 0.69/1.0beta, available at http://foremost.sourceforge.net/

[7] B. Carrier, "Open Source Digital Forensics Tools: The Legal Argument," @stake Research Report.

[8] E. Kenneally, "Gatekeeping Out of the Box: Open Source Software as a Mechanism to Assess Reliability for Digital Evidence," *Virginia Journal of Law and Technology* 13 (2001).

[9] Knoppix Security Tools Distribution (STD), http://www.knoppix-std.org/.

[10] Helix Live CD, http://www.e-fense.com/helix/.