Chapter 15

IN-PLACE FILE CARVING

Golden Richard III, Vassil Roussev and Lodovico Marziale

Abstract File carving is the process of recovering files from an investigative target, potentially without knowledge of the filesystem structure. Current generation file carvers make complete copies of recovered files. Unfortunately, they often produce a large number of false positives – "junk" files with invalid formats that frequently consume large amounts of disk space.

This paper describes an "in-place" approach to file carving, which allows the inspection of recovered files without copying file contents. The approach results in a significant reduction in storage requirements, shorter turnaround times, and opens new opportunities for on-the-spot screening of evidence. Moreover, it can be used to perform in-place carving on local and remote drives.

Keywords: File carving, in-place carving

1. Introduction

File carving is a useful digital forensic technique for recovering files from an investigative target, potentially without knowledge of the filesystem structure. The process is based on information about the format of the files of interest and on assumptions about how the file data is laid out on the block-level device. Filesystem metadata is typically used – if at all – only to establish cluster sizes and to avoid carving undeleted files (which can be extracted without file carving).

Unfortunately, the current practice of carving recovered data into new files carries a huge performance penalty that is inherent and cannot be solved by optimizing the carving application. The only justification for this approach is that virtually all the tools used to view or process the recovered data require a file-based interface to the data. A new strategy is needed that provides a filesystem interface to the output of the carver without actually creating carved files. In other words,

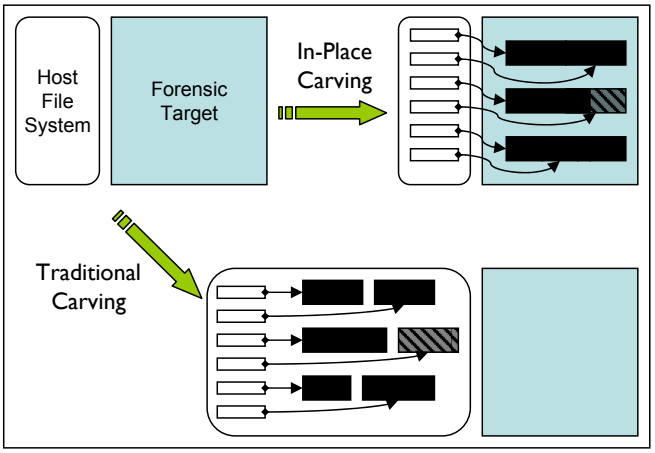

Figure 1. Conceptual differences between traditional and in-place carving.

if a filesystem interface is provided to candidate files without physically recreating them, existing forensic tools can still be used without creating new files, many of which will likely be invalid. We call this approach "in-place" file carving. The approach is similar to that used by current filesystems, except that filesystem metadata is stored outside the target.

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

To understand our motivation, consider a recent case, in which carving a wide range of file types in a modest 8 GB target yielded more than 1.1 million files that exceeded the capacity of the 250 GB drive being used. This represents an "over-carving" factor of more than 32. Clearly, this is a pathological case and an expert user might be able to substantially reduce the number of carved files and the required storage by tinkering with the carving rules. However, with drives in the 200-300 GB range being fairly typical for desktop systems, even an over-carving factor of two or three puts a significant burden on an investigator's resources. First, he must purchase and dedicate a significant amount of temporary storage for file carving. Then, he must pay a substantial price in performance that could easily add days to the investigation.

It is not difficult to pinpoint the reasons for the huge performance penalty. The Scalpel file carving tool we use in our experiments [6] allows us to separate the time taken to identify the candidate files from the time taken to recreate them in the host filesystem. Scalpel performs these two tasks in separate passes. In 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 filesystem being the limiting factor. In general, if there are a large number of candidates to carve, the second pass will require much more time than the first pass.

Although the root of the problem is the false positives generated by the carving rules, the performance penalty is the result of recreating the carved files. Specifically, the main culprit is the randomized set of write accesses generated during the carving process, which causes worst-case performance for mechanical hard drives because writing disk blocks and updating filesystem metadata generate non-sequential disk accesses. Obviously, any over-carving will guarantee randomized disk accesses as the same data block is copied to more than one new location on the host filesystem and the corresponding file metadata is updated. It is worth observing that any optimizations based on interleaving/pipelining Scalpel's two passes (as some other tools attempt to do) has limited potential and will not solve the problem. This is because 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 8 GB 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 4 GB RAM, 40-45% 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 16 GB of recreated files yields no tangible caching benefits due to non-overlapping file operations.

The following subsections 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 forensic investigations. We consider such targets as "large-scale" and note that performance problems in dealing with them 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 stretches or exceeds the available resources would be viewed as being large-scale by the examiner, so even a 100 GB target could prove to be a challenge. Economy is also an important factor. The original design of Scalpel was motivated by the fact that file carvers required "elite" hardware to perform acceptably. We now address a different, but related question: Should an investigator who wants to recover data from a 200 GB drive using file carving be expected to have 1 TB or more of available disk space?

The answer is no because in-place carving is much more scalable than traditional carving in terms of storage requirements. The contents of a target are dominated by file data with metadata usually taking up well under 5% of storage. By choosing not to copy file content (often more than once), the overhead is limited to a small fraction of the target size. In our 8 GB example, the new metadata for 1.1 million candidate files would take up less than 128 MB if we allow 128 bytes of metadata per file, or about 1.6% of the target size. Extrapolating to a 1 TB 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 initiating file carving operations can the examiner begin to 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 any of the candidate files from the 8 GB drive mentioned earlier can be delivered after only about 70 minutes of work.

A third aspect to providing a scalable solution is the flexibility to react to the specifics of the target by adjusting the carving settings to minimize false positives or refocus the investigation. Traditionally, investigators make several carving attempts to adjust maximum file sizes and the set of file types to carve. These adjustments are mostly motivated by performance concerns as 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 to an investigation. Such an assessment, which can be performed in a forensic laboratory or in the field, helps an investigator prioritize and filter 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 that cannot be taken down for practical or legal reasons. Live carving has the potential to increase voluntary cooperation by equipment owners 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 environment, such as an internal corporate inquiry, where the goal is not to go to court, live carving has even more applications. The primary benefit of live in-place carving is that large amounts of additional storage are not required. In fact, if network block device [5] connectivity is used 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 USB device).

Another important aspect of in-place carving is that it preserves privacy because no copies of the file data are made. Furthermore, it should be easy to gain owner cooperation as the investigator is unlikely to need anything more than a USB device to initiate and store the carving results.

The forensic triage – whether in the lab or in the field – could be performed in parallel on a group of machines controlled by a single investigator over a local network. The idea of an on-the-spot investigation has been explored in a more general setting by the Bluepipe Project [3], where file carving is just a special function that could be performed and controlled in parallel.

So far, we have argued that traditional file carving approaches, 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

This section describes representative work on file carving and the technologies used to develop in-place carving tools.

2.1 File Carving

File carvers (e.g., [6, 7]) read databases of file headers and 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. The rules help reduce false positives. For example, a rule may associate the footer closest to a discovered header; another rule may indicate that files should be no larger than a specified size. The goal is to identify the starting and ending locations of files in disk images and "carve" (copy) sequences of bytes into regular files.

File carving is a powerful technique because files can be retrieved from raw disk images regardless of the type of filesystem. File retrieval is possible even when the filesystem metadata has been completely destroyed. For example, a file deposited on a FAT partition often can 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 filesystem metadata is quite fragile, file data is much more resilient. The problem with current file carvers is that they require considerable additional storage as large numbers of false positives are generated. Good rules for guiding carving operations can reduce the number of false positives (and the storage needed). However, our in-place carving scheme requires virtually no additional storage. Moreover, the amount of time that an investigator has to wait to preview the results is substantially reduced.

2.2 User-Space Filesystems

FUSE [8] is a system for the rapid development of novel filesystems. The FUSE architecture is presented in Figure 2. A FUSE kernel component, which implements a Virtual File System (VFS), traps system calls and redirects them to a user-space filesystem implementation, which is 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 (e.g., `open`, `read`, `write` and `close`). Each of the functions can completely redefine the system call or augment its functionality.

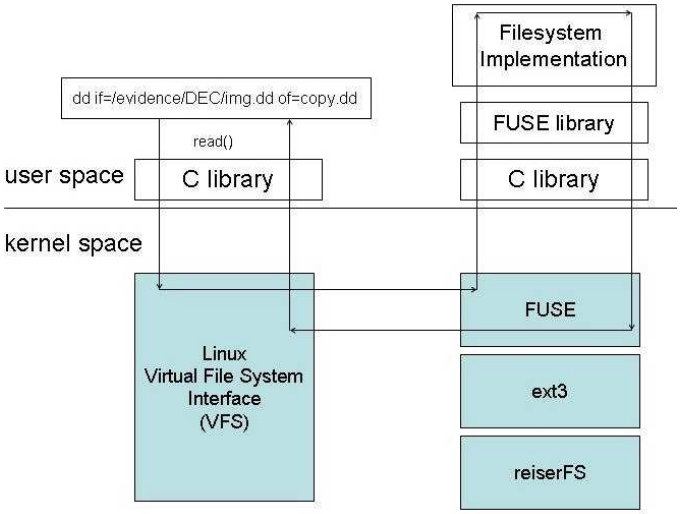

Figure 2. FUSE architecture.

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 NTFS support in Linux [10]. FUSE (kernel version 2.6.14) 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

A network block device [5, 11] provides a traditional local interface to a remote (or distributed) block device, enhancing performance [4, 11] and/or accessibility [5]. We use the Linux network block device (NBD) [5] to facilitate in-place carving of live remote targets. The NBD simulates a block device (e.g., hard disk or hard disk partition) using a Linux kernel module and a user-space application on the client side and a user-space application on the server side. Server-side application ports are available for virtually all Unix platforms and for Microsoft Windows.

2.4 CarvFs

In parallel with our in-place file carving efforts related to the ScalpelFS tool, the Dutch National Police Agency (DNPA) has been developing a similar tool called CarvFs [9]. Unlike ScalpelFS, CarvFs does not use a database to store metadata, but instead relies on designation and file

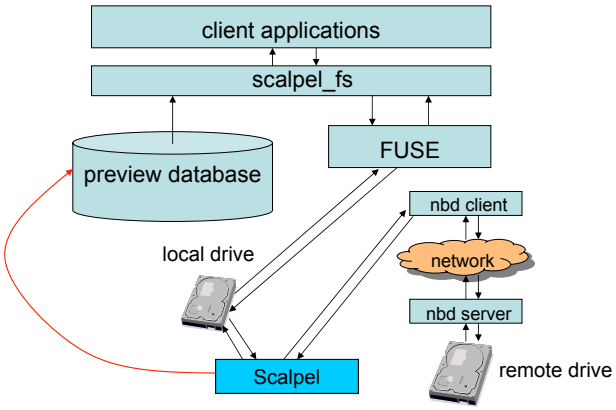

Figure 3. In-place file carving architecture.

path properties. The DNPA refers to this technique as "zero-storage" rather than "in-place" carving. Furthermore, whereas ScalpelFS focuses on accessing disks either directly or using `smb`, CarvFs focuses on accessing disk images (primarily `ewf`). The current release of CarvFs provides patched versions of some Sleuth Kit [1] tools, and a post-processing script to work with Scalpel's preview mode. Currently the viability of integrating the ScalpelFS and CarvFs efforts is being investigated.

3. In-Place File Carving

This section discusses the architecture and performance of ScalpelFS.

3.1 Architecture

A multi-level architecture is employed to achieve the time and space savings characteristic of our in-place carving approach. The ScalpelFS architecture has three main components: Scalpel v1.60, which provides a new "preview" mode, a custom FUSE filesystem for providing a standard filesystem view of carved files, and a Linux network block device, which allows carving of remote disk targets (Figure 3). The architecture supports live and dead investigative targets, and local and NBD disks. Carving operations may be performed on the same machine that hosts the target disk device (or image) or on a separate investigative machine.

Operating above a remote disk target is the network block device server, which provides live, remote access to the disk. Local disks are accessed directly by the Scalpel file carver that operates in the preview mode. When operating in this 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 the preview mode, however, Scalpel produces entries in a database detailing the starting positions of files on a device, whether or not the files are truncated, their length, and the devices on which they reside. These entries have the following format:

```
    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 source image of 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`. Inside the `scalpel_fs` directory are files and directories that mirror 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; this 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 in 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 first checking that the `fs_object` is 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` and `readdir`). Operations that create content (e.g., `write`, `mkdir` and `link`) are disallowed 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 forensic 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 need to use large amounts of storage space as in the case of a normal carving operation.

Table 1. Experimental results for an 8 GB target.

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)

3.2 Performance Study

We conducted several experiments to test the advantages of in-place carving. This section reports the results of experiments conducted with 8 GB and 100 GB targets.

Table 1 presents the experimental results for an 8 GB target; an empty 250 GB IDE drive was used to store the carving results. The machine performing the carving operations was a 933 MHz Pentium III with 512 MB RAM running CentOS Linux v4.3. An "unrestricted" full carve of the drive, using all the file types supported in the standard Scalpel v1.60 configuration file, crashed with an out of disk space error 7% during Scalpel's second carving pass (after almost 1,000,000 files were carved). Clearly, the disk space required to complete this carving operation would be very much more than 250 GB. In contrast, using Scalpel's "preview" mode, the same carving operation completed within 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 that consumed 212 GB of disk space. In contrast, the preview carve took only 1 hour and 3 minutes, and consumed only 39 MB (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

Table 2. Experimental results for a 100 GB target.

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

approximately 60 minutes, requiring 5.9 GB of storage for a normal carving operation. A preview carve with the same parameters finished in 26 minutes and required 500 K of storage for metadata. The results in this case are obviously much closer, but in-place carving is still more efficient. Also, in-place carving has advantages over traditional carving in situations where it is important that no copies of the files are created until the investigation progresses (e.g., child pornography cases).

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 8 GB disk image. A Thinkpad T40p with a 1.6 GHz Pentium 4M processor and 2 GB of RAM, also running CentOS Linux, was used to perform carving operations over NBD. Carving only JPG images (in preview mode) required 16 minutes and 10 seconds. Performing the same carve over a 100 megabit LAN increased the time to 31 minutes and 10 seconds. A local preview carve, using a copy of the 8 GB image loaded on the T40p, took 7 minutes and 40 seconds.

Table 2 presents the results of experiments conducted on a 100 GB target. The machine performing the carving operations was a 3 GHz Pentium 4 with 2 GB RAM 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 required 4.9 TB of space. Our system used approximately 70 MB of space and took 1 hour and 43 minutes.

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

We also performed experiments over NBD with the 100 GB 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 types took 2 hours and 11 minutes.

4. Conclusions

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 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 two or three can require too much disk space and have an unacceptable impact on execution time.

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

Several enhancements to the in-place carving architecture are being undertaken. 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 filesystem 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 to 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 to delay the processing of files that are unlikely to be useful.

Acknowledgements

This work was supported in part by NSF Grant CNS 0627226. The authors are grateful to Daryl Pfeif of Digital Forensics Solutions for suggestions that improved the organization of the paper.

References

[1] B. Carrier, The Sleuth Kit (www.sleuthkit.org).

[2] Digital Forensics Research Workshop (DFRWS), File Carving Challenge – DFRWS 2006 (www.dfrws.org/2006/challenge).

[3] Y. Gao, G. Richard III and V. Roussev, Bluepipe: An architecture for on-the-spot digital forensics, *International Journal of Digital Evidence*, vol. 3(1), 2004.

[4] S. Liang, R. Noronha and D. Panda, Swapping to remote memory over InfiniBand: An approach using a high performance network block device, *Proceedings of IEEE International Conference on Cluster Computing*, 2005.

[5] P. Machek, Network Block Device (nbd.sourceforge.net).

[6] G. Richard III and V. Roussev, Scalpel: A frugal, high performance file carver, *Proceedings of the Fifth Annual Digital Forensics Research Workshop* (www.dfrws.org/2005/proceedings/index.html), 2005.

[7] SourceForge.net, Foremost 1.4 (foremost.sourceforge.net), February 4, 2007.

[8] SourceForge.net, FUSE: Filesystem in Userspace (fuse.sourceforge.net).

[9] SourceForge.net, The Carve Path Zero-Storage Library and Filesystem (ocfa.sourceforge.net/libcarvpath).

[10] The Linux NTFS Project (www.linux-ntfs.org).

[11] D. Tingstrom, V. Roussev and G. Richard III, dRamDisk: Efficient RAM sharing on a commodity cluster, *Proceedings of the Twenty-Fifth IEEE International Performance, Computing and Communications Conference*, 2006.