Chapter 9
File Carving

Learning Objectives
The objectives of this chapter are to:

- Understand fundamentals concept and techniques of file carving
- Understand advanced state-of-the-art file carving techniques
- Become familiar with popular open-source file carving tools

In previous chapters, we have learned how file system structures, particularly the residual file system metadata or information about the deleted files and directories, can be utilized and analyzed to recover deleted or lost files or data. As discussed in Chaps. 6 and 8, recovering deleted files from a file system with residual file system metadata is a simple task; many programs are available to the average home use to do this. Nevertheless, savvy criminals are aware of the digital trail they left behind and attempt to delete whatever evidence they can in such a way that a tool which makes use of file system structure cannot recover. For example, they can simply format the partition where the file system resides. As a result, it makes these traditional recovery methods based on file system metadata remnants ineffective. A more sophisticated data recovery solution which does not rely on this file system structure is therefore necessary. These new and sophisticated solutions are collectively known as file carving, and tools, using such technology, are called file carvers. File carving is the process of reconstructing files based solely on their contents. In other words, the technique of recovering files from a block of binary data without using any information available in the file system structure. As early as 2002, research in digital forensics focused on the recovery of data files independent of the file metadata was begun. Using only the content found inside the data blocks themselves, researchers have attempted to reconstruct files in whole or in part using techniques that will be

X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_9

discussed in this chapter. In this chapter, we'll study file carving, a branch of digital forensics that reconstructs data from a digital device without any prior knowledge of the data structure, size, content or type of files located on the storage medium. Also, we will get familiar with open source file carvers.

9.1 Principles of File Carving

File carving is the newest technique of recovering data from digital devices and does not rely on any information located in the file system [3, 4, 7]. There are many tools available, using such technology, and they are called file carvers. File carvers are able to recover files using only the information available in the data blocks stored on the disk. A non-fragment file could be carved with the same ease as contiguous files using traditional methods relying on file system structures because many files have unique header/footer structures. Using information about their unique headers, footers (or the file size), and even internal data structure, file carvers reassemble contiguous data blocks beginning with the file header and ending at the file footer, if present. This type of file carving technique is also known header/footer carving. Figure 9.1 shows the file format of the Bitmap image file (BMP).

In Fig. 9.1, it can be seen that a BMP file starts with a unique two-byte header, "42 4D", and the size of the BMP file in bytes is given next at byte offsets 3–6. Figure 9.2 shows a hex dump of the first 512 bytes of a BMP file by using a Linux utility *xxd*. The left side (the 8 digits before each colon) is the offset address, in hexadecimal format, which is used to locate individual bytes (start at byte offset 0). The middle is the hex dump data, and the right is the ASCII interpretation of the dump data. Each byte represents a two-digit hexadecimal number. As can be seen in Fig. 9.2, the first two bytes are "0x424D", which is a special marker indicating the beginning of a BMP file, and 4 bytes of data starting at byte offset 3 is the file size, which is "38 04 04 00". However, since the computer here uses the little-endian system, the real value of this 4-byte number is "0x00040438". It means the file size is 1854 bytes.

9.1.1 Header/Footer Carving

Header/footer carving, also known as file structure-based carving, is based on the fact that most known files have distinctive header/footer structures which can be used to identify the start and end of a file. Obviously, it is very straightforward to recover deleted files by simply putting everything together between the header and the next corresponding footer as our target file given that most of files are stored in contiguous blocks (Fig. 9.3).

The growing demand by forensic investigators and corporations to collect digital evidence from devices with damaged or missing file system structures led to a wave of file carving tools that could carve contiguous files automatically. For example,

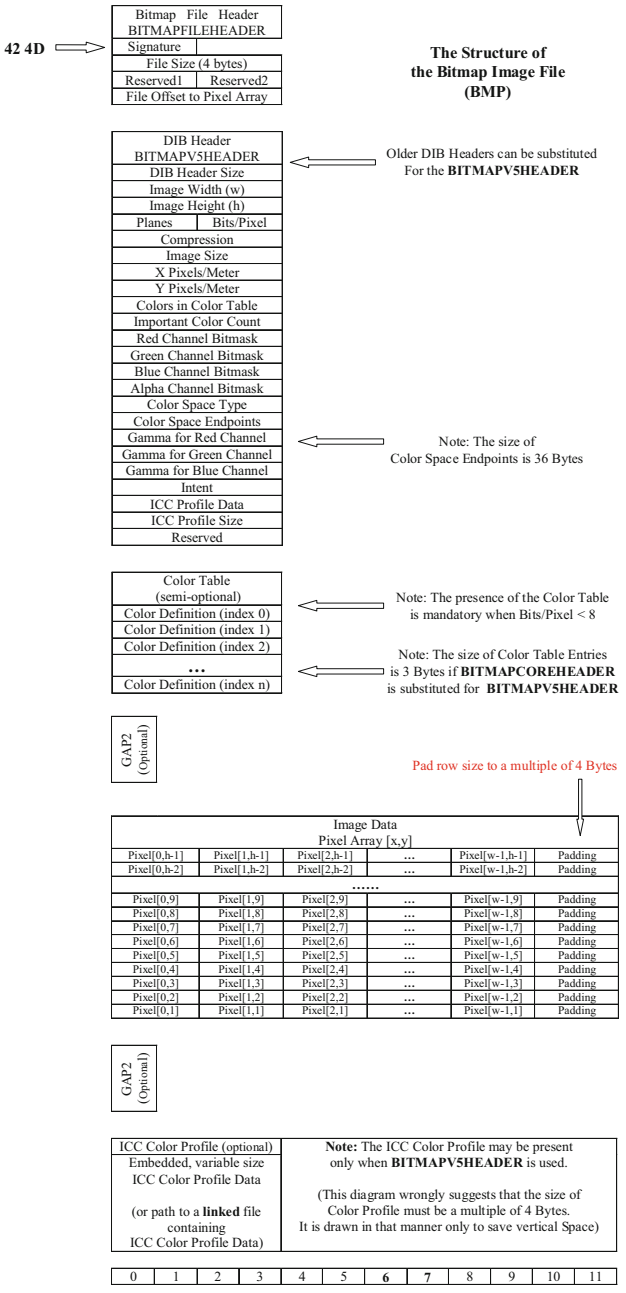


Fig. 9.1 The structure of the bitmap image file [1]

```
0000000: 424d 3804 0400 0000 0000 3604 0000 2800  BM8.......6...(.
0000010: 0000 0002 0000 0002 0000 0100 0800 0000  ................
0000020: 0000 0000 0000 120b 0000 120b 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0101 0100 0202  ................
0000040: 0200 0303 0300 0404 0400 0505 0500 0606  ................
0000050: 0600 0707 0700 0808 0800 0909 0900 0a0a  ................
0000060: 0a00 0b0b 0b00 0c0c 0c00 0d0d 0d00 0e0e  ................
0000070: 0e00 0f0f 0f00 1010 1000 1111 1100 1212  ................
0000080: 1200 1313 1300 1414 1400 1515 1500 1616  ................
0000090: 1600 1717 1700 1818 1800 1919 1900 1a1a  ................
00000a0: 1a00 1b1b 1b00 1c1c 1c00 1d1d 1d00 1e1e  ................
00000b0: 1e00 1f1f 1f00 2020 2000 2121 2100 2222  ......   .!!!.""
00000c0: 2200 2323 2300 2424 2400 2525 2500 2626  ".###.$$$.%%%.&&
00000d0: 2600 2727 2700 2828 2800 2929 2900 2a2a  &.'''.(((.))).**
00000e0: 2a00 2b2b 2b00 2c2c 2c00 2d2d 2d00 2e2e  *.+++.,,,.---...
00000f0: 2e00 2f2f 2f00 3030 3000 3131 3100 3232  ../// .000.111.22
0000100: 3200 3333 3300 3434 3400 3535 3500 3636  2.333.444.555.66
0000110: 3600 3737 3700 3838 3800 3939 3900 3a3a  6.777.888.999.::
0000120: 3a00 3b3b 3b00 3c3c 3c00 3d3d 3d00 3e3e  :.;;;.<<<.===.>>
0000130: 3e00 3f3f 3f00 4040 4000 4141 4100 4242  >.???.@@@.AAA.BB
0000140: 4200 4343 4300 4444 4400 4545 4500 4646  B.CCC.DDD.EEE.FF
0000150: 4600 4747 4700 4848 4800 4949 4900 4a4a  F.GGG.HHH.III.JJ
0000160: 4a00 4b4b 4b00 4c4c 4c00 4d4d 4d00 4e4e  J.KKK.LLL.MMM.NN
0000170: 4e00 4f4f 4f00 5050 5000 5151 5100 5252  N.OOO.PPP.QQQ.RR
0000180: 5200 5353 5300 5454 5400 5555 5500 5656  R.SSS.TTT.UUU.VV
0000190: 5600 5757 5700 5858 5800 5959 5900 5a5a  V.WWW.XXX.YYY.ZZ
00001a0: 5a00 5b5b 5b00 5c5c 5c00 5d5d 5d00 5e5e  Z.[[[.\\\.]]].^^
00001b0: 5e00 5f5f 5f00 6060 6000 6161 6100 6262  ^.___.```.aaa.bb
00001c0: 6200 6363 6300 6464 6400 6565 6500 6666  b.ccc.ddd.eee.ff
00001d0: 6600 6767 6700 6868 6800 6969 6900 6a6a  f.ggg.hhh.iii.jj
00001e0: 6a00 6b6b 6b00 6c6c 6c00 6d6d 6d00 6e6e  j.kkk.lll.mmm.nn
00001f0: 6e00 6f6f 6f00 7070 7000 7171 7100 7272  n.ooo.ppp.qqq.rr
```

Fig. 9.2 Bitmap image file Dump Data (the first 512 bytes)

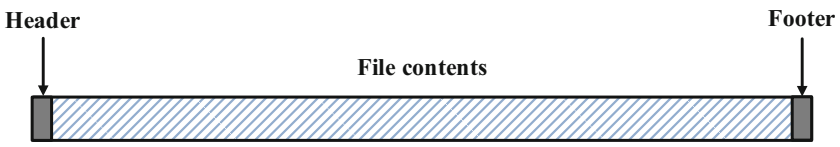


Fig. 9.3 Header/footer carving

Foremost [14] works by first creating a configuration file which contains file header/footer information of certain file format. It then tries to match each of the headers with a corresponding footer. Unfortunately this software will repeatedly search through data that has previously been matched, increasing the length of time a few search can take. Richard and Roussev proposed to fix this performance issue by creating a high performance multiple file systems carver [11]. Foremost spent much of its time reading and writing to the hard drive; often the slowest piece in a computer system. The improved file carver, Scalpel [15], first indexes all headers and footers, then looks for potential matches from within that index which is stored in memory; a much faster method than repeatedly searching the hard drive. Additionally, the software also contains improved memory-to-memory copy operations,

as well as faster byte writing output. These improvements made Scalpel a much more efficient file carving tool. These two tools, however, make no effort to validate the recovered data, so false positive header/footer matches result in corrupt file recoveries. Especially in a highly fragmented disk these files contain gibberish at the end, may be incomplete or not viewable in the target program, thus requiring additional manual intervention by the investigator to remove the incorrect data blocks.

Due to the poor performance of header/footer carvers and the lower success rate for file recovery in less than ideal conditions, utilities for searching the dataset for specific file signatures were built up. Investigators looking for specific files (e.g. in cases of intellectual property theft) could load a source file into the carver and then search the dataset for binary patterns that are similar or the same to the source file. Expert users could create custom hexadecimal signatures (e.g. for files types unknown to the carver) to search for and manually stitch data blocks together.

Many of these header/footer carvers are still in use today and have enjoyed wide commercial success. The most popular among them is EnCase by Guidance Software, which is widely used by law enforcement agencies as well as corporations for a wide variety of forensic investigations [5]. EnCase was successfully used by the FBI in investigations of Enron and WorldCom and boasts a robust range of carving targets on a wide variety of platforms including mobile phones and TiVo boxes [6]. Other popular file carvers in this category include Scalpel, Foremost and FTK.

Except for these well-known works based on header and footer recovering, many detailed works focusing on some specific file types have also been proposed, such as carving the RAR file [12], the PDF file [13]. Since RAR file is the most commonly archived file, in [12], Wei et al. designed a carving algorithm based on the information and internal structure of the RAR. They applied mapping functions to locate the header and footer of an RAR file, comparing the size of the file in the RAR file with the distance between the header and footer of the RAR file or the file size to determine whether the file is fragmented. After they applied enumeration to reassemble the two fragments which were extracted, they implemented the CRC of decompressed data stored in the file header to validate the integrity of RAR file which is a good reminder for us to do the file validation after a file is extracted. In [13], Chen et al. aimed at another widely used file type, PDF. Specifically, they introduced an effective validation method for recovered PDF files by inspecting both content characters and internal structure.

All the tools previously discussed are great at the recovery of contiguously stored files, but issues arise, for example, when trying to recover files that have been split into multiple pieces and stored in different locations across on the disk. Header/footer carving suffers from two significant limitations which recent file carving advances have attempted to address. First, it can only recover contiguous files, which means that non-contiguous files cannot be recovered automatically. Unfortunately, it is very often to see that a file of interest to the forensic investigator is fragmented, for example, email archives, such as the PST or “Personal Folders” files for Microsoft Outlook, due to frequent modification and their large sizes. Second, it requires manual intervention often and as a result requires specially trained

investigators to perform secondary analyses. Recent file carving techniques were developed to address these limitations [2]. Although, most of them have been primarily focused on recovering fragmented files, many have also worked on reducing false positives and automating as many processes as possible in order to ensure widespread use of the tools by novice users. Significant strides have been made in recent years in the development of advanced file carvers due, to a large extent to the Digital Forensic Research Workshop (DFRWS) annual digital forensics challenges begun in 2005 [8]. In the 2006 challenge, participants were are provided with a raw binary dataset and challenged to recover a multitude of file types including JPG, ZIP, and Office documents stored either as fragments or in contiguous data blocks. Several prominent file carving algorithms like Bifragment Gap Carving (BGC) were been developed as a direct result of a DFRWS challenge and will be discussed next.

9.1.2 Bifragment Gap Carving (BGC)

Although it can become very difficult to recover deleted files that are fragmented, an interesting approach has been proposed to deal with a specific data recovery scenario where the deleted file is fragmented into two pieces. When a file is fragmented into two pieces, one piece (also known as base fragment) contains the file header and the other (also known as second fragment) file footer (Fig. 9.4). In [9], this was identified as a bi-fragmented file. The new approach is called Bifragment Gap Carving (BGC). BGC was first designed and developed by Garfinkel to solve the DFRWS 2006 file carving challenge. It was the first algorithm successfully tested on real world datasets and takes a vastly different approach to solving the data reassembly problem.

BGC is based on a dedicated survey. The survey from more than 300 hard drives used on the second hand market shows that 97% of files were either contiguous or fragmented into two parts (bi-fragmented) and 50% of recovered fragmented files are bi-fragmented files. No matter whether or not file system structures exist, it is always easy to recover deleted or lost contiguous file or data. In [9], Garfinkel concentrated his efforts on developing an algorithm for bifragmented files (i.e. files fragmented into two pieces), hence the first part of the name of the algorithm. Similar to header/footer carving, it first uses the unique header and footer of the file to find the range of

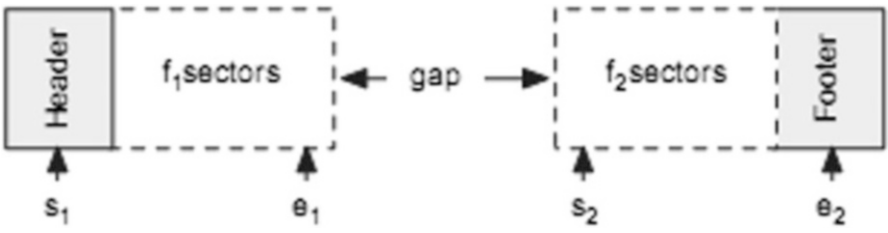


Fig. 9.4 A bi-fragmented file [15]

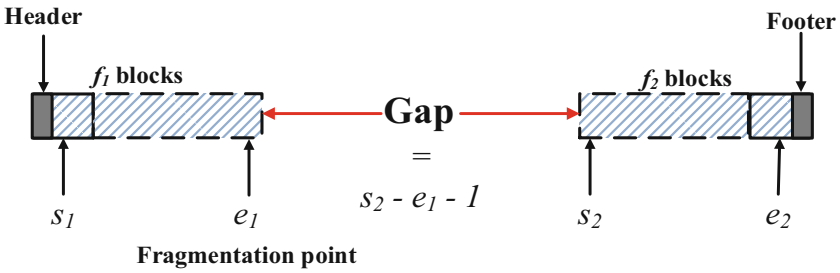


Fig. 9.5 Gap carving problem formulated with two variables gap size g and fragmentation point e_1

the file. "Gap Carving" refers to the algorithm's technique of choosing a gap size, or distance between data blocks then testing all possible combinations of data block sequences using a technique called object validation to see if the candidate sequence of blocks separated by the selected gap size produces a valid reconstructed file [15]. If the validation failed, the gap size is incremented and the test repeated until there are no more data blocks or the validation passes. "Gap Carving" becomes effective because another finding obtained in the survey is that the gap between the first fragment and the second fragment is a relatively small number of disk sectors.

Next, we formulate gap carving problem with two variables, gap size g and fragmentation point e_1, shown in Fig. 9.5. We first identifies where the file header and footer are located. The file header, contained in block s_1, is considered the starting point of the first fragment (base fragment) and the file footer, contained in block e_2, is considered the ending point of the second fragment. A gap, "G", of blocks (or clusters) containing non relevant data must therefore exist between the two pieces of a bi-fragmented file. If the size of a gap and the last cluster of the base fragment (fragmentation point e_1) are fixed, the gap is determined so it can be removed accordingly to have a candidate recovered file. Afterwards, a test can be put in to see whether the reconstructed file is valid, aka object validation. An initial value of G is assigned (1 by default, which means there only exists one block not belonging to the file) and each possible location of gaps with the same size (value) is tested from either side of the file. This value sequentially increases until the correct file sequence is found or the maximum possible gap value, i.e., e_2-s_1-1, is reached and all possible combinations of gap sizes and fragmentation points are tried, which means it fails to carve out the file. The maximum possible gap value appears when both the base fragment containing the file header and the second fragment containing the file footer contain only one data block. For example, as shown in Fig. 9.6 below, a file **F** is fragmented into two pieces (fragments) separated by two data blocks not belonging to the file. The first piece (base fragment) contains four data blocks and the second piece (second fragment) contains two data blocks. Thus, the maximum gap size is 6.

The above problem, which is abstracted below, can be solved by the following brute-force algorithm which check all possible gaps, where a gap can be defined by a fragmentation point e_1 and a gap size g.

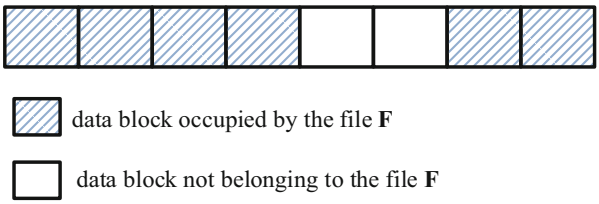


Fig. 9.6 Example of bi-fragmented file

Problem: Given a deleted bi-fragmented file, where base fragment extends from blocks s_1 to e_1 and the second fragment extends from s_2 to e_2, return the recovered deleted file. Note that the block of **address** s_1 containing a file header and another block of **address** e_2 containing the corresponding footer, which are known.

Inputs: Block **addresses** s_1 and e_2

Outputs: Recovered deleted file or failed.

Algorithm: Checking all the possibilities for gap size G and fragmentation point e_1 (Brute Force) for finding the correct gap size and fragmentation point

1. Set initial gap size G_{init} to 1.
2. Calculate the maximum gap size $G_{max}=e_2-s_1-1$
3. For each gap size G from G_{init} to G_{max},
   (a) Remove G blocks at each middle location between two blocks s_1 and e_2, and reassemble the rest of data blocks between the file header and the file footer as a candidate recovered file;
   (b) Verify whether the candidate recovered file is a valid file. If so, return the file as the recovered file.
4. Return "failed".

Generally speaking, BGC is a two-part process—(a) selecting a candidate sequence of blocks; and (b) validating or decoding the sequence to ensure that it follows the structured rules for its file type.

9.1.2.1 Selecting a Candidate Sequence of Blocks

The first run of BGC tries to find all contiguous files, i.e. files that are not fragmented. This is achieved by searching for file headers and file footers in the binary dataset, then merging all the blocks in between and attempting to validate this candidate sequence of blocks. If validation is successful, the file is assumed to be successfully carved and its data blocks are removed from further consideration. The process is repeated until all contiguous files are recovered. Where the validator fails, the file is assumed to be fragmented and the data blocks will be left for more an in-depth reconstruction using gap carving.

BGC attempts to carve files by determining the gap between two fragments, including its size and location. Knowing where a file starts and ends provides a maximum gap and a starting point. In gap carving, an initial gap size of g (e.g., a gap

of 1 block) is chosen, then starting with the base fragments of 1 block i.e. data blocks containing the file headers a candidate sequence of blocks chosen to be combined with the second fragment beginning with a block that is g distance from the last block in the base fragment [15]. In other words, g blocks in the middle location between file header and file footer are considered not part of the file and then eliminated. The left data blocks are then pieced together as a candidate recovered file. The gap moves between file header and file footer through the increase of the size of base fragment to see if two fragments can be reassembled properly. At each stage the candidate sequence of blocks is validated by object validation. If validation is successful, the candidate sequence is saved as the recovered file. If the validation fails, the gap size is incremented and the test is repeated to find the end of base fragment, aka fragmentation point, or the beginning of the second fragment until a successful validation occurs or until the maximum value of gap has been tested.

For example, as shown in Fig. 9.7, a bi-fragment file takes up three clusters and the initial value assigned to G is 1. Step 1 begins with removing the first two clusters right after file's header since we assume the base fragment only has one block. The remaining clusters would be concatenated and tested. When the test result of step 1 is negative, the algorithm moves one cluster forward (or extends the base fragment one block) and repeats its concatenation and testing; this would be step 2. This process repeats until the algorithm reaches the file's footer. At this point the value of G

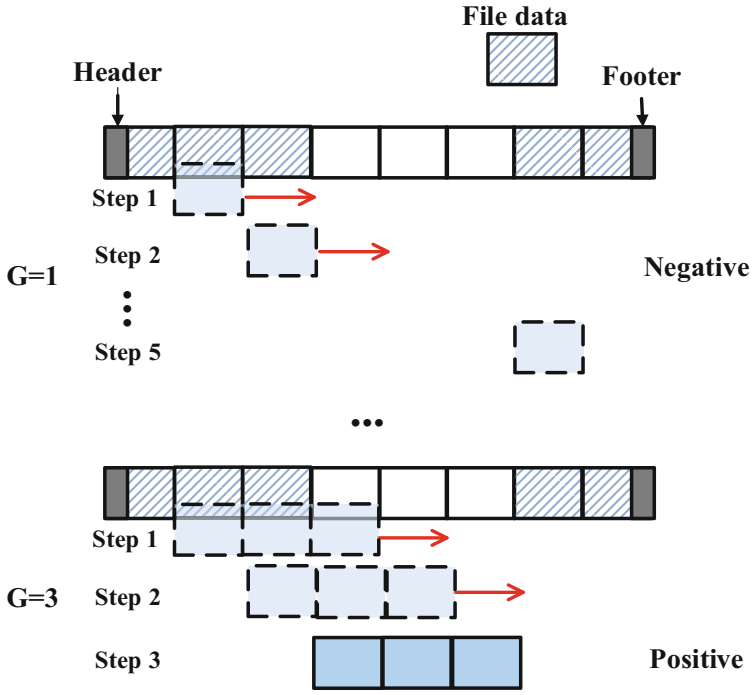


Fig. 9.7 Using gaps to concatenate the file

would be increased and the process would restart removing the appropriate amount of clusters after the file's header. In the example, when G has a value of 3, the third step would result in a positive match, and a proper extract of the file can be made. After the file is recovered, several object validation designs were introduced to validate the carved file in an effort to eliminate false positives.

9.1.2.2 Object Validation

BGC object validation model uses a series of tests to validate a sequence of data blocks to determine if it meets the file structure requirements for its assumed file type. There are many options for validating file or data, individually or in combination increasing accuracy of file validation. Object validation begins by applying simple and fast tests to a candidate sequence of blocks, and only proceeding with more rigorous and slower tests if the preliminary tests are passed [15]. For example, JPEG files all begin with the same series of hexadecimal numbers `FF DE FF` followed by `E0` or `E1` and they all end with `FF D9`. BGC's first pass of JPEG validations therefore can eliminate the majority of candidates that do not meet this requirement and can avoid running the expensive tests on candidates that are of other file types.

The second round of object validation involves the use of container structures for verifying file types. ZIP files and Microsoft Office documents for example are container files with several internal sections with distinct predictable structures that can be validated very quickly. In addition, some container files contain clues about the total size of the file, thus providing additional guidance to the BGC carver. Further, a cyclic redundancy check (CRC) code is used in many file formats for verifying the integrity of the data. CRC is a technique for detecting errors in digital data, as referred to as data integrity. In the CRC method, a certain number of check bits, often called a checksum, are appended to the data or file. Whenever the data or file has been read, we can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in storage or transmission of the entire data. For example, CRC for zip files, Microsoft Office 2007 or 2010 file (.docx, .pptx, .xlsx), etc. Thus, we can use CRC to validate the integrity of recovered data of these file types. The third and final round of object validation entails the use of decompression and semantic validation on the candidate sequence of blocks that pass the first and second round of tests [15]. In addition, manual validation could be performed on the resulted data, for example, viewing a recovered image to see if it is corrupt or not.

BGC has the automatic effect of reducing the number of false positives that plagued **header/footer** carvers because of its object validation that provides continuing validation of file types at the time of reconstruction. The modular design of the validator framework also allows for pluggable validators such that if an improved version of a validator for JPEG files for example is available, it can easily be added into the algorithm without affecting its overall methodology.

The drawback of BGC is its restricted use to files fragmented into at most two pieces. As Pal et al. note in [10], modifying the BGC algorithm for files fragmented into three or more pieces becomes exponentially more difficult. Included in the 3% of files fragmented into more than two pieces are PST, LOG, MDB and other frequently modified files that are very valuable to a forensic investigator, none of which can be recovered by BGC. The false-positive rate of BGC is directly related to the false-positive rate of its validator framework and not every type of file may be decodable, further limiting the use of BGC to recovering only file types that can be successfully validated. In addition, BGC cannot recover files with missing headers or missing data blocks because these files would fail in the validator.

Although this work proposes a promising carving algorithm, it assumes having the correct file header and footer. Obviously, it is no longer feasible if the file doesn't have a unique header/footer structure. In addition, in a realistic scenario, a file may be huge and therefore, it might be time consuming to try all possible gaps. Moreover, all aforementioned file carvers become ineffective when dealing with files that have more than two fragments. The challenge in file carving turns out to be how to recover the file with more than two fragments. Despite many efforts to tackle the problem of file carving, especially carving out those files with more than two fragments, file carving remains an open challenge but of central importance of digital investigation.

9.2 File Carving Tools

There are different carving tools available, many of them open source. Next, we will introduce three popular file carvers.

9.2.1 Foremost

Foremost [14] was written by US Air Force special agents. It is a very popular file carver, and works by first creating a configuration file which contains file header/footer information of certain file format. It then recover files based on these predefined headers and footers. Specially, it first sifts through disk looking for file headers defined in the configuration file. Once a header is found, the search then proceeds to look for its corresponding footer in the configuration file. Figure 9.8 shows an example of file header and footer for JPEG files in Foremost's configuration file. Note that any line that begins with a '#' is considered a comment and ignored. In this example, it means that Foremost is configured to recover JPEG files in a disk image provided.

The next step is to start Foremost to recover JPEG files in DFRWS 2006 Forensics Challenge test image called "dfrws-2006-challenge.raw" [18].

```
# more /usr/local/etc/foremost.conf
......
# GIF and JPG files (very common)
#      (NOTE THESE FORMATS HAVE BUILTIN EXTRACTION FUNCTION)
#      gif     y      155000000      \x47\x49\x46\x38\x37\x61       \x00\x3b
#      gif     y      155000000      \x47\x49\x46\x38\x39\x61       \x00\x00\x3b
       jpg     y      20000000       \xff\xd8\xff\xe0\x00\x10       \xff\xd9
       jpg     y      20000000       \xff\xd8\xff\xe1 \xff\xd9
       jpg     y      20000000       \xff\xd8       \xff\xd9
......
```

Fig. 9.8 Foremost configuration file

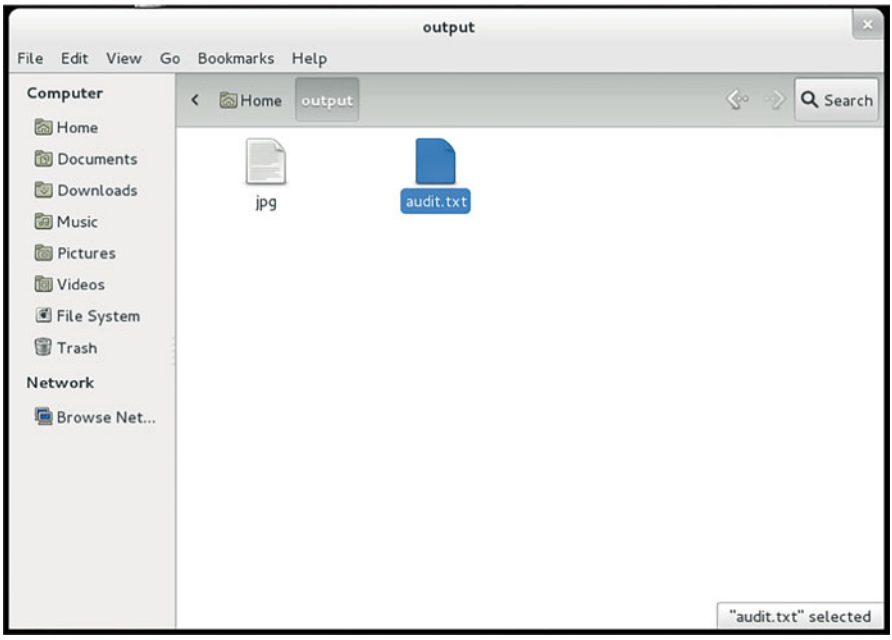


Fig. 9.9 Results of Foremost examining Test Image for the DFRWS 2006 Forensics Challenge

```
foremost dfrws-2006-challenge.raw
```

After Foremost has finished, it will provide a summary what it has done and put it along with recovered files in a subfolder named output in the directory from where you called foremost. Figure 9.9 shows an example output folder when using Foremost to recover JPEG files in data set from DFRWS 2006 Forensics Challenge. The output folder created by Foremost contains two items, the "jpg" subfolder which

contains the recovered jpg files, and the "audit.txt" file which contains a list of all the files recovered.

9.2.2 Scalpel

Despite the successfulness of Foremost, there are many major performance flaws that hamper its functionality. This is caused by the factors that Foremost will repeatedly search through data that has previously been matched, increasing the length of time a few searches can take and memory usage. Scalpel was introduced by Richard and Richard and Roussevet [11] to enhance performance and decrease memory usage. Foremost spent much of its time reading and writing to the hard drive, often the slowest piece in a computer system. The improved file carver, Scalpel [15], first indexes all headers and footers, then looks for potential matches from within that index which is stored in memory; a much faster method than repeatedly searching the hard drive. Additionally, Scalpel also contains improved memory-to-memory copy operations, as well as faster byte writing output. These improvements made Scalpel a much more efficient file carving tool. Nevertheless, Scalpel keeps the same functionalities as Foremost since Scalpel has been derived from Foremost and reads the same configuration file.

9.2.3 TestDisk and Photorec

TestDisk [16] is an open source, multi-platform, data recovery tool developed by the cgsecurity team. It is capable of running on multiple operating systems such as:

- DOS
- Windows XP, 7 and 10
- Linux
- FreeBSD, NetBSD, OpenBSD
- SunOS
- MacOSx.

Unlike file carvers specializing in the recovery of deleted or lost files, TestDisk has many useful features when it comes to data recovery. They are as follows:

- Fix Partition table
- Recover deleted partition
- Recover Fat32 boot sector from backup
- Rebuild Fat12/16/32 boot sector
- Fix Fat Tables
- Rebuild NTFS boot sector
- Recover NTFS boot sector from its backup

- Fix MFT using MFT mirror
- Locate EXT2/3/4 backup superblock
- Undelete files from Fat, exFAT, NTFS and ext2 file systems
- Copy files from deleted FAT, exFAT, NTFS and EXT2/3/4 partitions.

There is another open source application that is offered by cgsecurity, which focuses only on file recovery. The application is called PhotoRec [17], which essentially ignores the file system and goes directly for the data. This application is quite similar to TestDisk since it is also used for data recovery. The main difference between the two is that PhotoRec is strictly only for files while TestDisk offers many other options in which the users can choose from. Also, PhotoRec stands for Photo Recovery, and was originally designed to recover lost pictures or lost files from digital camera memory. PhotoRec is superior when recovering deleted or lost photos and pictures. Actually, PhotoRec is a companion program to TestDisk, and provides file recovery functionality for TestDisk. However, TestDisk supports more data recovery functions, such as deleted or lost partition recovery.

Next, let's go through the TestDisk application and see how it works. Let's consider a scenario where you have to recover data from a USB drive where you accidentally deleted the partition. The USB drive originally had one partition formatted with NTFS file system, shown below (Fig. 9.10).

First, make an image of USB drive using *dcfldd*.

In order to create an image of a USB flash drive, you need to find out the device name for the USB drive you inserted into your Forensics workstation, which is a Linux VM. In Linux, Linux disks and partition names have their own basic naming scheme, for example, the master disk on IDE primary controller is named /dev/hda and the slave disk on IDE primary controller is named /dev/hdb; and the first SCSI disk (SCSI ID address-wise) is named /dev/sda.

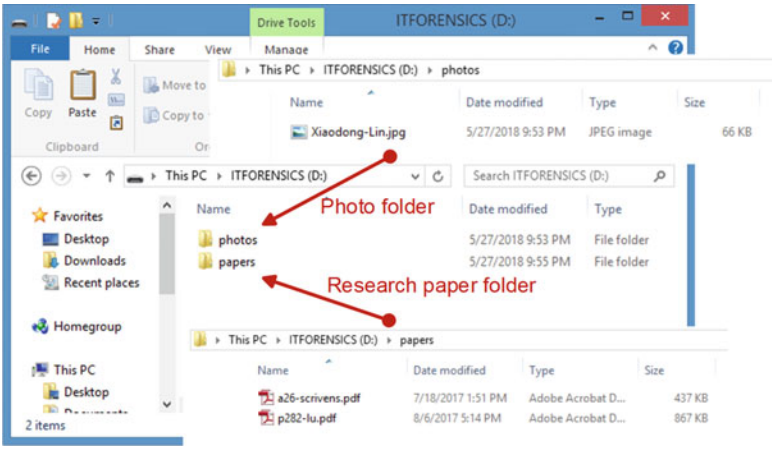


Fig. 9.10 Test USB drive with one partition formatted as NTFS file system, which contains two folders, one photos folder and another research papers folder

Also, after you plug in your USB drive to your USB port, your USB drive is connected to your host computer first instead of your Forensics workstation. For VMWare, each removable device can be connected either to the host or to one virtual machine. Therefore, you have to go to VM > Removable Device and choose your device and then connect the device to your virtual machine. Afterwards, your virtual machine will add new block device into /dev/ directory. To find out what name your USB block device file have you can run fdisk command:

```
fdisk -l
```

In this example, shown in Fig. 9.11, the USB drive that will be imaged is located at /dev/sdb, and the logical volume that the image will be saved on is /dev/mapper/vg-lv_root.

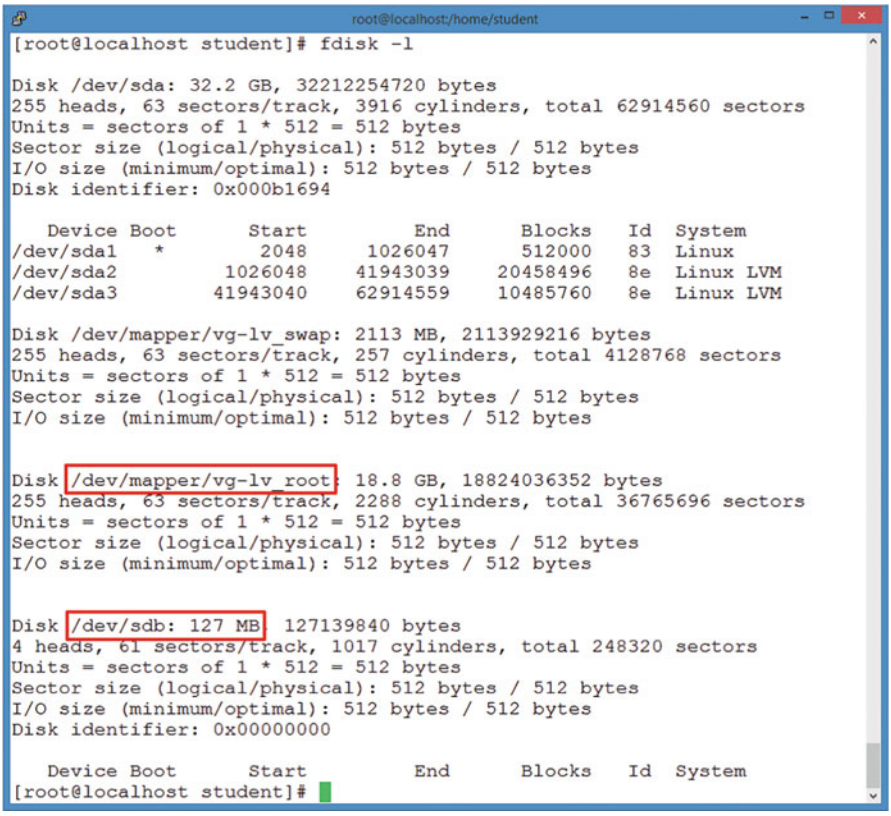


Fig. 9.11 List of disks/partitions of forensics workstation after test USB drive is inserted

Note that when imaging the USB drive, every precaution must be used to prevent data on USB drive from being corrupted or overwritten, for example, using use write blocker.

The next step is to use the dcfldd utility to create an image copy of the USB drive.

```
dcfldd if=/dev/sdb of=/home/student/datatraveller.img hash=md5 hashlog=/
home/student/hashlog.txt
```

where /dev/sdb is the device name of the USB drive to be imaged, "datatraveller.img" is the name of the file used to store the USB drive image and located within a folder of /home/student. Also, the hash option specifies what kind of cryptographic hash function(s) will be applied to the acquired data, and in our example, the hash function used is MD5. The hashlog option specifies where the output of the hashing should be stored; in our example, it will be saved into a text file in the same directory as the disk image.

After imaging is done, you can check the md5 hash value by looking at the file "hashlog.txt".

```
cat/home/student/hashlog.txt
Total (md5): 91c896ec3c836bf82424fdf7e8134332
```

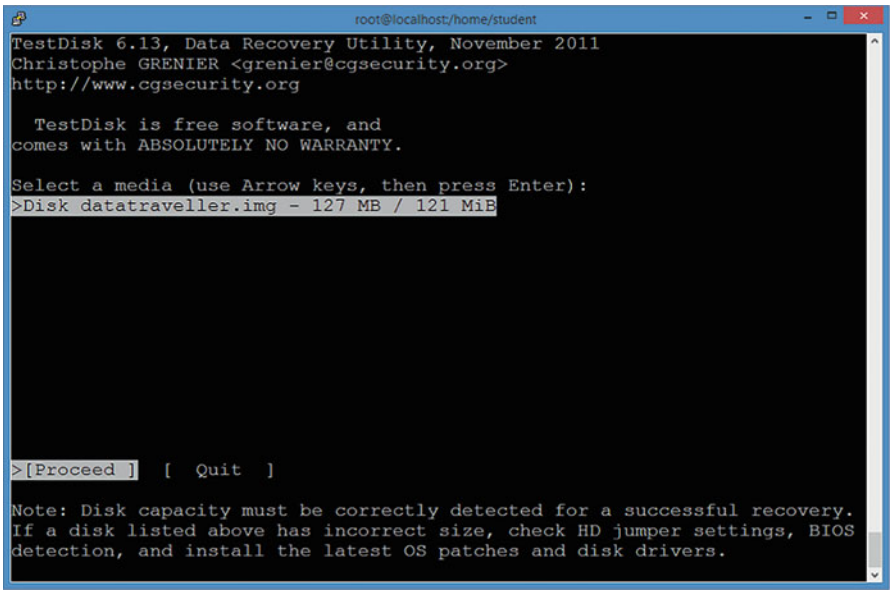
It can be verified by computing the MD5 hash value of the USB drive image using the following command

```
md5sum datatraveller.img
91c896ec3c836bf82424fdf7e8134332 datatraveller.img
```

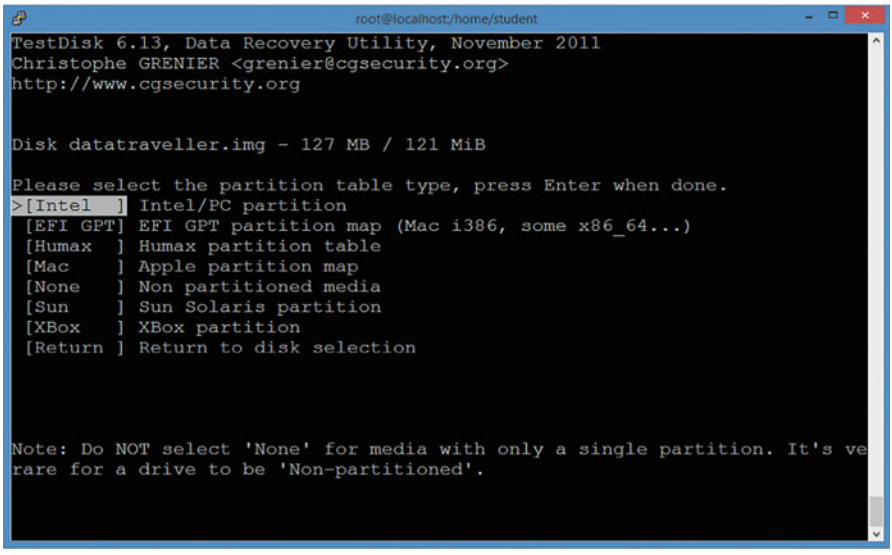
It can be evident that the hash value in the file hashlog.txt is the same as the one calculated above.

Second, launch TestDisk by loading the USB drive image acquired above using the following command.

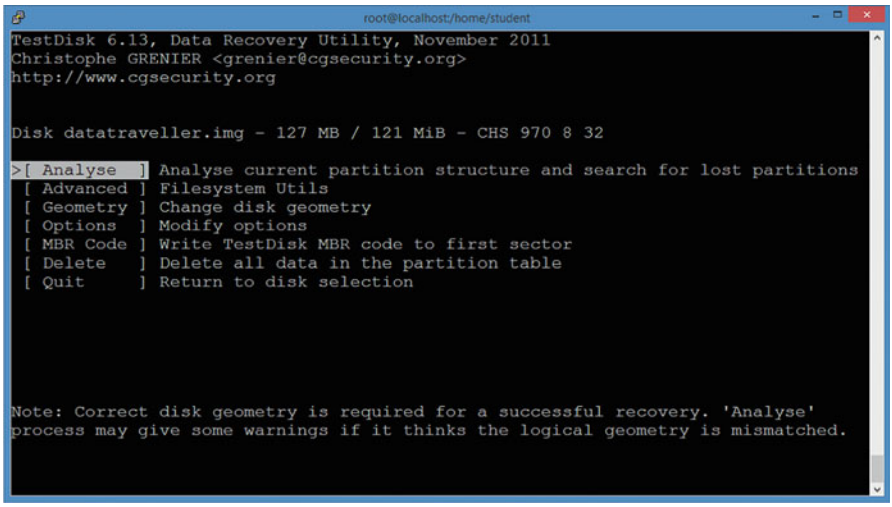
```
testdisk datatraveller.img
```



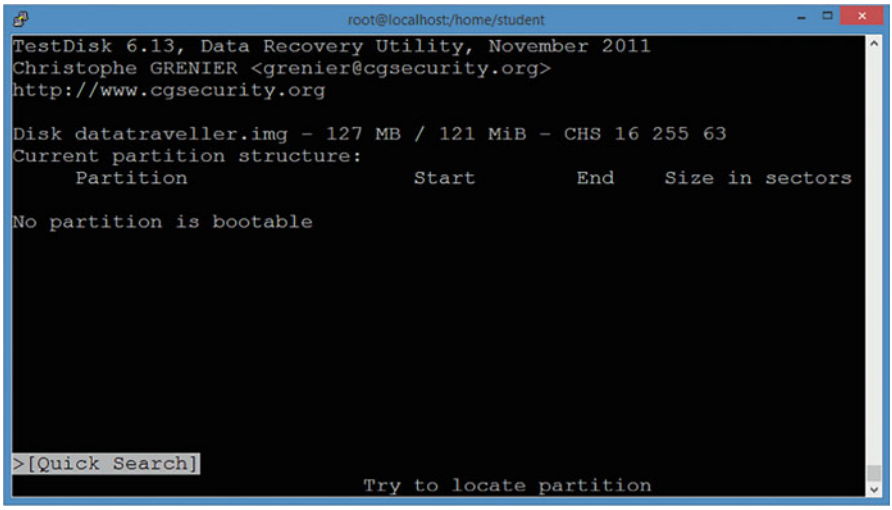
- Select the "Proceed" option and Press ENTER to continue.



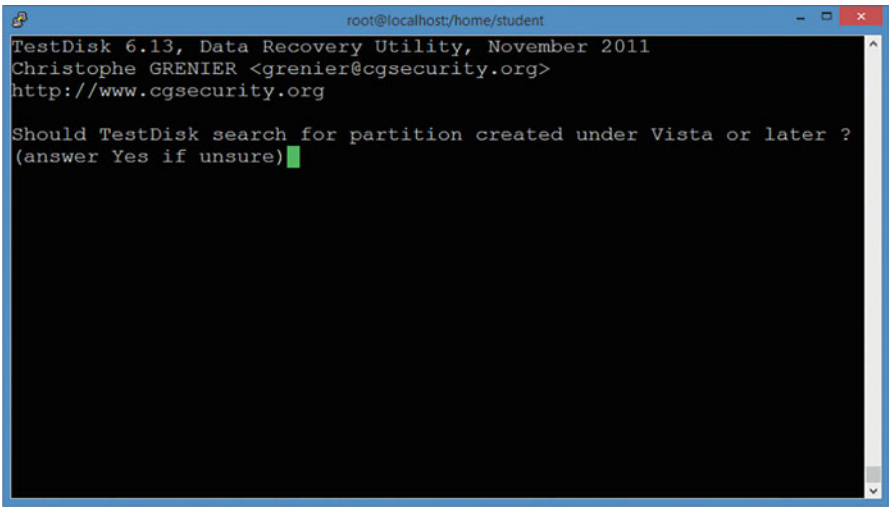
- Select the partition table type to continue. The option chosen in this example is Intel/PC partition. Note that it is very important to choose the correct partition type. If the user selects the wrong type then the data recovery process will most likely fail.



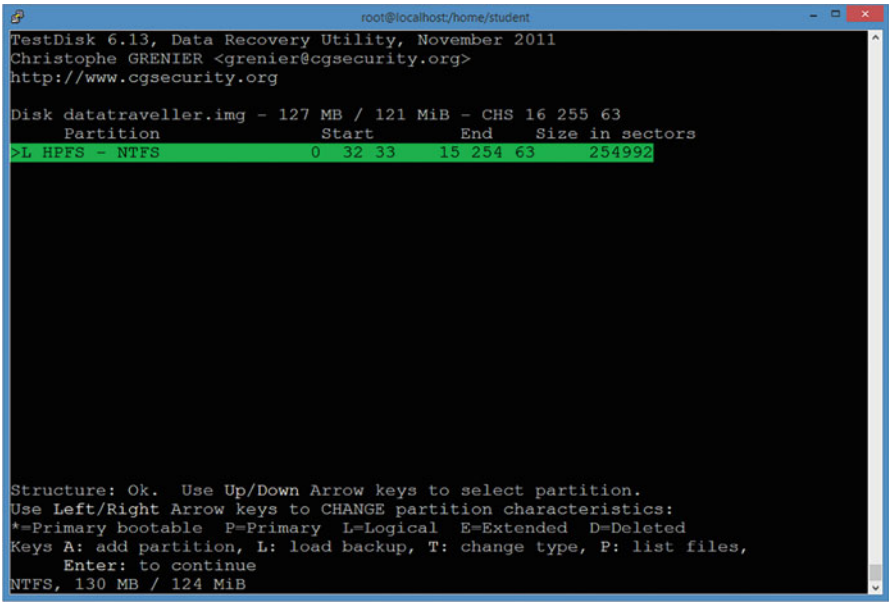
- This is the main menu of TestDisk, which provides the user with the various options. Select the "Analyze" option. It allows for the analysis of the current partition structure and search for lost partitions, which is the objective of this example.



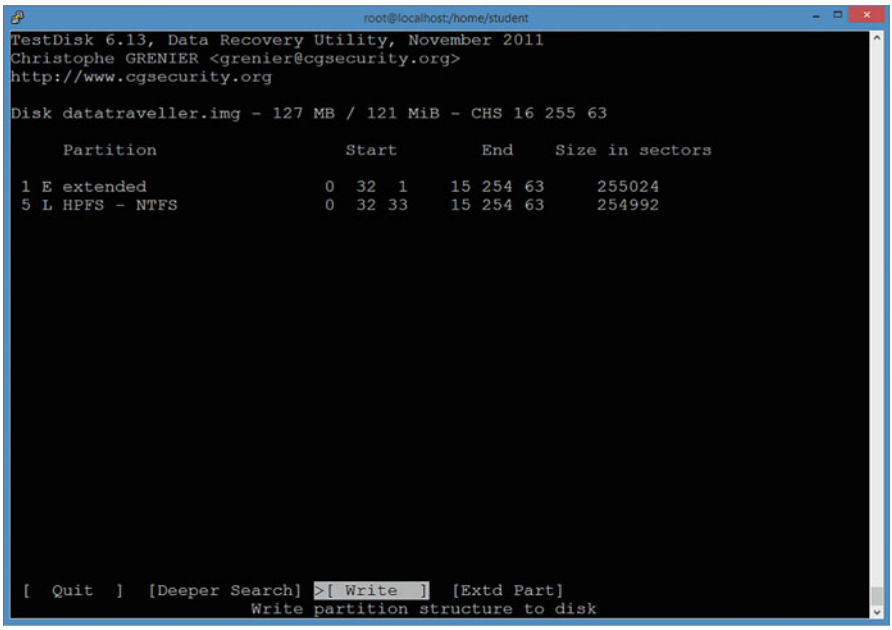
- No partitions are showing. Select the "Quick Search" option and press ENETR to continue.



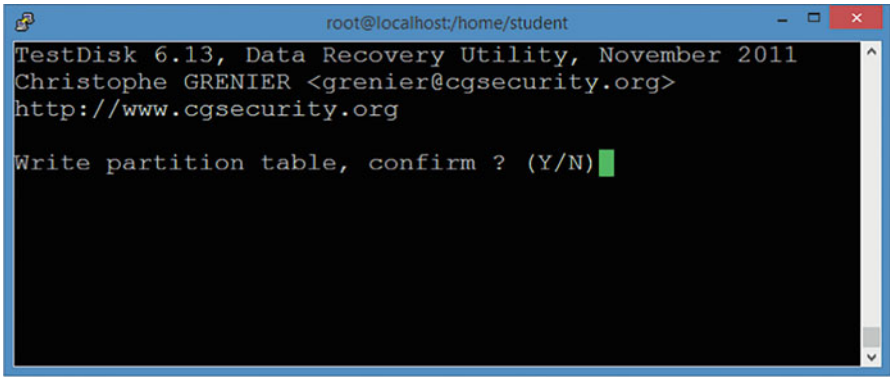
- Confirm whether TestDisk search for the partition created by Vista or later. Press Y if yes/not sure. N otherwise.



- After the scan is over, all the partitions found by “Quick Search” will be displayed. Also, if any detected partition is not corrupted, it will be displayed in green. It can be observed that the deleted partition is detected by TestDisk. Next, we will restore deleted partition and press ENETR to continue.



- Select the "Write" option, and Press ENTER to continue.



- Confirm write partition table to the disk image by pressing **Y**

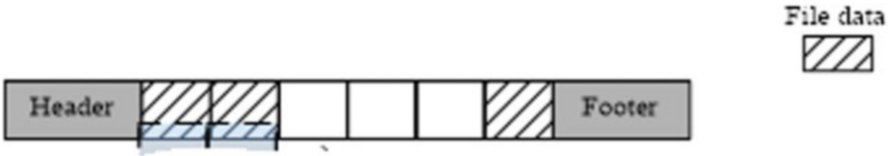
Now, we have restored the deleted partition in the USB driver image. Finally, restore the image with fixed partition onto USB drive.

```
dcfldd if=/home/student/datatraveller.img of=/dev/sdb
```

Once it is done, the deleted partition is restored and you are able to access all your files on your USB drive again.

Review Questions

1. What is file carving?
2. What is Header/footer carving?
3. What is the main difference between file carving and file recovery techniques using file system information about files and directories?
4. Bifragment Gap Carving (BGC) is a promising algorithm for effectively carving out a file from unallocated disk spaces. Explain how BGC works and discuss the limitations and drawbacks of BGC. If necessary, give an example and/or a diagram to help in your explanation.
5. Consider a deleted bi-fragmented file containing the following data blocks:



   (a) If you use BGC to recover the above file with the initial gap size of 2, what is the number of tries needed to successfully recover the deleted file?
   (b) Suppose that the second block of the file is damaged or overwritten. As a result, BGC has to try all possible gaps before it exists but fails to recover the file, since file validation should never succeed. What is the total number of tries before BGC exits? Assume the initial gap size is 1.

6. What is object validation? Give one example of object validation techniques and explain how it works.

9.3 Practical Exercise

The objective of this exercise is practise file carving skills on some publically available dataset.

9.3.1 Setting Up Practical Exercise Environment

1. Download a compressed Digital Forensics Tool Testing Image [19], "11-carve-fat.zip", and extract it for a "raw" partition image of a FAT32 file system, "11-carve-fat.dd", and upload it to Forensics Workstation. To download it, go to http://dftt.sourceforge.net/test11/index.html.

2. Download and Install Scalpel on Forensics Workstation. To download Scalpel, go to https://github.com/sleuthkit/scalpel. Please see Appendix A in Chap. 3 for detailed instructions on how to install software in Linux.

After properly installing Scalpel and testing image, you can proceed to complete the following exercises.

9.3.2 Exercises

Part A—Evidence Hashing

Use md5sum to calculate the hash value of the raw partition image of a FAT32 file system ("11-carve-fat.dd") and answer the following question:

Q1. What is the MD5 hash value of the raw partition image used in the exercise? _____

Part B—Data Carving with Scalpel

Configure Scalpel to carve out PDF and JPG graphic files from the image ("11-carve-fat.dd") and answer the following questions:

Q2. How many PDF files are recovered by Scalpel? _____

Q3. How many JPG files are recovered by Scalpel? _____

Q4. Is a file with a MD5 hash of "c0de7a481fddfa03b764fa4663dc6826" one of recovered JPG files? _____ (Yes/No)

Q5. Is a file with a MD5 hash of "80dc29617978b0741fa2ad3e452a6f9d" one of recovered PDF files? _____ (Yes/No)

Q6. There is a PDF file in the image ("11-carve-fat.dd") called "lin_1.2.pdf" whose md5 hash value is "e026ec863410725ba1f5765a1874800d" in hex. What is the size of the file "lin_1.2.pdf"?

References

1. BMP file format. https://en.wikipedia.org/wiki/BMP_file_format
2. A. Pal and N. Memon. The evolution of file carving. IEEE Signal Processing Magazine, vol. 26, no. 2, pp. 59-71, March 2009
3. Z. Lei. Forensic analysis of unallocated space. Master's Thesis, University of Ontario Institute of Technology, 2011
4. C. Beek. Introduction to File Carving. McAfee White Paper. Available at: http://www.mcafee.com/ca/resources/white-papers/foundstone/wp-intro-to-file-carving.pdf
5. Guidance Software. Investigations of individuals. Available: http://www.guidancesoftware.com/computer-forensics-ediscovery-individual-investigation.htm
6. X. Lin, C. Zhang, T. Dule. On Achieving Encrypted File Recovery. In: X. Lai, D. Gu, B. Jin, Y. Wang, H. Li (eds) Forensics in Telecommunications, Information, and Multimedia. e-Forensics 2010. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 56. Springer, Berlin, Heidelberg
7. File carving - forensics wiki. Available: http://www.forensicswiki.org/wiki/File_Carving
8. Digital Forensics Research Workshop 2005. http://old.dfrws.org/2005/index.shtml

9. S. L. Garfinkel. Carving contiguous and fragmented files with fast object validation. Digital Investigation, vol. 4, pp. 2-12, 2007
10. A. Pal, H. T. Sencar and N. Memon. Detecting file fragmentation point using sequential hypothesis testing. Digital Investigation, vol. 5, pp. S2-S13, 2008
11. G. Richard and V. Roussev. Scalpel: A Frugal, High Performance File Carver. DFRWS 2005 USA
12. Y. Wei, N. Zheng, and M. Xu. "An Automatic Carving Method for RAR File Based on Content and Structure", Second International Conference on Information Technology and Computer Science, 2010, pp. 68-72
13. M. Chen, N. Zheng, X. Xu, Y.J. Lou, and X. Wang. "Validation Algorithms Based on Content Characters and Internal Structure: The PDF File Carving Method", International Symposium on Information Science and Engineering, Dec. 2008, vol. 1, pp. 168-172
14. Foremost (http://foremost.sourceforge.net/)
15. Scalpel. https://github.com/sleuthkit/scalpel
16. "TestDisk - Partition Recovery and File Undelete", Cgsecurity.org, 2017. [Online]. Available: http://www.cgsecurity.org/wiki/TestDisk. [Accessed: 06- Apr- 2017]
17. "PhotoRec - Digital Picture and File Recovery", Cgsecurity.org, 2017. [Online]. Available: http://www.cgsecurity.org/wiki/PhotoRec#Operating_systems. [Accessed: 06- Apr- 2017]
18. DFRWS 2006 Challenge: http://old.dfrws.org/2006/challenge/submission.shtml
19. Digital Forensics Tool Testing Images. http://dftt.sourceforge.net/