

Automated Generation of a Natural Challenge File for File Carving Algorithms

^[1] K. Srinivas, ^[2] Dr. T. Venugopal

^[1] Research Scholar (External) in CSE JNTUH University Hyderabad, Telangana, India

^[2] Professor, Department of CSE, JNTUH College of Engineering, Sultanpur Sangareddy Dist. Telangana, India

Abstract:- File Carving is a technique of reassembling unordered mixed file fragments, without using files' metadata such as FAT, for reconstructing the actual files present on the disk. In the areas of data recovery and digital forensics this situation arises. A challenge file consists of number of files, in the form of fragments, mixed in random order. In this paper authors have presented a software system that generates a challenge file by implementing, at user level, a file system which broadly follows FAT file system. This software system uses a large size file to store file fragments just like a kernel level file system uses disk to store files. The kernel level file system fragments the file, as per the availability of free clusters, at the time of creation of the file. By viewing the challenge file as a virtual disk, it consists of the number of virtual clusters. The software system presented in this paper, a user level file system, fragments the file, as per the availability of free clusters, on the virtual disk i.e., the challenge file. This challenge file consists of mixed file fragments of number of user files. The content of the challenge file is a result of software module which broadly follows FAT file system. The challenge file thus generated is, therefore a natural challenge file. This challenge file provides the writers of file carving algorithms a platform to test their algorithms. The designers of file carvers can use the challenge file conveniently as a virtual disk, in place of the actual disk, thus eliminating the need of physical hard disk for testing their algorithms. There are number of other benefits of this approach as outlined in this paper

Index Terms: File carving; file system; challenge file; digital forensics; data recovery

I. INTRODUCTION

When a user saves a file on a disk, the Operating System uses its File System component to handle it. A File System is a set of software modules, at kernel level, for file handling operations. Assume that a user has created a file named as "one.txt" containing the text "abc". The size of this file is 3 bytes. The file system allocates one free clusters for this new file, at the time of its creation, from the pool of free clusters that it maintains. A cluster is a set of consecutive sectors on the disk. In this paper authors use the term cluster to mean consecutive 4096 number of bytes each cluster starting at byte offset in a challenge file, equals to, a multiple of 4096. A cluster is an allocation unit. The kernel file system views the disk as a set of clusters than as a set of bytes. When a new file is created by a user, the required number of free clusters is allocated for it. And when a file is deleted all the used clusters by the file are freed. So, for the above "one.txt" file, one cluster is allocated. Thus when the properties of the file "one.txt" are viewed on Windows 7 Operating System, we notice file size as 3 bytes and size on disk as 4096 bytes.

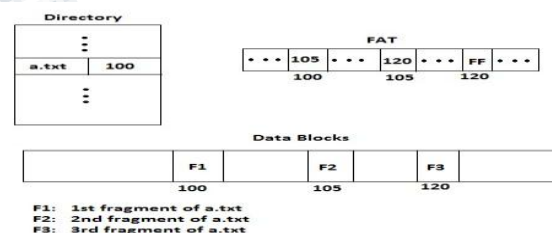


Figure 1. File fragments and entries in directory and FAT

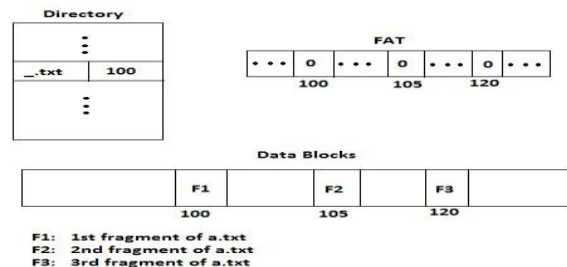


Fig. 2. File fragments and modified entries in directory and FAT after deleting a.txt file

Consider a file named "a.txt" of size 10KB on the disk saved at cluster numbers 100, 105 and 120. To perform read operation on this file, the file system obtains these clusters numbers (that were saved in file system's data structures when the file was created, as shown in Fig 1), reads data from these clusters and presents the data to user application. It is up to user application how to interpret this data. When the file is deleted, the File System changes the first byte of file name of a.txt to '_'. Then it stores 0 in each of the FAT locations at indexes 100, 105 and 120 to indicate that the clusters 100, 105 and 120 are free now. The actual data of a file a.txt is not erased. It is illustrated in Fig 2. If the files were deleted accidentally they need to be recovered. If the File System data structures 'directory' and/or 'FAT' is/are corrupted and files' data is available on the disk then File System cannot present the files' data to the user. Important files may need to be recovered from a state of a disk in which file's content is available but not its metadata. Criminals using computers for their criminal activities may also intentionally delete or format the disk on knowing the raid by investigating agencies. In this case also files' data is available on the disk but its metadata is not available. The investigating agencies, on obtaining such disks, need to find the files that were created or used by criminals. To face the above situations technically in the areas of data recovery and digital forensics, a new technology known as file carving has evolved. File carving is a technique of reassembling mixed file fragments, in the absence of files' metadata, to reconstruct the actual files present on the disk. In conventional method of reading a file, the File System refers to file system's data structures, then reads the data present in the clusters and presents it to the application at user level. The situations described in the above paragraphs are not suitable for accessing files using a conventional method. File carving is an unconventional method of accessing files from the disk when the files' metadata is missing. A file carving algorithm, during its development phase, needs to test a used disk partition of small size to verify the correctness of the developed algorithm. As a replacement for the test disk, in the recent research, a large file containing mixed file fragments but not containing the metadata is used to verify the correctness of the developed file carving algorithms. When a used disk partition is considered, it contains naturally file fragments because the file system always cannot allocate clusters in contiguous area on the disk for a file. The same state is created artificially on a large file and is used as an alternative to a test disk. This large file therefore contains unordered mixed file fragments of number of files without any metadata of files. It is a challenge for a file carving

algorithm to join these pieces for reconstructing the actual files present in it and present to the user applications. Therefore, a large file containing unordered, mixed file fragments of number of files without any files' metadata is called a challenge file. The authors are not aware of availability of any tool to construct a challenge file that is as natural as a used disk partition. In this paper, authors present a software system that implements an automated construction of a challenge file that is as natural as a used disk partition. The presentation of our work is planned, in this paper, as follows. In section II, structure of a challenge file is described. In section III, the principles adopted for creating a challenge file are described. In section IV, design and implementation of software system is presented. In section V, results of our experiments are presented.

II. STRUCTURE OF A CHALLENGE FILE

The automated construction of a natural challenge file consists of the THREE phases. 1). An initial phases 2) Construction phase 3) Fine-tuning phase. In this section, the structure of a challenge file during each of the above phases is described.

A. Structure Of a Challenge File in Initial Phase

In an initial phase, the challenge file is viewed as consisting of contiguous clusters with each cluster containing all zeros in it. A cluster is a set of 4096 contiguous bytes starting at a byte offset satisfying the equation (1). The above state of a challenge file is equivalent to a disk containing no files on it and files system data structures in resonance with the same. The structure of a challenge file, thus is as shown in Figure 3.

$$\text{Byte-offset \% } 4096 = 0 \quad (1)$$



Figure 3. Structure of a challenge file in initial phase

B. Structure of a Challenge File in Construction Phase

During construction phase, the challenge file consists of three regions as shown in Figure 4. The three regions are a) DIR b) FAT c) Data Clusters as described in section I. The state of a challenge file is equivalent to a disk containing many files of which three files File-A, File-B and File-C such that the File-A and File-B (fragmented files) and File-C (contiguous file) they required 4 clusters, 3 clusters and 2 clusters respectively. The three files together require 9 clusters. Therefore nine corresponding locations of FAT store non-zero values. All FAT locations corresponding to free clusters store zeroes. The three entries in DIR are allocated one for each of the three files File-A, File-B and

File-C. The total number of allocated entries in DIR region is equal to the number of files created on the disk.

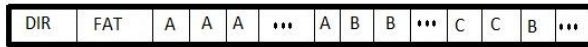


Figure 4. Structure of a challenge file during construction phase

C. Structure of a Challenge File in Fine-tuning Phase

The structure of a challenge file after fine-tuning phase is shown in Figure 5. It is equivalent to a 'used disk' containing data of files created during the construction phase but not containing metadata of all the files created during the construction phase. Therefore, in Figure 5 data of File-A, File-B and File-C is present but does not contain their metadata.



Figure 5. Structure of a challenge file after fine-tuning phase

The above challenge file is a challenge for a file carving algorithm. File carving algorithm, during its development phase, can be verified about its correctness in joining pieces for retrieving File-A, File-B, File-C and others. Submitting challenge file to file carver developer is equivalent to submitting a disk to a digital forensic expert or a data recovery expert. The challenge file acts as a used disk for file carving algorithms and therefore we use virtual disk as a synonym for the term challenge file.

III. THE TECHNIQUE OF CONSTRUCTION OF A CHALLENGE FILE

A. The Technique of Construction in Initial Phase

A virtual disk (i.e. a challenge file) of user specified size S GB is created in binary mode and initialized to contain all zeroes. Depending upon the size of the virtual disk, the size DIR region and the size of FAT region are calculated. The various parameters for the virtual disk are specified in the Table 1. The cluster size is decided as 4KB as this is the common size used in Operating Systems. Each directory entry contains two fields namely filename and starting cluster. The directory entry size is 16 bytes because the filename maximum size is decided as 14 bytes and 2 bytes for starting cluster number. Each cluster number is represented by a 2byte unsigned number and hence the maximum number of clusters supported is 216 clusters.

TABLE 1. THE SPECIFICATIONS OF VIRTUAL DISK

Cluster size	4KB
Each FAT entry size	2 bytes
Maximum clusters	2 ¹⁶ clusters
Max Virtual Disk Size	0.25 GB
Each DIR entry size	16 bytes (14+2)
Max no of DIR entries/ cluster	256 entries
Max no. of FAT entries/cluster	2K entries
minimum file size on disk	4KB

B. The Technique of Construction in Construction Phase

In this phase, the user performs a sequence of command operations of his choice. Each command is selected from the following list a) Create a new file b) Modify an existing file c) Delete a file. The result of the sequence of operations is that the user files are saved on the virtual disk (i.e., the challenge file) in fragmented/contiguous areas depending upon the availability of free clusters for each file just like the kernel level file system saves files on a real disk. Therefore natural fragmentation is achieved. The above operation is automated by executing a script file containing text that represents a sequence of command operations of user's choice. Therefore automated generation of a natural challenge file is achieved. The challenge file contains metadata during this phase.

C. The Technique of Construction in Fine-tuning Phase

In this phase, the DIR and FAT regions of virtual disk are erased so that the virtual disk is a real challenge for the file carving algorithm. After the three phases of construction are over the result is a challenge file. It contains data of user files in data clusters but does not contain any metadata. This challenge file is generated automatically by executing a script file. The files are naturally fragmented just like a file system fragments files on the disk because the similar file system is used but in user space.

IV. DESIGN AND IMPLEMENTATION

This software system consists of the following four classes. 1) DIR class 2) FAT class 3) FileSystem Class 4) UserSpace Class. In this section, the responsibilities assigned to these classes are described in detail.

A. The DIR Class

DIR class is assigned with the following responsibilities. 1) Write a Directory entry 2) Obtain a free directory entry 3) Obtain a start cluster of a given file 4) Make free an existing directory entry. Each directory entry consists of two fields. They are filename and starting cluster number. Maximum size of a filename is fixed as 14 characters. Each cluster number is represented by 2 bytes. So each directory size is

16 bytes. The member function is supposed to prepare the record containing these two fields and should write this record to the first free entry in the directory table. A slot in a directory table is said to be free if its starting cluster number is zero or filename starting byte is '_'. This operation needs to be performed when a new file is created on a virtual disk. To read a file present on the virtual disk, in conventional method, we need to know the fat chain of a file. The fat chain starting cluster is present in directory region and the actual chain is present in fat region. The end of the fat chain is marked as a number 0xFFFF. The initial cluster number is obtained from the file's corresponding entry in directory region. When a file is deleted, the corresponding entry in the directory region must be made free. It is marked as a free entry by changing the first byte of filename to '_'. When the directory region is initialized the whole of it is written with all zeroes. It is not required to make the starting cluster as zero for an entry corresponding to the file being deleted. In kernel level file system, each entry in a directory table contains other attributes like file size, time of file creation etc. From file carving point of view, it is required to introduce fragmentation naturally and therefore it does not require storing all the attributes of a file in directory entry. So the fields in a directory are restricted to filename and starting cluster number in user space file system.

B. The FAT Class

The FAT class is assigned with the following responsibilities. 1) Get the required number of free clusters 2) Given a set of cluster numbers, form a fat chain 3) Obtain the fat chain given a starting cluster number. The FAT region is basically an array, each element occupying 2 bytes. When the FFAT entry is zero, then the corresponding cluster is a free cluster. When a new file is created or when an existing file is extended, we need a certain number of free clusters on the virtual disk so that the new file data or extended file data is written to these clusters. After writing data to the free clusters, a fat chain needs to be formed. If a file's data is written to cluster numbers 10, 105 and 120 then at 10th location the element 100 is written. At 100th location the element 105 is written. At 105th location, the element 120 is written. At 120th location, the element 0xFFFF is written. The value 0xFFFF marks the last cluster in the chain. Suppose that a file is stored at cluster numbers 10, 105 and 120. When a file is to be read, starting cluster number is obtained from a directory region. The value 10 is obtained from the corresponding entry in a directory region. Then in the fat region, the element at 10th location is read. Its value is 105. Then the element at 105th location is read. Its value

is 120. Then the element at location 120 is read. The value at this location is 0xFFFF. So the fat chain is 10->105->120.

C. The FileSystem Class

The FileSystem class is assigned with the following responsibilities. 1) Read the cluster data given a cluster number 2) write data to the specified cluster. Irrespective of file type, the file system always views the file content as a sequence of bytes. To read a file, a starting cluster is obtained from a directory entry. Then fat chain is obtained from the fat region. Then one-by-one cluster data is read to present it to the user application. So it is required to have a facility in User Space File System (USFS) class that reads the cluster data given a cluster number. When a new file is created, it obtains a free directory entry, finds the number of required clusters, forms fat chain and then writes data to the data clusters. Therefore, in File System class it is required to have one facility for writing the file's bytes to the specified cluster.

D. The UserSpace Class

The UserSpace class is assigned mainly with the following responsibilities. 1) Create a file 2) Modify a file 3) Delete a file. These operations are implemented using the facilities provided by the above three classes.

The programming language C++ is used to implement the above classes. The four C++ classes are given below.

```
class DIR
{
public:
int write_dir_entry(char *fn, unsigned int sc);
unsigned int get_free_entry();
unsigned int get_start_cluster(char *fn);
int make_dir_entry_free(char *fn);
};
```

```
class FAT
{
public:
ui get_free_cl();
ui get_next_cl(ui clno);
ui get_next_cls(ui clno, ui cls[]);
void set_next_cl(ui clno, ui nextcl);
ui FAT::operator [] (ui clno);
int get_free_cls(ui cls[], ui n);
void set_next_cls(ui cls[], ui n);
void free_next_cls(ui cls[], ui clcnt);
};
```

```
class FileSystem
```

```
{
public:
void read_cl(ui clno , uc b[CLSIZE] , ui bytecnt) ;
void write_cl(ui clno , uc b[CLSIZE] , ui bytecnt) ;
} ;
```

```
class UserSpace
{
DIR dir ;
FAT fat ;
FileSystem fs ;
char command[60] ;
char cmd[20], char arg1[20], char arg2[20] ;
public:
void showfile() ;
void format() ;
void hdump(ui low=0 , ui high=240) ;
void write(char fn[]) ;
void directory() ;
ui del() ;
void modify() ;
UserSpace() ;
void create() ;
void help() ;
} ;
```

Implementation of the two important operations of FileSystem class is explained below. The read_cl() member function reads a specified cluster from a challenge file into a buffer b. Using fseek() the file pointer is made to point to the starting byte of the specified cluster. fread() is used to read the cluster of size 4KB into buffer b. The write_cl() member function writes a buffer 'bytes' to a specified cluster on virtual disk. Using fseek() the file pointer is made to point to the starting byte of the specified cluster. fwrite() statement writes the buffer of size 4KB to the virtual disk. In the above two member functions and all other applicable member functions, "test.dat" is a virtual disk and acts as a challenge file after completing the three phases of construction.

```
void FileSystem :: read_cl(ui clno , uc b[] , ui bytecnt)
{
FILE *fp = fopen("test.dat" , "rb+") ;
fseek(fp , CLSIZE * (long)clno , 0) ;
fread(b , 1 , bytecnt , fp) ;
fclose(fp) ;
}
```

The program utilizing the above classes is compiled and executed using Turbo C++ compiler and the results of the experiments are presented in the next section.

V. EXPERIMENTS AND RESULTS

The various operations supported by User Space File System (USFS) software for the user are shown in the screenshot in Figure 6. In the screenshot, vd> is a prompt for the user. The prompt vd is a short form for virtual disk. "?" is a command to display a list of commands that are available for the user to execute.

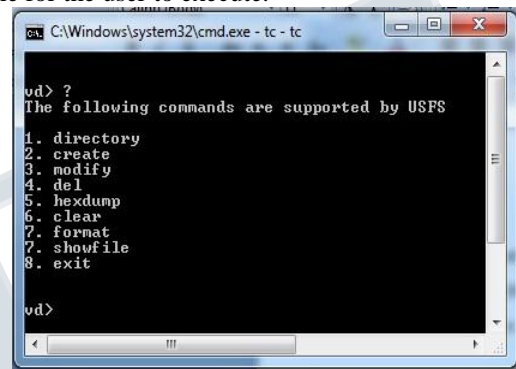


Figure 6. Response of "?" command

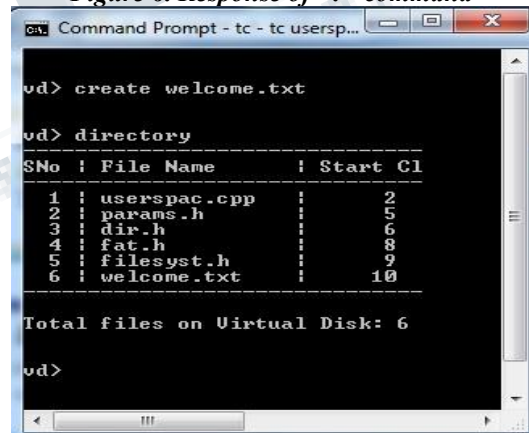


Figure 7. Response of "directory" command

When "del" command is executed, it can be seen that the total files is 5. And it can also be seen that the deleted filename's starting byte is changed to "_". This is the procedure adopted in FAT file system. In Figure 9, screenshot of "hexdump" command is shown. It contains the cluster number at the top followed by a list of sub commands and followed by a table showing the content of challenge file. The table has 3 columns. The first column gives offset range. The second column displays content in

the specified offset range, in hex formats. In third column, same content in the form of text is displayed

```

C:\> del params.h

C:\> directory

SNo | File Name | Start Cl
-----|-----|-----
1 | userspac.cpp | 2
2 | _arams.h | 5
3 | dir.h | 6
4 | fat.h | 8
5 | filesyst.h | 9
6 | welcome.txt | 10

Total files on Virtual Disk: 5

C:\>

```

Figure 8. Response of “del” command

```

C:\> hexdump

Cluster Number : 0
<d:DIR f:FAT n:nextPage p:prevPage N:nextCl P:prevCl x:Exit>
LOW:HIGH | H E X | B Y T E S | Characters
-----|-----|-----|-----
0:15 | 175 73 65 72 73 70 61 63 2e 63 70 70 0 0 2 0 | userspac.cpp 0
16:31 | 170 61 72 61 6d 73 2e 68 0 63 70 70 0 0 5 0 | _arams.h cpp 5
32:47 | 164 69 72 2e 68 0 2e 68 0 63 70 70 0 0 6 0 | dir.h .h cpp 6
48:63 | 166 61 74 2e 68 0 2e 68 0 63 70 70 0 0 8 0 | fat.h .h cpp 8
64:79 | 166 69 6c 65 73 79 73 74 2e 68 0 70 0 0 9 0 | filesyst.h p 9

80:95 | 177 65 6c 63 6f 6d 65 2e 74 78 74 0 38 0 a 0 | welcome.txt 0 \n
96:111 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
112:127 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
128:143 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
144:159 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
160:175 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
176:191 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
192:207 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
208:223 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 
224:239 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 

```

Figure 9. Response of “hexdump” command

The table 2 gives the list of sub commands of “hexdump” command and their actions. These sub-commands help file carver developer to view different portions of the challenge file. Correctness of the challenge file can be verified with the help of the subcommands. Then the correctness of their file carver algorithms can be verified by using the verified challenge file as input media.

TABLE 2. SUBCOMMANDS OF “HEXDUMP”

Command	Action
d	To display <u>d</u> irectory region
f	To display <u>f</u> at region
n	To display <u>n</u> ext page
p	To display <u>p</u> revious page
N	To display <u>N</u> ext Cluster

Finally “format” command is executed as part of the fine-tuning phase that erases the metadata. A disk containing files content but not containing the metadata of files is the prime condition for any file carving algorithm.

VI. BENEFITS OF AUTOMATED GENERATION

- Avoid purchase of physical used disks.
- Disk size flexibility
- Natural generation of challenge file
- Avoid artificial /manual generation of challenge file
- Various scenarios can be generated on each virtual disk
- Number of virtual disks can be created
- The virtual disk can be zipped and sent easily over an internet
- Avoids read/write operations on physical disk in raw mode
- Researchers can implement their new ideas of file system without the need to work at kernel level
- Before the final step of automated construction, the directory and fat regions contain the correct answer to the proposed challenge.
- International workshops like DFRWS can create challenge files using USFS and compare participants results with the correct results available in the directory and fat regions before fine-tuning step
- The source code can be used on any platform where as FUSE and Kernel level programming are specific to Linux

VII. CONCLUSION

The User Space File System (USFS) is need of the hour until new file systems take over for solving the problem of file carving altogether. At present, file carving algorithms are file type specific and for future file types new techniques need to be designed. No file carving algorithm can yield any useful result if the criminal executes a loop that zeroes out the entire disk. This hazardous operation needs O(N) time where N is the number of clusters on a disk. And hence it is practically possible for a criminal. Methods to prevent such operations by criminals need to be invented. In future work authors would like to try to address this problem.

REFERENCES

- [1] Nasir Memon, Anandabrata Pal, “Automated Reassembly of File Fragmented Images Using Greedy Algorithms”, IEEE Transactions on Image Processing, Volume 15, No.2, February, 2006
- [2] Maurice J Bach Pearson, “The Design of the Unix Operating System” (Pearson)
- [3] Peter Abel, “IBM PC Assembly Language and Programming”, Third Edition – PHI..
- [4] Kulesh Shanmugasundaram, Nasir Memon, Automatic Reassembly of Document Fragments via Context Based Statistical Models, Department of Computer and Information Science Polytechnic University Brooklyn.

