Chapter 1

IN-PLACE FILE CARVING

Golden G. Richard III, Vassil Roussev, and Lodovico Marziale
Department of Computer Science, University of New Orleans
New Orleans, Louisiana 70148, USA
{golden,vassil, lmarziale}@cs.uno.edu

Abstract

File carving is the process of recovering files from an investigative target, potentially without knowledge of the file system structures. The process is based on information about the format of the file types of interest, as well as on assumptions about how files are typically laid out on block devices. If the filesystem metadata is used at all, it is typically used only for establishing cluster sizes and avoiding carving of undeleted files (which can be extracted without file carving).

Current generation file carvers make copies of recovered files. Unfortunately, it is common to end up with a large volume of false positives during a file carving operation. These false positives are "junk" files that have invalid formats and can consume a large amount of disk space. In this paper, we present an *in-place* approach to file carving, which allows inspection of recovered files without actually copying file contents. This results in significant reduction in storage requirements (even in pathological cases), much shorter turnaround times, and opens up new opportunities to perform on-the-spot screening of evidence. Our system can perform in-place carving on both local and remote drives.

Keywords: Digital forensics, file carving, in-place carving

1. Introduction

File carving is a useful technique in digital forensics for recovering files from an investigative target, potentially without knowledge of the file system structures. The process is based on information about the format of the files of interest, as well as on assumptions of how the file data is usually laid out on the block-level device. If the filesystem metadata is used at all, it is typically used only for establishing cluster sizes and avoiding carving of undeleted files (which can be extracted without file carving).

[0]This work was supported in part by the National Science Foundation under grant # CNS-0627226.

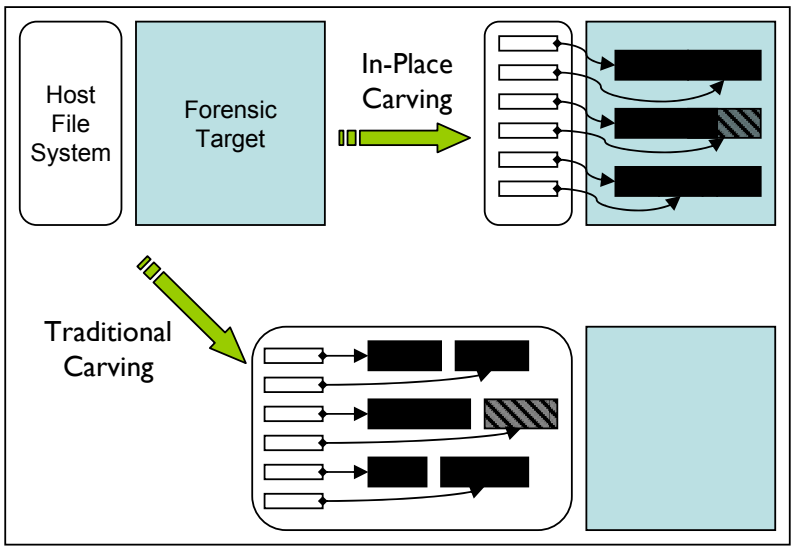

Figure 1. Conceptual differences between traditional and in-place carving. The host filesystem and forensic target can be thought of as "input" to the carving process, in both the traditional and in-place cases. For in-place carving, a database of metadata is inserted into the host file system, indicating where potentially interesting files are located in the target. For traditional carving, both the metadata and the data for carved files is dumped into the host filesystem and the target has no significant role after the carving operation completes.

Unfortunately, the current practice of carving recovered data into new files carries a huge performance penalty that is inherent and cannot be completely solved by optimizing the carving application. The *only* justification of this approach is that virtually all tools used to view or process the recovered data require a file-based interface to the data. A new strategy is needed: providing a filesystem interface to the output of the carver without actually creating carved files. In other words, if we provide a file system interface to the candidate files without physically recreating them, then we can still use our tools without paying the price for creating new files, many of which will likely be invalid. We refer to this approach as *in-place file carving*. Our approach is very similar to that used by current filesystems, except that filesystem metadata is stored *outside* the target. Figure 1 provides an illustration of the conceptual differences between traditional and in-place carving.

To understand the motivation behind our work, consider a recent data recovery case, in which carving a wide range of file types for a modest 8GB target yielded over 1.1 million files, with a total size exceeding the capacity of one of our 250GB drives. This represents an 'over-carving' factor surpassing 32. Clearly, this is a pathological case and an expert user might be able to substantially reduce both the number of carved files and the needed storage by tinkering with the carving rules. Yet it demonstrates that surprises can come even in small packages. With drives in the 200-300GB being typical for average desktop systems, even an over-carving factor of 2-3 puts a significant burden on an investigator's resources. First, he must purchase

and dedicate a substantial amount of temporary storage for file carving. Then, he must pay a substantial price in performance that might easily add hours or days to the investigation.

It is not difficult to pinpoint the reasons for such a huge perfomance penalty. The file carving tool we use in our experiments, *Scalpel* [9], allows us to easily separate the time it takes to identify the candidate files from the the time it takes to recreate them in the host file system. In Scalpel, these two tasks are performed during two separate passes over the target. During the first pass, the entire target is processed sequentially to identify the locations of candidates for carving. The result is a list of carving jobs indicating which sequences of blocks constitute candidate files. Given a non-trivial number of rules, the process is CPU-bound, because a large number of binary string searches must be performed. During the second pass, the carving jobs are carried out by placing copies of the identified data blocks into newly created files. This is a completely I/O-bound process with write access to the host file system being the limiting factor. In general, if there are a large number of candidates to carve, the second pass takes substantially longer.

Although the root of the problem is false positives generated by the carving rules, the real performance penalty is a result of recreation of the carved files. Specifically, the main culprit is the randomized set of write accesses generated during this process, which causes worst-case performance for mechanical hard drives, since writing disk blocks and updating filesystem metadata generate non-sequential disk accesses. Obviously, *any* level of over-carving will guarantee randomized disk accesses as the same data block is copied to more than one new location on the host file system and the corresponding file metadata is updated. It is worth observing that any optimizations based on the interleaving/pipelining of Scalpel's two passes (as some other tools attempt to do) has very limited potential and will not solve this problem, as the creation of carved files clearly dominates total execution time.

Arguably, the penalty for recreating the candidates goes beyond the carving itself and is carried into the subsequent processing steps. Consider our 8GB target and a 'reasonable' over-carving factor of two and assume that the goal is to carve out all JPEG images. On a modern workstation with 4GB of RAM, 40-45 percent of the target could conceivably be cached. With proper clustering of file accesses based on information generated by the carving application, it is possible to take advantage of the caching effects most of the time. In contrast, reading 16GB of recreated files yields *no* tangible caching benefits due to non-overlapping file operations.

For the rest of this section we explore two baseline scenarios where in-place carving can make a significant difference.

1.1 Large-scale carving

It is increasingly common to encounter terabyte scale targets in digital forensics investigations, with terabyte-level storage solutions becoming a commodity item. We consider such targets 'large-scale' and note that performance problems in dealing with these targets are compounded by the fact that high-capacity drives tend to be noticeably slower than the fastest drives available. However, we use the term 'large-scale' in a relative sense–any target that would stretch or exceed currently available resources would be viewed as large-scale by the

examiner, so even a 100GB target could prove to be a challenge. Economy is also an important factor. The original design of Scalpel was motivated by existent file carvers requiring "elite"-class hardware to perform acceptably. We now address a different, but related question: Should an investigator who wants to recover data from a 200GB drive using file carving be expected to have 1TB of available disk space available? Or more?

The answer is no, and in-place carving is far more scalable than traditional carving techniques in terms of storage requirements. The contents of a target are dominated by file data with metadata usually taking up well under five percent. By choosing not to copy file content (often more than once), we limit the overhead to a small fraction of the target size. In our 8GB example, the new metadata for 1.1 million candidate files would take up less than 128MB if we allow 128 bytes of metadata per file, or about 1.6 percent of the target size. Extrapolating to a 1TB target, the metadata is still small enough to fit on a miniature USB drive.

A second aspect of scalability is turnaround time. How long after inititating file carving operations can the examiner begin work in earnest on the results? Scalpel v1.60's preview mode performs only the first carving pass and then outputs a database of locations for candidate files. This information is sufficient for the in-place carving architecture described later in the paper to recreate metadata for candidate files on-the-fly. This means that we can deliver any of the candidate files from the 8GB drive mentioned earlier after only about 70 minutes of work.

A third aspect of providing a scalable solution is the flexibility to quickly react to the specifics of the target and adjust the carving settings to minimize false positives or refocus the investigation. Traditionally, investigators must often make several file carving attempts to adjust maximum file sizes and adjust the set of file types to carve. These adjustments are mostly motivated by performance concerns, since carving a large number of file types can be expensive. With in-place carving, a large number of file types with large maximum carve sizes can be specified without incurring significant performance penalties.

1.2 Triage

It is often desirable to preview a set of possible targets (or a very large target) to judge the relevance of the data on the target to an inquiry. This assessment can be performed either in a forensic lab setting or in the field and can help an investigator prioritize and filter out targets. In the case of file carving, only an in-place approach will deliver the necessary short turnaround time.

Going a step further, it is possible to perform in-place file carving on live machines, which cannot be taken down for practical or legal reasons. Live carving has the potential to increase voluntary cooperation by the owners of the machines, because critical machines can run uninterrupted. While it would be more challenging to present the results of live file carving operations in court, the results may be useful in justifying a search warrant for seizure of the equipment and a more stable "dead" investigation. In a less restrictive legal environment, such as an internal corporate inquiry, where the goal is not to go to court but to solve a specific problem, live carving has even more applications. The primary benefit of an in-place approach to live carving is that large amounts of additional storage are not required. In fact, using network

block device [7] connectivity between the target and an investigator's machine, only additional storage for metadata for carved files is required. This can be stored on a small portable USB device, such as a thumbdrive.

Another important aspect of in-place carving is that it preserves the privacy of the original information because no copies of the file data are made. Furthermore, it is easy to convince the owner of that since the investigator is unlikely to need anything more than a flash USB drive to initiate and store the carving results.

The forensic triage (whether in the lab or on the spot) could potentially be performed in parallel on a group of machines controlled by a single investigator, over a local network. The idea of on-the-spot forensic investigation has already been explored in a more general setting by the *Bluepipe*[4] project. In that context, file carving is just a special function that could be performed and controlled in parallel.

So far, we have argued that traditional file carving aproaches, which create new files for each candidate, are wasteful and carry significant and unavoidable performance overhead. The main contribution of this work is to present an architecture for in-place carving and to demonstrate that the benefits of file-based access to the candidates can be realized by recreating file metadata without copying file contents.

2. Related Work

In this section, we provide a quick overview of some representative work on file carving and the technologies upon which we depend for developing in-place carving tools.

2.1 File Carving

File carvers (e.g., [2][9]) read databases of file headers, footers, and rules defining specific file types and then search target disk images for occurrences of files of these types. The headers and footers are typically binary character strings and the rules attempt to reduce false positives. The rules may, for example, associate the footer closest to a discovered header or indicate that files should be no larger than a specified size. 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.

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 file-system's metadata can be quite fragile, file data is much more resilient. The problem with current generation file carvers is that they may require a large amount of additional storage, because large numbers of false positives are generated. Better rulesets for guiding carving operations can reduce the number of false positives, but the benefit of the proposed scheme, in-place carving, is that virtually no additional storage is required and the time until an investigator can begin to preview results is substantially reduced over traditional approaches.

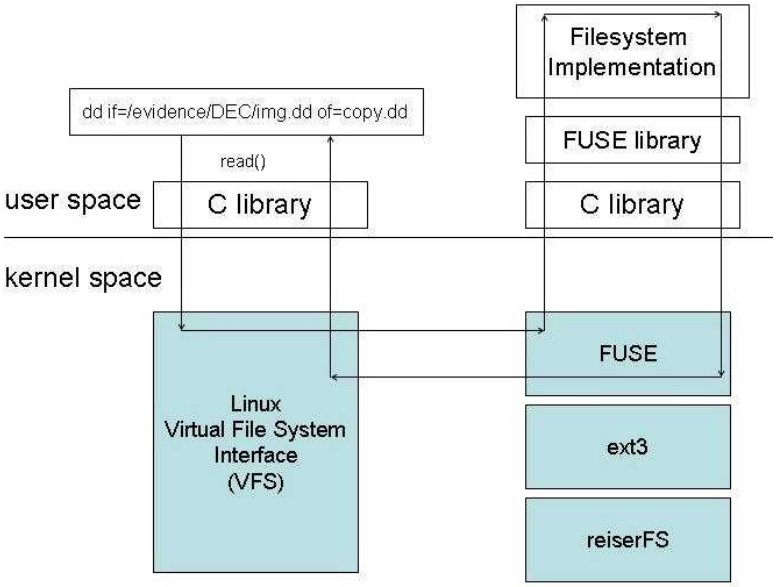

Figure 2. FUSE architecture.

2.2 User-space Filesystems

FUSE [3] is a system for rapid development of novel filesystems. The architecture of FUSE is illustrated in Figure 2. A FUSE kernel component, which implements a Virtual File System (VFS) filesystem, traps system calls and redirects them to a user-space filesystem implementation, compiled against the FUSE library. This allows new filesystems to be quickly designed and built without the complexity of in-kernel hacking. The FUSE kernel module acts as a bridge to the VFS kernel interfaces.

To instrument system calls in FUSE, the new filesystem supplies a table of functions that redefine standard system calls, such as open, read, write, and close. Each of the functions can completely redefine a filesystem operation or augment its functionality.

FUSE currently has ports for Linux and FreeBSD and a number of language bindings, including C, C++, Python, and Perl. FUSE is being actively developed and is in widespread use; one of the most important uses of FUSE is for NTFS support in Linux [6]. As of kernel version 2.6.14, FUSE is integrated into the Linux kernel. Our in-place carving system uses FUSE to present a standard filesystem interface for files carved in-place.

2.3 Networked Access to Physical Block Devices

Network block device [7][10] provides a traditional, local interface to a remote (and sometimes distributed) block device. The goals of using a network block device are typically increased performance[5][10] or accessibility [7]. We use the Linux network block device (NBD) [7] to

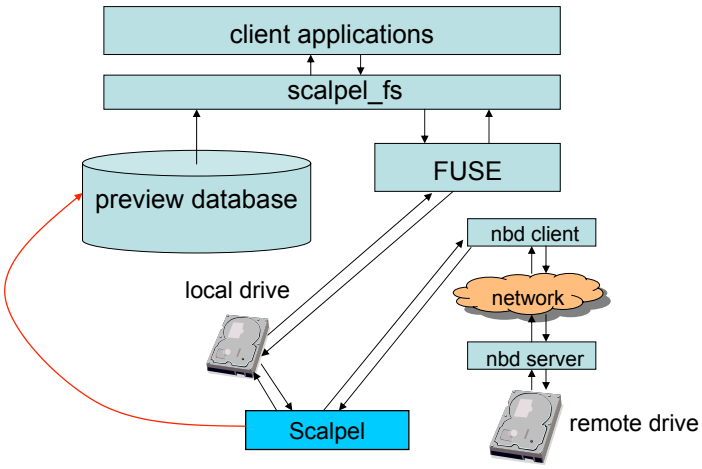

Figure 3. The in-place file carving architecture.

allow in-place carving of live remote targets. The NBD simulates a block device, such as as a hard disk or hard disk partition, using a Linux kernel module and user-space application on the client side and a user-space application on the server side. Ports of the server-side application are available for virtually all Unix platforms and for Microsoft Windows.

2.4 CarvFs

In parallel with the work on scalpel_fs, the Dutch National Police Agency has been working on a very similar project called CarvFs [11]. CarvFs, unlike scalpel_fs, does not use a database to store any metadata, but instead relies on designation and file path properties. The DNPA refers to this technique as 'zero-storage' rather than 'in-place' carving, but with the exception of the use of a database, the two are basically equivalent. Further where scalpel_fs focuses on accessing disks either directly or using smb, CarvFs focuses on accessing disk images (currently primarily ewf). The current release of CarvFs provides patched versions of some sleuthkit [8] tools, and a post-processing script to work with the scalpel preview mode. Currently the viability of integrating the scalpel_fs and CarvFs efforts is being investigated.

3. In-Place File Carving

3.1 Architecture

In order to achieve the time and space savings characteristic of in-place carving, a multi-level system is employed. Our in-place carving architecture, ScalpelFS, is comprised of three major components: Scalpel v1.60, which provides a new "preview" mode, a custom FUSE filesystem for providing a standard filesystem view of carved files, and the Linux network block device, which allows carving of remote disk targets. We support both live and dead investigative

targets and both local and NBD disks. Similarly, carving operations may be performed either on the same machine that hosts the target disk device (or image) or on a separate investigative machine. The architecture is depicted in Figure 3.

Operating above a remote disk target is the the network block device server, which provides live, remote access to the disk. Local disks are accessed directly by the Scalpel file carver, operating in a new "preview" mode. When operating in preview mode, Scalpel executes file carving rules specified in its configuration file to identify candidate files for carving on the disk devices. These files are only potentially interesting, as current carving strategies may generate copious amounts of false positives, depending on the carver configuration. Normally, Scalpel would then carve the disk blocks associated with candidate files and write them out to new files. In preview mode, however, Scalpel produces entries in a database detailing the starting position of the file on the device, whether or not the file was truncated, its length, and the device where it resides. The format is as follows:

```
filename           start      truncated       length    image

...
htm/00000076.htm   19628032   NO                239    /tmp/linux-image
jpg/00000069.jpg   36021248   NO             359022    /tmp/linux-image
htm/00000078.htm   59897292   NO                 40    /tmp/linux-image
jpg/00000074.jpg   56271872   NO              16069    /tmp/linux-image
...
```

The original names of files are typically not available when file carving is used for data recovery, so Scalpel assigns a unique pathname to each carved file. The pathname indicates where the files would be created if Scalpel were not operating in preview mode.

Our custom FUSE filesystem, *scalpel_fs*, accesses the Scalpel preview database and the targeted disk devices. The *scalpel_fs* application is implemented in C against the FUSE API and provides the standard Linux filesystem interface to files described in the Scalpel preview database, without carving the files from the target device. Executing *scalpel_fs* with arguments detailing a directory to be used as a mount point, a Scalpel preview database and the device that the database was generated from causes the following actions to be taken. First, a directory named *scalpel_fs* is created under the mount point. Then, using the carved file specifications in the preview database, a tree of filesystem objects is created in a structure determined by the filenames of the entries in the database. Files and directories are modeled as *fs_object* structs:

```
struct fs_object {
   int type;                   // file or directory
   char *name;                 // this object's fully qualified pathname
   int start;                  // starting index in the source image of the file
   int length;                 // size in bytes
   char clipped;               // true if the file was truncated
   char *source;               // the source image file name
   struct fs_object *children; // empty for files
   struct fs_object *next;     // peer nodes
}
```

This tree structure provides the information necessary to present the user with a filesystem appearing to contain the carved files from the target devices. For efficiency, a pointer to each *fs_object* is also entered into a hash table for fast lookups. Listing the contents of the mounted directory shows one directory named *scalpel_fs*. Appearing inside the *scalpel_fs* directory are files and directories mirroring those in the filesystem tree created from the Scalpel preview database.

All file-oriented system calls targeting the mount point are intercepted. Preceding most filesystem operations is a call to *getattr* for the filesystem object in question, which returns a *stat* structure containing information such as object type (file, directory, etc.), size, creation, access and modification times, and permissions. On receiving the *getattr* call, *scalpel_fs* dynamically constructs and returns a new *stat* structure with type and size taken from the *fs_object* for the object and creation/modification/access times and permissions duplicating those of the *scalpel_fs* directory. Directory listings are created on the fly using the *children* structures of the *fs_object* for the directory being listed. Opening a file returns a file handle (after checking the existence of the *fs_object* in the hash table).

Attempts to read a file are handled as follows: the target device is opened and reading begins at the offset given by the *start* member of the *fs_object* struct for the file (plus any offset passed to the read operation itself). This is all transparent to the client application (and to the user). Other non-write filesystem operations also work transparently (e.g., *access*, *getattr*, *readdir*, etc..) Any operations that would create content (*write*, *mkdir*, *link*...) are disallowed in order to maintain the forensic soundness of the target. An exception is the delete (*unlink*) operation, which is allowed, but only in a shallow manner: the *fs_object* for the deleted file is removed but the target disk device is left untouched. This removes the file from the view provided by *scalpel_fs* without destroying any data.

At the top level of the system are other user-level applications. They can freely and transparently operate on the files under the mount point as if they were regular files (aside from the disallowed write operations). A user can obtain cryptographic hashes of the files with hashing programs, view the files in text editors or image viewers, or use specialized forensics software on the files. This is particularly useful in an investigation involving image files (e.g., JPG or GIF images) as the images can be previewed as thumbnails by most filesystem browsers. Note that all of this occurs without the large amounts of space required by a normal carving operation.

3.2 Performance Study

We ran a number of experiments to test the advantages of in-place carving. All of the reported results came from experiments conducted against an 8GB target and a 100GB target.

Figures 4 presents results for our experiments with an 8GB target, in which an empty 250GB IDE drive was used to store carving results. The machine performing the carving operations was a 933MHz Pentium III with 512MB of RAM, running CentOS Linux v4.3. An "unrestricted" full carve of the 8GB drive, using all the file types supported in the standard Scalpel v1.60 configuration file, crashed with an out of disk space error 7% into Scalpel's second carving pass, after carving almost 1,000,000 files. Thus the necessary disk space for completing this

Carving Description	Execution Time	Total # of files carved	Total disk space required
regular, all file types, 8GB local disk image	Out of disk space	---	>> 250GB
preview, all file types, 8GB local disk image	80m56s	1,125,627	62MB (for metadata)
regular, constrained set of file types, 8GB local disk image	222m11s	724,544	212GB
preview, constrained set of file types, 8GB local disk image	63m22s	724,544	39MB (for metadata)
regular, image file formats only (JPG, GIF, PNG, BMP), 8GB local disk image	60m35s	9,068	5.9GB
preview, image file formats only (JPG, GIF, PNG, BMP), 8GB local disk image	26m28s	9,068	500K (for metadata)

Figure 4. Experimental results for 8GB target.

carving operation would be much larger than 250GB. In contrast, using Scalpel's "preview" mode, the same carving operation completed in about 1 hour and 20 minutes, yielding an index for 1,125,627 files.

For a more concrete comparison, the number of carved file types was reduced substantially and the test was repeated. This time, the full carve took about 3 hours and 42 minutes, carving 724,544 files which consumed a total of 212GB of disk space. In contrast, the preview carve took only 1 hour, 3 minutes and consumed only 39MB, for the in-place carving metadata.

Reducing the number of file types further and carving only image file formats (JPG, GIF, PNG, BMP), resulted in 9,068 carved files in approximately 60 minutes, requiring 5.9GB of storage for a normal carving operation. A preview carve with the same parameters finished in approximatley 26 minutes and required 500K of storage for metadata. The results in this case are obviously much closer, but in-place carving is still more efficient and in cases where it is important that no copies of the files are created until the investigation progresses (e.g., in a child pornography case), in-place carving has advantages over traditional carving.

We were also interested in measuring the performance of carving operations over Linux's network block device (NBD), since we support carving of live remote disk targets over NBD. We ran a simple experiment using the Pentium III machine described above as the NBD server, providing access over a quiet gigabit network to the 8GB disk image. A Thinkpad T40p with a 1.6GHz Pentium 4M processor and 2GB of RAM, also running CentOS Linux, was used to perform carving operations against the 8GB image over NBD. Carving only JPG images (in preview mode) from the 8GB target, 16m10s were required. We adjusted the network and performed the carve over a 100Mb LAN, which increased the time to 31m10s. A local preview

Carving Description	Execution Time	Total # of files carved	Total disk space required
preview, restricted file types, 100GB, local	103m30s	1,338,766	71MB (for metadata)
preview, restricted file types, 100GB, NBD	131m27s	1,338,766	71MB (for metadata)
regular, restricted file types, 100GB, local	---	1,338,766	5.9TB
preview, image file formats (JPG, GIF, PNG, BMP), 100GB, local	77m15s	470,181	25MB (for metadata)
preview, image file formats (JPG, GIF, PNG, BMP), 100GB, NBD	106m27s	470,181	25MB (for metadata)
regular, image file formats (JPG, GIF, PNG, BMP), 100GB, local	---	470,181	313GB

Figure 5. Experimental results for 100GB target.

carve, using a copy of the 8GB image loaded on the T40p, took 7m40s. Additional tests of NBD performance were conducted against the 100GB target, discussed below.

We also conducted experiments using a 100GB target. The results are illustrated in Figure 5. For the experiments with the 100GB target, the machine performing the carving operations was a 3GHz Pentium 4 with 2BG of ram, also running Centos 4.3. Using Scalpel in preview mode to carve a heavily restricted set of files types resulted in 1,338,766 files which, if carved by traditional methods would have taken 4.9TB of space. Our system used 70MB of space and took 1 hour and 43 minutes.

Carving the 100GB target for only the image file formats listed above resulted in 470,181 files, requiring 25MB of space and taking 1 hour and 17 minutes. If these files were copied out of the image they would have required 313GB.

We also performed experiments over NBD with the 100GB target. Here we used 2 Pentium 4 machines as described above working over an unloaded gigabit network. Preview carving for the set of image file types listed above took 1 hour and 46 minutes. Preview carving for the heavily restricted set of file used above took 2 hours and 11 minutes.

4. Conclusions and Future Work

Traditional file carving applications often require large amounts of disk space to execute because they make copies of carved files. Since many of these "recovered" files are actually false positives, the amount of data carved can exceed the size of the target by an order of magnitude. For larger targets, even more typical over-carving factors of 2-3 can require too much disk space and have an unacceptable impact on execution time.

This paper proposes a new approach called *in-place* carving, which recreates file metadata outside the target and uses the original forensic image for retrieving file contents on-demand. Our strategy allows carving to be performed faster and with substantially reduced disk storage requirements, without losing any functionality. In-place file carving also allows new usage scenarios, such as large-scale carving and on-the-spot carving. Our architecture uses a custom filesystem, the preview mode of the Scalpel file carver, and the Linux network block device. Both "live" and "dead" forensic targets are supported and carving operations can be executed either on the machine hosting the target disk or on a separate investigative machine.

While our in-place carving architecture is functional, we are working on a number of issues. Currently, ScalpelFS does not support Scalpel's options for carving fragmented files. We are also working to reduce an investigator's "wait time" by presenting a dynamically updated view of the file system as file carving proceeds, allowing the investigator to process files as they become available. This will require modifications to the Scalpel file carver and some minimal changes in ScalpelFS. Finally, we are investigating feedback mechanisms that perform file validation during carving operations, disabling or prioritizing carving rules depending on how many false positives are generated by particular carving rules. The goal is to provide the investigator with "good" evidence, as quickly as possible, and delay processing of files that are unlikely to be useful.

5. Acknowlegments

The authors are grateful to Daryl Pfeif, of Digital Forensics Solutions, for suggestions that improved the organization of the paper.

References

[1] 2006 Digital Forensics Research Workshop File Carving Challenge, http://www.dfrws.org/2006/challenge/.

[2] Foremost File Carver, http://foremost.sourceforge.net.

[3] FUSE: Filesystem In User Space, http://fuse.sourceforge.net/.

[4] Y. Gao, G. G. Richard III, V. Roussev, "Bluepipe: An Architecture for On-the-Spot Digital Forensics," International Journal of Digital Evidence (IJDE), 3(1), 2004.

[5] S. Liang, R. Noronha and D. K. Panda, "Swapping to Remote Memory over InfiniBand: An Approach Using a High Performance Network Block Device'," Proceedings of IEEE Cluster Computing, 2005.

[6] The Linux NTFS Project, http://www.linux-ntfs.org/.

[7] P. Machek, Network Block Device, http://nbd.sourceforge.net/.

[8] B. Carrier, The Sleuthkit and Autopsy, http://www.sleuthkit.org/.

[9] G. G. Richard III, V. Roussev, "Scalpel: A Frugal, High Performance File Carver," Proceedings of the 2005 Digital Forensics Research Workshop (DFRWS 2005), New Orleans, LA.

[10] D. Tingstrom, V. Roussev, G. G. Richard III, "dRamDisk: Efficient RAM Sharing on a Commodity Cluster," Proceedings of the 25th IEEE International Performance, Computing, and Communications Conference (IPCCC 2006).

[11] The Carve Path Zero-storage Library and Filesystem, http://ocfa.sourceforge.net/libcarvpath/.