Anandabrata Pal and Nasir Memon

The Evolution of File Carving

The benefits and problems of forensics recovery

© BRAND X PICTURES

Year by year, the number of computers and other digital devices being used is increasing. The recent Pew Research Center Globalization Review [1] showed that 26 of the 36 countries surveyed had increased their computer usage. This increase is occurring simultaneously with an increase in usage of other digital devices, such as cell phones. In fact, in the United States alone 81% of the population now owns a cell phone, which is a 20% increase compared to 2002. Some countries, including Russia, have shown upwards of a 50% increase in cell phone ownership.

Computers are now one of many devices where digital data is stored. Devices such as cell phones, music players, and digital cameras all now have some form of internal storage or else allow data to be stored to external devices like flash cards, memory sticks, and solid-state devices (SSDs). With this huge increase in digital data storage, the need to recover data due to human error, device malfunction, or deliberate sabotage has also increased.

Data recovery is a key component of the disaster recovery, forensics, and e-discovery markets. Digital data recovery can consist of both software and hardware techniques. Hardware techniques are often used to extract data from corrupted or physically damaged disks. Once the data has been extracted, software recovery techniques are often required to order and make sense of the data. In this article, we will be solely discussing software techniques for recovery of data with a focus on digital forensics. We will begin by providing a quick overview of traditional data recovery techniques and then describe the problems involved with such techniques. We then introduce the techniques involved in file carving.

Digital Object Identifier 10.1109/MSP.2008.931081

1053-5888/09/$25.00©2009IEEE

TRADITIONAL DATA RECOVERY

Traditional data recovery methods rely on file system structures like file tables to recover data that has been deleted. This is because most file systems do not touch the physical location of the files during a deletion, they simply mark the location as being available for storing information. After deletion, the entry of the file in the file table may still be present and the information linking the clusters to the file deleted may also still be present, and as a result, such a file can be easily recovered. Another advantage of accessing file system structures is to also be able to identify and quickly extract existing undeleted data, therefore, only the areas in the disk that are considered unallocated, need to be parsed. However, when the file system structures are not present, corrupt, or have been deliberately removed, the data while present, can not be accessed via traditional means.

[THE PRIMARY PROBLEM WITH THE HAMILTONIAN PATH FORMULATION IS THAT IT DOES NOT TAKE INTO ACCOUNT THAT IN REAL SYSTEMS MULTIPLE FILES ARE FRAGMENTED TOGETHER AND THAT RECOVERY CAN BE IMPROVED IF THE STATISTICS OF MULTIPLE FILES ARE TAKEN INTO ACCOUNT.]

FILE CARVING

Once it became clear that traditional recovery techniques may fail on data sets, additional techniques needed to be introduced to recover forensically important user files. Some examples of these files are Microsoft Office documents, digital pictures, and e-mails. More often than not, the files of importance for forensic recovery are those that are created and modified by the users. Operating system and application files can be reinstalled, however, user files not backed up and deleted require recovery. File carving is a forensics technique that recovers files based merely on file structure and content and without any matching file system meta-data. File carving is most often used to recover files from the unallocated space in a drive. nallocated space refers to the area of the drive which is no longer holding any file information as indicated by the file system structures like the file table. In the case of damaged or missing file system structures, this may involve the whole drive.

FILE SYSTEMS AND FRAGMENTATION

Before describing what a file system is and how files are stored and deleted, we want to briefly introduce the physical blocks on a disk that are used to store data. Most disks are described in terms of data blocks called sectors and clusters. The cluster size is a function of the sector size (1). The disk size or capacity is calculated by multiplying the cluster size by the number of clusters (2). It is very important to note that a cluster (not a sector) represents the smallest unit of storage that is addressable (can be written to or read). Therefore, files are typically stored in terms of clusters and not sectors. Typical values of clusters range from 512–32,000 B. Cluster sizes are normally multiples of 512 B.

$$\text{Cluster Size} = \text{Sector Size} \times \text{Sectors per Cluster}. \quad (1)$$

$$\text{Disk Size} = \text{Cluster Size} \times \text{Count}. \quad (2)$$

In his book, *File System Forensic Analysis*, Carrier [12] describes a file system by stating that it "consists of structural and user data that are organized such that the computer knows where to find them." While there are currently many file systems in use today, we will begin by describing the operation of two well known and commonly used file systems: file allocation table (FAT)-32 and new technology file system (NTFS).

FAT32

Microsoft introduced the file system FAT in 1980 and has updated it throughout the years. As disk storage increased, the need to improve the FAT system resulted in FAT-12, FAT-16, and now FAT-32. Even though Microsoft introduced NTFS for modern computing systems, most memory cards, universal serial bus (USB) drives, and other removable media still use FAT-32, since NTFS is more complicated and takes up a large amount of disk space.

FILE ALLOCATION

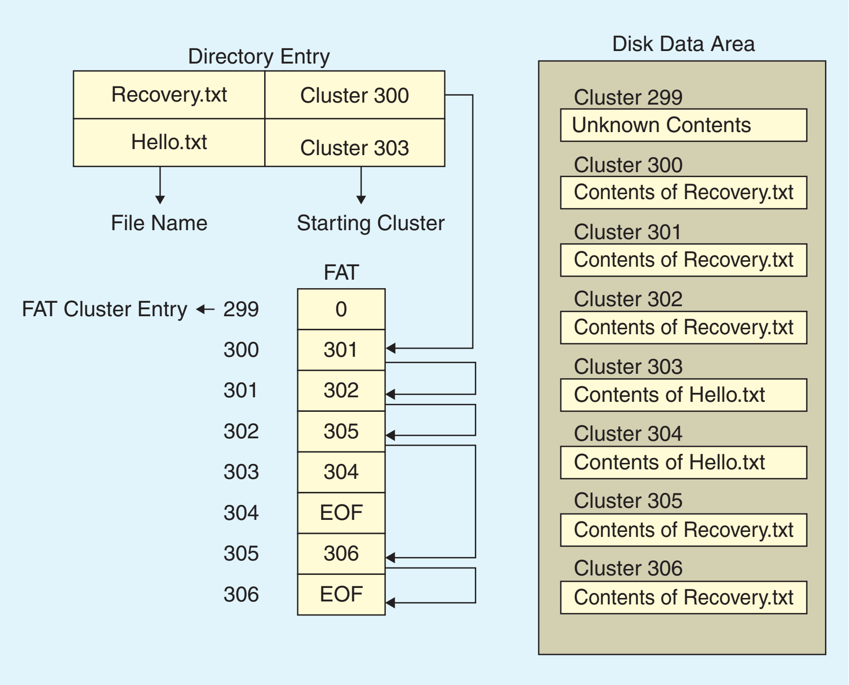
File allocation in FAT-32 is a two step process. First, the file is assigned to a directory/folder. If the assigned directory/folder is the root of the drive, then the file name is stored in the root directory entry, if not, one must parse the root directory entry to find the directory that is in the path of the final directory to which the file is assigned to. Along with the file name, the file entry contains the starting cluster number of the file. The starting cluster number represents the cluster where the file's contents begin. In reality, additional information, including access, creation, and modification time stamps along with long file names are also stored in the directory entry. To retrieve the file, the file system looks at the starting cluster and then goes to the starting cluster index in the file allocation table. The FAT can be considered to be an array of cluster numbers pointing to the next cluster of a file. Therefore, when a file is retrieved, the file system goes to the starting cluster index in the FAT, and gets the next cluster number for that file. This cluster in turn can point to another cluster. This process is repeated until a cluster has the hex value "FF" indicating end of file (EOF). The file system is then able to retrieve the file by reading the actual clusters on the disk.

An example of this is creating a file called "recovery.txt," which requires five clusters to store its contents. The file system may find that the cluster number 300 is where the file should start to be stored. As shown in Figure 1, a root entry for "recovery.txt" is made indicating the starting cluster of the file shown as cluster number 300. The file system stores the remaining clusters in cluster numbers 301, 302, 305, and 306 and makes the appropriate changes to the FAT. Note that clusters 303 and 304 are used for storage of another file "hello.txt."

DELETION

When deleting a file, the file system does not actually go to the clusters of the file and zero them out. Instead, all the file system does is go to the cluster links in the FAT and set them to hex value "00" that indicates that the clusters have been unallocated. The actual contents of the drive are still present after the deletion. Interestingly, while the FAT cluster entries for each of the clusters of the deleted file are indicated to be unallocated, the directory entry of the file still points to the starting cluster with a special character being used to mark that this entry represents a deleted file.

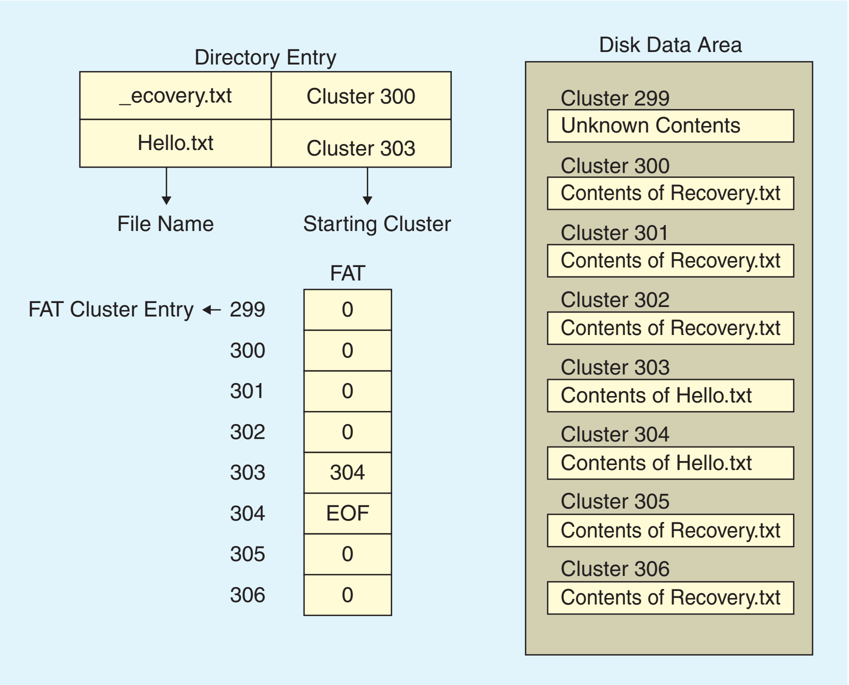
Figure 2 shows what happens to the FAT when recovery.txt is deleted. The clusters represented by the file in the FAT are changed to zero to indicate availability, however, the actual cluster contents of recovery.txt have not been removed as yet. The first byte of the name is also changed in the file entry directory to represent that the file was deleted.



[FIG1] FAT file allocation for file called recovery.txt, which is stored in clusters 300, 301, 302, 305, and 306. File hello.txt is stored in clusters 303 and 304.

RECOVERY

Since the cluster sequence in the FAT is set to hex value "00" (unallocated) on a file's deletion, recovery programs have to check that the file starting cluster is not in use, and then assume that the rest of the file is stored in sequential clusters. Once a file is deleted, there is absolutely no way to determine the exact sequence of clusters to reconstruct the file using the file system meta-data alone. In the example using recovery.txt, traditional file recovery programs will find the entry for "_ecovery.txt" and see that the starting cluster for the deleted file was 300. Some recovery programs will only recover a partial file (clusters 300, 301, 302), while the smarter file recovery programs are able to recover the whole file because they see that clusters 303 and 304 are in use by another file, and that clusters 305 and 306 are available, so they will merge the unallocated clusters only. However, what happens if the file is broken up into more than one piece or there are unallocated clusters belonging to another file in between? For example, if hello.txt and recovery.txt are deleted, then file recovery becomes much more difficult without analyzing the contents or structure of the files.



[FIG2] FAT file deletion for file called recovery.txt, which was stored in clusters 300, 301, 302, 305, and 306. FAT indexes representing the clusters were set to 0 to indicate that they are now available for use. File hello.txt was not deleted.

NTFS

FAT-32, while an improvement over earlier versions of the FAT, does not support files of greater than 4 GB. Another limitation is the time taken to determine the free space for the FAT increases with an increase in the number of clusters in the system. Finally, all versions of the FAT do not provide enhanced security features and support for meta-data. As a result, Microsoft introduced NTFS with Windows NT in 1993, as the preferred file system for its modern operating systems.

FILE ALLOCATION

In NTFS, file information is stored in a B-Tree [12], however, unlike the FAT, NTFS has a separate structure called a bitmap (BMP) to represent the allocation status of clusters. In the BMP file ($bitmap) representing clusters, each cluster is assigned a value of one if it belongs to an existing file, and a zero otherwise. By parsing the BMP, NTFS is able to quickly determine the best place to put a file in to prevent fragmentation. In addition, just as in the FAT, NTFS stores the file cluster links.

DELETION

When a file is deleted, its associated clusters in the BMP are set to zero. Again, just as in the FAT, the actual data is not deleted when deleting files. Unlike the FAT, the file cluster links are not deleted either so if the deleted file entry is present, then the original cluster links are also still present. This makes recovery much easier in NTFS, as the full cluster link representing the file is still present. The clusters are simply shown as being deallocated in the $bitmap file.

> FILE CARVING IS A FORENSICS TECHNIQUE THAT RECOVERS FILES BASED MERELY ON FILE STRUCTURE AND CONTENT AND WITHOUT ANY MATCHING FILE SYSTEM META-DATA.

RECOVERY

In NTFS, file recovery is quite straightforward when the file entry is still present. This is because the file's cluster numbers should also be present, and all that needs to be done is to verify that the file's clusters have not been overwritten, or are not allocated to another file. In the case where the file system is damaged or the deleted file's entries removed, file system meta-data can not be used for recovery.

FRAGMENTATION

As files are added, modified, and deleted, most file systems get fragmented. File fragmentation is said to occur when a file is not stored in the correct sequence on consecutive clusters on disk. In other words, if a file is fragmented, the sequence of clusters from the start of a file to the end of the file will result in an incorrect reconstruction of the file. The example of "recovery.txt" provided in Figure 1 provides a simplified example of a fragmented file. In the figure, the "recovery.txt" has been broken into two fragments. The first fragment starts at cluster 300 and ends at cluster 302. The second fragment starts at cluster 305 and ends at cluster 306. This file is considered to be bifragmented as it has only two fragments. Garfinkel showed that bifragmentation (two fragments only) is the most common type of fragmentation [2], however, files fragmented into three or more pieces are not uncommon.

Garfinkel's fragmentation statistics [2] come from identifying over 350 disks containing FAT, NTFS, and Unix file system (UFS). He shows that while fragmentation in a typical disk is low, the fragmentation rate of forensically important files (user files) like that of e-mail, jpeg, and Microsoft Word are high. The fragmentation rate of jpegs was found to be 16%, Microsoft Word documents had 17% fragmentation, audio video interleave (AVI) (movie format) had a 22% fragmentation rate, and personal information store (PST) files (MS-Outlook) had a whopping 58% fragmentation rate. Fragmentation typically occurs due to low disk space, appending/editing files, wear-leveling alogorithms in next generation devices, and file systems.

LOW DISK SPACE

If disk space is low and the disk is already fragmented, there may be many small groups of clusters that are available for storing information. However, future files to be stored may be larger than the largest of these free groups of clusters, and as a result such files will be stored in a fragmented manner. For example, if the largest consecutive clusters for allocation available is ten clusters, and a file needs 14 clusters for allocation, then most file systems will store the file in the first ten clusters and then store the remaining four clusters in another area on the disk that has unallocated clusters available. This will result in fragmentation.

APPENDING/EDITING FILES

If a file is saved on disk and then additional files are also saved starting after the cluster that the original file ended at, fragmentation may occur if the original file is then appended to (and increases in size larger than the cluster size). Some file systems, like the Amiga Smart File System [18], may attempt to move the whole file in such scenarios. Some other file systems like UFS attempt to provide "extents" that are attempts to preallocate longer chunks in anticipation of appending [20]. Another technique, called delayed allocation, used in file systems like XFS [21] and ZFS, reserve file system clusters but attempt to delay the physical allocation of the clusters until the operating system forces a flushing of the contents. However, while some of these techniques are able to reduce fragmentation, they are unable to eliminate fragmentation completely. Constant editing, deletion, and additions of e-mails are the primary reasons why PST files (Microsoft Outlook files) are highly fragmented (58% of pst files analyzed in the wild [2]).

WEAR-LEVELING ALGORITHMS IN NEXT GENERATION DEVICES

SSDs currently utilize proprietary wear-leveling algorithms to store data on the disk [16] in order to enhance its lifetime. Wear leveling is an algorithm by which the controller in the storage device remaps logical block addresses to different physical block addresses. If the controller gets damaged or corrupted, then any data extracted will be inherently fragmented, with no easy way of determining the correct sequence of clusters to recover files.

FILE SYSTEM

In rare cases, the file system itself will force fragmentation. The UFS will fragment files that are long or have bytes at the end of the file that will not fit into an even number of sectors [12].

In Table 1, we give basic definitions concerning fragmentation that will be used throughout the article. Figure 3 provides a simple visualization of these definitions.

A fragmented file lacking the cluster-chain information cannot be recovered with traditional file recovery techniques. Also, for newer and more esoteric file systems, recovery may not be possible based solely on file system meta-data unless support is directly provided. To recover files that do not have any file table information present, recovery techniques that analyzed the structure and contents of the file were introduced. These techniques are called file carving. In the next section, we detail the process required to recover both fragmented and nonfragmented files using different forms of file carving.

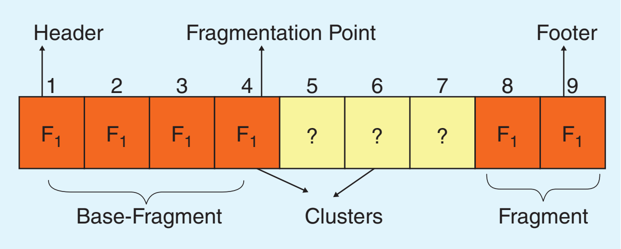
FILE STRUCTURE-BASED CARVERS

File carving was born due to the problems inherent with recovery from file system meta-data alone. File carving does not use the file system information directly to recover files. Instead, it uses knowledge of the structure of files. More advanced carvers not only use knowledge of the structure of files but also use the contents of individual files to recover data.

Given a digital storage device, a file carver may or may not recognize the file system being used, and it may or may not trust the information in the file system to be accurate. As a result, it is up to the carver to determine the "unallocated" clusters in the disk to carve from. This may involve all clusters in the disk. To reduce the amount of a disk to recover from, a number of forensic applications are able to identify a large number of files that are common to operating systems and applications based on their MD5 Hash and keywords. Both Encase and Forensic Toolkit (FTK), the two leading commercial disk forensic software providers, provide this option to quickly eliminate common and well-known files.

The first generation of file carvers used "magic numbers," or to be more precise, byte sequences at prescribed offsets to identify and recover files. File carving techniques were first used for files that contain a "header" and "footer." A header identifies the starting bytes of a file and the footer identifies the ending bytes of the file. Headers are used by some operating systems to determine which application to open a file with. In regard to jpegs, starting clusters must begin with the hex sequence FFD8, and a footer is a cluster containing the hex sequence FFD9. A file carver relying on structure alone will attempt to merge and return all unallocated clusters between the header and footer.

There are many file types that may not contain footers but instead contain other information like the file size. An example of this is the Windows BMP file that contains the size of the file in bytes. Again, traditional file carvers would recover files by identifying the header and then merging as many unallocated sequential clusters following the header as required to equal the file size.



[FIG3] File F_1 has been broken into two fragments spanning six clusters, with three clusters in between not belonging to F_1.

Foremost [7], developed by the United States Air Force Office of Special Investigations, was one of the first file carvers that implemented sequential header to footer carving and also implemented carving based on a header and the size of the file. Golden et al. [6] then proposed Scalpel, which was built on the Foremost engine, but greatly improved its performance and memory usage.

The problem with the first generation of file structure-based carvers was that they simply extract data between a known header and footer (or ending point determined by size), with the assumption being that the file is not fragmented and there is no missing information between the header and the footer. As a result, these file carvers frequently provide results that have "garbage" in the middle. Figure 4 provides an example of a jpeg file that was recovered based solely on the header footer technique. From the image, it can be seen that a large number of clusters were decoded that did not belong to the image before the decoding completely failed. The image belongs to the DFRWS 2006 [22] file carving challenge, which does not contain

[TABLE 1] DEFINITIONS OF TERMS.

TERM	DEFINITION
CLUSTER	THIS IS THE SIZE OF THE SMALLEST DATA UNIT THAT CAN BE WRITTEN TO DISK AND b_y WILL DENOTE THE CLUSTER NUMBERED y IN THE ACCESS ORDER.
HEADER	THIS IS A CLUSTER THAT CONTAINS THE STARTING POINT OF A FILE.
FOOTER	THIS IS A CLUSTER THAT CONTAINS THE ENDING DATA OF A FILE.
FRAGMENT	A FRAGMENT IS CONSIDERED TO BE ONE OR MORE CLUSTERS OF A FILE THAT ARE NOT SEQUENTIALLY CONNECTED TO OTHER CLUSTERS OF THE SAME FILE. FRAGMENTED FILES ARE CONSIDERED TO HAVE TWO OR MORE FRAGMENTS (THOUGH ONE OR MORE OF THESE MAY NOT BE PRESENT ON THE DISK ANYMORE). EACH FRAGMENT OF A FILE IS ASSUMED TO BE SEPARATED FROM EACH OTHER BY AN UNKNOWN NUMBER OF CLUSTERS.
BASE-FRAGMENT	THE STARTING FRAGMENT OF A FILE THAT CONTAINS THE HEADER AS ITS FIRST CLUSTER.
FRAGMENTATION POINT	THIS IS THE LAST CLUSTER BELONGING TO A FILE BEFORE FRAGMENTATION OCCURS. A FILE MAY HAVE MULTIPLE FRAGMENTATION POINTS IF IT HAS MULTIPLE FRAGMENTS.
FRAGMENTATION AREA	A SET OF CONSECUTIVE CLUSTERS b_y, b_{y+1}, b_{y+2}, b_{y+3}. . . CONTAINING THE FRAGMENTATION POINT.

any file system meta-data. The DFRWS challenge test-sets [22], [23] were specifically created to test and stimulate research in the area of fragmented file carving.

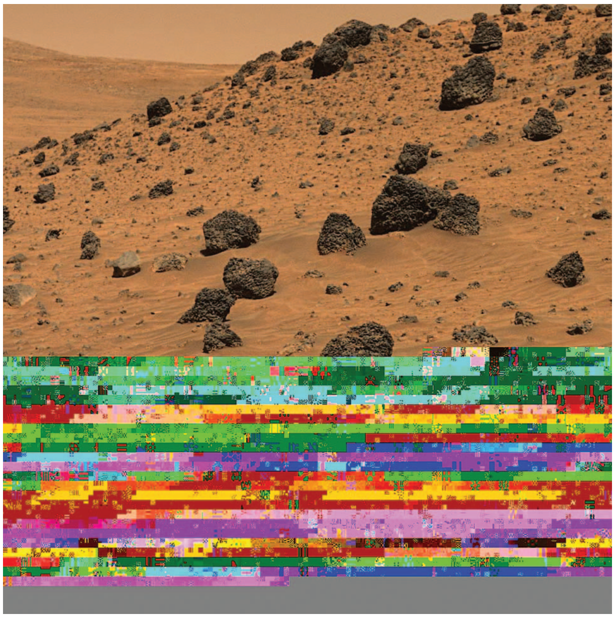
DECODING IS THE PROCESS OF TRANSFORMING INFORMATION IN THE DATA CLUSTERS ASSOCIATED WITH A FILE INTO ITS ORIGINAL FORMAT THAT DESCRIBES THE ACTUAL CONTENT.

GRAPH THEORETIC CARVERS

The problem of recovery of fragmented files with file structure-based carvers became readily apparent, and there was a subsequent push to develop other techniques that could be used to recover fragmented files. Shanmugasundaram et al. [8] were some of the first to tackle recovery of fragmented files. They formulated the problem of recovery as a Hamiltonian path problem and provided the alpha-beta heuristic from game theory to solve this problem.

REASSEMBLY AS A HAMILTONIAN PATH PROBLEM

Given a set of unallocated clusters $b_0, b_1 \ldots b_n$ belonging to a document A, one would like to compute a permutation Π of the set that represents the original structure of the document. To determine the correct cluster ordering one needs to identify fragment pairs that are adjacent in the original document. This is achieved by assigning candidate weights ($W_{x,y}$) between two clusters b_x and b_y that represents the likelihood that cluster b_y follows b_x. Once these weights are assigned, the permutation of the clusters that leads to correct reassembly, among all possible permutations, is likely to maximize the sum of candidate weights of adjacent clusters. This observation provides a technique to identify the correct reassembly with high probability. That is to compute the permutation Π such that the sum of the ordering of the weights is maximized over all possible permutations Π of degree n.



[FIG4] Incorrect sequential recovery shown for a fragmented image from DFRWS 2006.

This permutation is most likely to be the one that leads to correct reconstruction of the document. This can also be abstracted as a graph problem if one takes the set of all candidate weights (W) to form an adjacency matrix of a complete graph of n vertices, where vertex x represents cluster b_x and the edge weight $W_{x,y}$ represent the likelihood of cluster b_y following cluster b_x. The proper sequence Π is a path in this graph that traverses all the vertices and maximizes the sum of candidate weights along that path. Finding this path is equivalent to finding a maximum weight Hamiltonian path in a complete graph.

The question then becomes, how does one assign candidate weights between two clusters? Kulesh et al. described the process of weighing two clusters using a technique called prediction by partial matching (PPM). PPM is a finite-order context modeling technique first introduced by Cleary and Witten [24] and has become a benchmark for lossless data compression techniques. PPM employs a suite of fixed-order context models, from zero up to some predetermined maximum k, to predict the upcoming characters. Using PPM, each cluster is individually processed and the resulting statistics are combined to form a single model for a document. The weight $W_{x,y}$ is then calculated by sliding a window across the ending bytes of cluster b_x and the starting bytes of cluster b_y. Figure 5 shows the weight being calculated using a sliding window of size four. The window is initially put on the edge of the cluster b_x and the model is looked up to see the probability of "wor" being followed by the letter "l." The window then slides one byte and now the probability of the "orl" followed by "d" is found and this is multiplied to the previous probability. We repeat the process until no part of the window is in cluster b_x. This provides us with the candidate weight for $W_{x,y}$.

While PPM is shown to be good for structured data like text, its performance is not as good for images and compressed documents.

Pal et al. [4], extended the work done by Shanmugasundaram by introducing a system for fragmented file recovery for images. In their research, they used 24-b Windows BMPs that were completely fragmented, and they developed their test scenarios with absolutely no file system meta-data present. In order to determine the weight of an edge between two clusters, they compared the image boundaries between each cluster and the subsequent cluster and developed a weighting technique that analyzed the pixel differences between two clusters as shown in Figure 6. The sum of differences of the pixels across the boundary as shown in the figure is calculated and then stored as the weight. If w is the width of an image then the last w pixels of b_x are compared against the first w pixels of cluster b_y when computing W_{xy}. The weight calculated between clusters b_x and b_y (W_{xy}) is normally different than the weight between b_y and b_x (W_{yx}).

BIFRAGMENT GAP CARVING RECOVERY OCCURS BY EXHAUSTIVELY SEARCHING ALL COMBINATIONS OF CLUSTERS BETWEEN AN IDENTIFIED HEADER AND FOOTER WHILE EXCLUDING DIFFERENT NUMBER OF CLUSTERS UNTIL A SUCCESSFUL DECODING/VALIDATION IS POSSIBLE.

REASSEMBLY AS A K-VERTEX DISJOINT PATH PROBLEM

The primary problem with the Hamiltonian path formulation is that it does not take into account that in real systems multiple files are fragmented together and that recovery can be improved if the statistics of multiple files are taken into account. Pal et al. subsequently refined the problem into that of a k-vertex disjoint path problem [3]. If one considers every cluster in the unallocated space to be a vertex in a graph and the edge between vertices to be weighted according to the likelihood that one cluster follows another, then the problem of reassembly is that of finding k-vertex disjoint paths, where k is equal to the number of files identified via their headers. The reason that this is a disjoint path problem is because each cluster on the disk is assumed to belong to one and only one file. The k-vertex disjoint path problem is also known to be nondeterministic polynomial-time (NP)-hard.

In the case of 24-b BMPs, the starting points for each file (i.e., the header - b_h) is known and the number of clusters (vertices) to recover the file is also known as the size of each cluster is fixed, and the size of the BMP can be determined by examining the header.

Weights are precomputed for all clusters, and then the authors' proposed a set of eight algorithms to attempt to recover the images. The interesting algorithms among the eight were unique path (UP) algorithms. An UP algorithm is one in which each cluster is assigned to one and only one file. As mentioned earlier, in file systems, each cluster normally belongs to no more than one file, therefore, an UP algorithm is more realistic. However, an UP algorithm has the disadvantage that if a cluster is incorrectly assigned to a file, the potential exists for the file it actually belongs to (if any) to also be recovered incorrectly. In other words, mistakes can cascade. Of the UP algorithms presented, we wish to discuss the two that provided the best results.

PARALLEL UNIQUE PATH

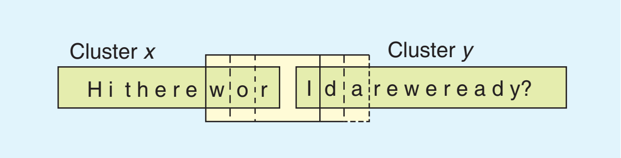
Parallel unique path (PUP) is a variation of Dijkstras single source shortest path algorithm [19], which is used to recover images simultaneously. Beginning with the image headers, the algorithm picks the best cluster match for each header from among the available clusters. The best match is determined by the edge weights discussed earlier. From the list of these best matches, the best cluster match (b_x) is taken and assigned to the header (b_hk) of the image. This cluster is then removed from the set of available clusters (thus ensuring this is an UP algorithm). The best match for b_x is now determined and compared against the best matches of the remaining images. This process continues until all the images are recovered.

More formally, the k file headers ($b_{h1}, b_{h2}, \ldots b_{hk}$) are stored as the starting clusters in the reconstruction paths P_i for each of the k files. A set $S = (b_{s1}, b_{s2}, \ldots, b_{sk})$ of current clusters is maintained for processing, where b_{si} is the current cluster for the ith file. Initially, all the k starting header clusters are stored as the current clusters for each file (i.e. $b_{si} = b_{\mathrm{hi}}$). The best greedy match for each of the k starting clusters is then found and stored in the set $T = (b_{t1}, b_{t2}, \ldots, b_{tk})$ where b_{ti} represents the best match for b_{si}. From the set T of best matches the cluster with the overall best matching metric is chosen.

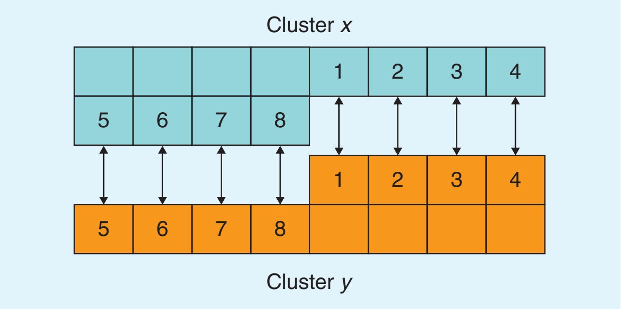
Assuming that this best cluster is b_{ti}, the following steps are undertaken:

1) Add b_{ti} to reconstruction path of ith file, (i.e. $P_i = P_i || b_{ti}$).
2) Replace the current cluster in set S for ith file (i.e. $b_{si} = b_{\mathrm{hi}}$).
3) Evaluate the new set T of best matches for S.
4) Again find best cluster b_{ti} in T.
5) Repeat Step 1 until all files are built.

Figure 7 shows an example of the algorithm where there are three files being reconstructed. Figure 7(a) shows the header clusters H_1, H_2, and H_3 of the three files and their best matches. The best of all the matches is presented with a dotted line and is the H_2-6 pair of clusters. Figure 7(b) now shows the new set of best matches after cluster six has been added to the reconstruction path of H_2. Now cluster four is chosen once each for clusters H_1 and six. However, the pair H_1-4 is the best and therefore four is added to the reconstruction path of H_1 and the next best match for cluster six is determined [Figure 7(c)]. This process continues until all files are reconstructed.



[FIG5] Incorrect sequential recovery shown for an image from DFRWS 2006 that is fragmented.



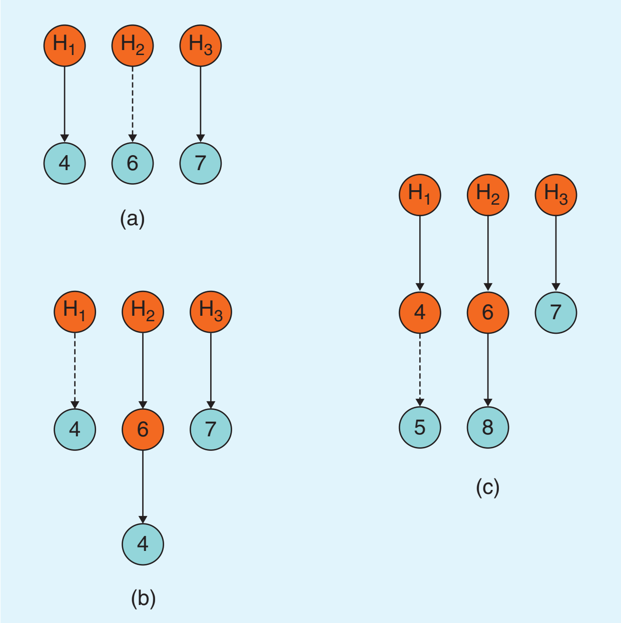
[FIG6] Image weighting technique where the sum of difference between each same numbered pixel in cluster X is compared with cluster Y.

SHORTEST PATH FIRST

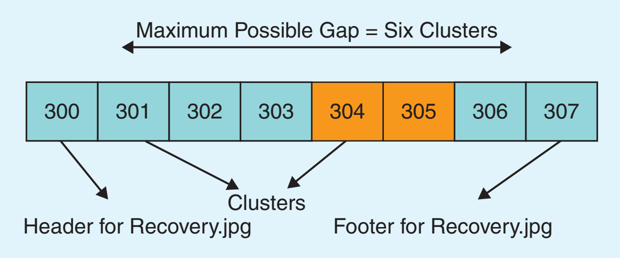
Shortest path first (SPF) is an algorithm that assumes that the best recoveries have the lowest average path costs. The average path cost is simply the sum of the weights between the clusters of a recovered file divided by the number of clusters. This algorithm reconstructs each image one at a time. However, after an image is reconstructed the clusters assigned to the image are not removed, only the average path cost is calculated. All the clusters in the reconstruction of the image are still available for the reconstruction of the remainder of the images. This process is repeated until all the image average path costs are calculated. Then the image with the lowest path cost is assumed to be the best recovery and the clusters assigned to its reconstruction are removed. Each of the remaining images that used the clusters removed have to redo their reassemblies with the remaining clusters, and their new average path cost is calculated. Once this process is completed for the remaining images, the one with the lowest average path cost is again removed, and this process continues until all images are recovered.

REASSEMBLY IS THE FINAL STEP OF THE SMARTCARVER. IT INVOLVES FINDING THE FRAGMENTATION POINT OF EACH UNRECOVERED FILE AND THEN FINDING THE NEXT FRAGMENT'S STARTING POINT (CLUSTER).

In terms of results, the SPF algorithms produced the very best accuracy with 88% of files being reconstructed, and the PUP algorithm reconstructed 83% of files. However, the PUP algorithm was substantially faster and scaled better than the SPF algorithm. The authors proved that their techniques could be used for image recovery under highly fragmented scenarios, but the primary issue with their techniques is that they do not scale. Since edge weights are precomputed and sorted so as to find the best matches quickly, the complexity of the algorithms are dominated by this step which is equal to $O(n^2 logn)$. Today typical hard-disks contain millions of clusters and, as a result, precomputing weights between all potential clusters is infeasible.



[FIG7] Simplified example of PUP algorithm.



[FIG8] Simplified example of BGC algorithm, where for recovery a gap size of two must be chosen and the clusters 300–303 and clusters 306–307 result in successful recovery of file.

BIFRAGMENT GAP CARVING

One of the first published file carving techniques to attempt to recover data from real-world data sets was introduced by Simson Garfinkel [2]. Garfinkel introduced the fast object validation technique for recovery of bifragmented files. Fast object validation only works for files with a known header and footer. In addition, the files should either be decodable or be able to be validated via their structure. Decoding is the process of transforming information in the data clusters associated with a file into its original format that describes the actual content. Many file types, like Joint Photographic Experts Group (JPEG), Moving Picture Experts Group (MPEG), ZIP, etc., have to be decoded before their content can be understood. Object validation is the process of verifying if a file obeys the structured rules of its file type. Therefore, an object validator will indicate whether a cluster violates the structure or rules required of the specific file or file type. For example in the portable network graphics (PNG) file format, the data can be validated through cyclic redundancy checking, and a mismatch will indicate either data corruption or fragmentation. A decoder can be trivially used as an object validator by observing whether or not it can correctly decode each cluster in the sequence. Plain text files and Windows 24-b BMPs are examples of file types that can't be validated and don't require decoding.

Bifragment gap carving (BGC) recovery occurs by exhaustively searching all combinations of clusters between an identified header and footer while excluding different number of clusters until a successful decoding/validation is possible. Let b_h be the header cluster, b_f be the last cluster of the first fragment of a file (also known as the fragmentation point), b_s be the starting cluster of the second fragment, and b_z be the footer cluster. Clusters b_h and b_z are known and b_f and b_s have to be determined. For each gap size g, starting with size one, all combinations of b_f and b_s are designated so that they are exactly g clusters apart, i.e., $s - f = g$.

A validator/decoder is then run on the byte stream representing clusters b_h to b_f and b_s to b_z. If the validation fails b_f and b_s are readjusted and the process continued until all choices of b_f and b_s are tried for that gap size, after which the gap size is incremented. This continues until a validation returns true or the gap size can't be increased any further.

Figure 8 shows a file called "recovery.jpg" that is fragmented into two fragments. The base-fragment starts from cluster 300 and the ending cluster for the fragment is 303. The starting cluster for the next fragment is 306 and it ends at 307. BFC would find the correct recovery via fast object validation when the gap size is two and b_f is 303 and b_s is 306.

This technique performs satisfactorily when the two fragments are close to each other; however, it has the following limitations for the more general case:

1) The technique does not scale for files fragmented with large gaps. If n is the number of clusters between b_h and b_z then in the worst case, n^2 object validations may be required before a successful recovery.

2) It was not designed for files with more than two fragments.

3) Successful decoding/validation does not always imply that a file was reconstructed correctly. Figure 9 from DFRWS 2007 is an example of a successfully decoded but incorrectly recovered JPEG file.

4) It only works with files that have a structure that can be validated/decoded.

5) Missing or corrupted clusters will result in the worst case often.

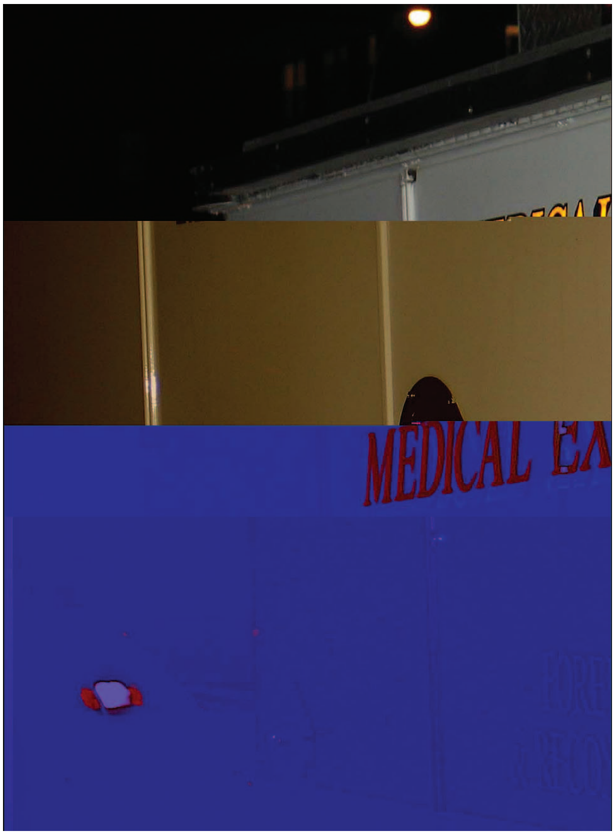
SMARTCARVER

Here we describe the design of a file carver that is not limited to recovering files containing two fragments. Pal et al. [5] recognized the problems inherent with their initial research into recovery of randomly fragmented images using precomputed weights and provided the framework and results for a carver that takes into account the typical fragmentation behavior of a disk and one that scales to huge disks.

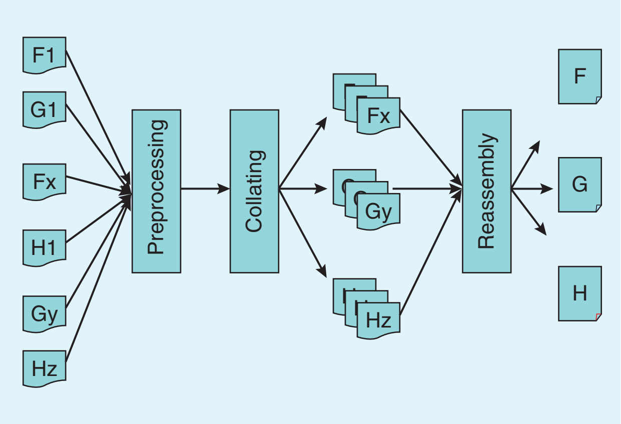
Figure 10 shows the three key components of the SmartCarver file carver that can handle both fragmented and unfragmented data across a variety of digital devices and file systems. The first phase of such a file carver is that of preprocessing, where file system data clusters are decrypted or decompressed as needed. The next phase is that of collation, where data clusters are classified as belonging to a file type, and a further attempt is optionally made to determine if they belong to a particular file. The final step is that of reassembly, where clusters identified and merged in the collation phase are pieced together to reconstruct files.

PREPROCESSING

The preprocessing phase needs to be applied on drives that have their contents compressed or encrypted. Windows Vista comes with BitLocker, which, if enabled, encrypts all the data on a drive. If file carving techniques are to be used (or any file recovery technique for that matter) then the disk must be decrypted. Recently, there have been demonstrations of memory-based attacks to determine the encryption key for encrypted drives on a computer by analyzing ghost images of the memory. Similarly, if the device utilizes compression technology then the data clusters need to be decompressed as well. Regardless of the encryption/compression used, the data



[FIG9] Fully validated but incorrect JPEG image from the DFRWS 2007 test set.



[FIG10] The three components required in a file carver handling fragmented files are (a) preprocessing, (b) collation, and (c) reassembly.

clusters must be decrypted/decompressed to continue the recovery process.

The preprocessing phase also removes all the known (allocated) clusters from the disk based on file system meta-data. This step can be skipped if it is suspected that the file table meta-data has been tampered with or is corrupt. However, the benefits of reducing the number of clusters to analyze by removing all known allocated clusters are great in terms of both speed and efficiency. If the file system is missing or the file system is determined to be corrupt, all clusters are considered to be "unallocated." Once the preprocessing phase is complete, the collation phase is started.

COLLATION

The collation phase of the SmartCarver operates on the unallocated clusters and is responsible for disk cluster classification. Disk cluster classification is the process by which a cluster is classified as belonging to one or more file types. A file type represents a specific file format, like that of jpeg or the Microsoft Office format. Disk cluster classification is important because it allows us to reduce the number of clusters to consider for each file being recovered. For example, a cluster containing pure ASCII English text would probably not belong in the reconstruction of a 24-b BMP or png (images) file. Research has been conducted in classifying byte streams, whether they are network traffic, disk clusters, or complete files. Some well-known techniques used for file type identification are the following:

- *Keyword/Pattern Matching* simply involves looking for sequences (sometimes at specific byte offset values) to determine the type of a cluster. The most obvious and regular use of this matching occurs when identifying headers. Headers typically consist of keywords that occur at the start of a cluster. For example, an HyperText Markup Language (HTML) file always begins with the character sequences <HTML> or <html>. A cluster that has a lot of "href=" commands may also be considered to be an html cluster. While keyword matching/pattern matching is beneficial to identify potential file types, there is always the danger that a cluster not belonging to the file type has the exact same pattern. For example, an HTML tutorial written in Microsoft Word or Adobe portable document format (PDF) may contain the words "href=" describing how Web links are defined, and the cluster containing this part of the tutorial may be incorrectly identified as an HTML document. Therefore, additional tests need to be undertaken before verifying that a cluster belongs to a particular type.
- *American Standard Code for Information Interchange (ASCII)* character frequency, if very high in a cluster, can suggest that the cluster belongs to a file type that is probably not a video, audio, or image file type (though these file types can contain meta-data that is text based). This is not a very robust technique for identifying file types and will not distinguish between an HTML, text file, or Microsoft Word document containing text.
- *Entropy* for each disk cluster can be computed and used to identify the file type of the cluster, however, a lot of false positives will occur. In addition, it is typically hard to distinguish between the entropy of certain file types. For example, the entropy of a jpeg cluster can be very similar to the entropy of a compressed file using zip technology. As with ASII, the entropy alone should not be used for file type classification.
- *File fingerprints* were proposed by McDaniel and Heydari [11], and it is one of the first robust techniques for determining a file's type. They proposed three different algorithms that produced "fingerprints" of file types where unknown files were compared against one another. One of the algorithms used is called the byte frequency distribution (BFD) algorithm. The BFD for a file is generated by creating a byte histogram for the file (simply the number of occurrences of each of the unique 256-B values possible in the file). A BFD fingerprint is created by choosing multiple different files of a file type (for example choosing a few hundred different jpegs) and creating the BFD for each file and then averaging the BFD for all the files of the file type.

[DISK CLUSTER CLASSIFICATION IS IMPORTANT BECAUSE IT ALLOWS US TO REDUCE THE NUMBER OF CLUSTERS TO CONSIDER FOR EACH FILE BEING RECOVERED.]

In addition, for different files of a specific file type, certain byte values will have consistent frequency values, while other byte values will have fluctuating frequencies. For example, in an HTML file, the bytes representing the characters "\"," "<," and ">," will occur frequently. The byte values that have consistent frequency values are considered to be strong indicators of the file type, and this information is taken into consideration by calculating a "correlation strength" for each byte value. A BFD fingerprint for a particular file type then uses the byte frequency distribution and the correlation strength of each byte.

Going back to an HTML file, the bytes representing the characters "<" and ">" should occur with roughly the same frequency. As a result, the authors realized that a fingerprint consisting of the byte frequency cross-correlation (BFC) could also be created by storing the frequency between all byte pairs and then calculating a correlation strength similar to the BFD algorithm.

Fingerprints were created by analyzing and averaging the BFD and BFC of files belonging to many different file types. Unknown file types were compared to these fingerprints and the fingerprint with the closest match was selected as the filetype for the unknown file. Unfortunately, the accuracy of the two techniques were found to be about 30% and 45% each. Their third algorithm used header and footer identification techniques to also determine the file type and they were able to improve the accuracy to 95%, however, for disk cluster identification, this technique will not work as each cluster that does not contain header or footer information also needs to be analyzed and classified.

In research designed to detect anomalous payloads in network traffic, Wang and Stalfo [9] described creation of models of network traffic by using the BFDs and standard deviation of standard traffic and then comparing new network traffic against this model. In a subsequent article, Wang et al. [10] attempt to improve the detection of files by extending and enhancing the work done earlier. Their improved technique for creating "fileprints" creates a set of models for each file type instead of a single model. This improves the performance of the byte frequency analysis (BFA) algorithm [11] because a single model cannot accurately capture the representation of a file type. Intuitively, this makes sense, since a set of Word documents containing Shakespeare's works and another set containing French novels will probably have a very different BFD. The accuracy of their "fileprints" algorithms were much improved compared to the BFA algorithm. For some file types they were getting a 100% accuracy and the lowest accuracy occurred for jpegs at 77%. However, their results clearly indicated that the accuracy for determining a file decreased as the number of bytes in the file being analyzed was reduced. In the case of disk clusters, the average size of a cluster being no more than 4,096 bytes means that the accuracy will further degrade when utilizing this technique.

[FILE CARVING WAS BORN DUE TO THE PROBLEMS INHERENT WITH RECOVERY FROM FILE SYSTEM META-DATA ALONE.]

- *Classifications for File Carving:* Karresand et al. [13], [14] were the first to look at classifying individual data clusters and not entire files. They recognized the problem in terms of disk forensics and file carving, and proposed the Oscar method for identifying individual data clusters on the disk. The original Oscar method was similar to the "fileprint" algorithm in that it created a model for file types using the byte frequency distribution. The centroid model for a file type is created by measuring the mean and standard deviation of each byte value. Additional file specific keywords are also used to improve the accuracy of the method. The original Oscar method utilizing BFD and jpeg specific markers was able to achieve a 97% accuracy rate. Subsequently, the authors developed an enhanced Oscar method that used the rate of change (RoC) in the byte values as a basis for the centroid model. While all BFD models take into account the byte values, they do not take into account the ordering of the bytes. The RoC is defined as the absolute difference between two consecutive bytes in a data cluster. Using RoC as a basis for their centroid models, they were able to improve the accuracy rate of identifying jpeg disk clusters to 99%.

Finally, Veenman [15] proposed creating models for identifying disk clusters by utilizing a combination of BFDs, entropy, and the Kolmogorov complexity. The Kolmogorov complexity was added as it measures byte order. One of the interesting aspects of this research was intended to show which file types are frequently confused with others (jpeg and zip), and the need for further research in differentiating between the two.

REASSEMBLY

Reassembly is the final step of the SmartCarver. It involves finding the fragmentation point of each unrecovered file and then finding the next fragment's starting point (cluster). This process is repeated until a file is built or determined to be unrecoverable. Garfinkel [2] showed that files rarely fragment into more than three fragments. This implies that the majority of files have a large number of consecutive clusters even if fragmented. As mentioned in Table 1 a fragment consists of one or more disk clusters, and a file that is fragmented will have two or more fragments. The starting fragment of a file is defined as the base fragment and this will typically include the header.

The reassembly process involves finding the starting cluster of the fragment following the base fragment, and then using a technique to identify the ending point of the fragment. The fragmentation point of the fragment is defined as the last consecutive cluster in the fragment. For example, if fragment F contains clusters b_x, b_{x+1}, b_{x+2}, and b_{x+3}, then b_{x+3} is considered to be the fragmentation point of the fragment F, assuming that it is not the end of the file (a footer). A file containing k fragments will contain $k-1$ fragmentation points.

One early method for finding the fragmentation point was done by analyzing consecutive clusters based on their structure and/or the sequence of bytes that straddled the boundary of the two clusters.

KEYWORD/DICTIONARY

A simple technique for merging clusters is to use a list of keywords or a standard word dictionary to determine if a cluster should be merged. A dictionary-based merging occurs when a word is formed between the two cluster boundaries. For example, if cluster b_x ends with the sequence "he" and cluster b_{x+1} begins with "llo world" then the word "hello" is formed and the clusters are merged. Similarly, keyword merging occurs when cluster b_x's file type has been identified. If for example b_x was identified as belonging to a HTML document and it ends with "</cen" and cluster b_{x+1} starts with "ter>" then the HTML keyword </center> is formed and the cluster is merged.

This process is repeated until two consecutive clusters are found that do not share a keyword or dictionary word, thereby identifying the fragmentation point. The most obvious limitations of this method are that a lot of file formats do not have keywords or words that can be used to merge two clusters.

FILE STRUCTURE MERGING

For certain file types, knowledge of the file type structure can result in multiple clusters being merged. For example, png keywords are associated with a length field, indicating the length of the data associated with the keyword. After the data a 4-B cyclic redundancy check (CRC) value is stored. For example, lets say that cluster b_x is identified as png and has a png keyword that also indicates the length of the data, and the cluster containing the CRC of the associated data is k clusters away b_{x+k}. If the 4-B value at the indicated byte offset (as determined by the data length) is the same as the CRC value calculated by collation for the data in $b_x, b_{x+1} \dots b_{x+k}$, then all the clusters between b_x and b_{x+k} can be merged.

[A SIMPLE TECHNIQUE FOR MERGING CLUSTERS IS TO USE A LIST OF KEYWORDS OR A STANDARD WORD DICTIONARY TO DETERMINE IF A CLUSTER SHOULD BE MERGED.]

However, what if the CRC value is different, thus indicating a fragmentation point? Then the problem becomes that of identifying which one of the clusters from $b_x, b_{x+1} \dots b_{x+k}$ is the actual fragmentation point. In addition, file types like 24-b BMPs and pure text documents do not have formats that can used to merge consecutive blocks.

Recognizing the limitations of the above methods, Pal et al. modified the PUP algorithm to use sequential hypothesis testing as described by Wald [17]. The primary idea being that for every cluster added to the path, there is a high likelihood that the subsequent clusters belong to the path as well. The authors use sequential hypothesis to determine if consecutive clusters should be merged together, not only based on the file type structure but also on the content of individual files. This modified reassembly algorithm called sequential hypothesis-parallel unique path (SHT-PUP) is now described.

SHT-PUP

SHT-PUP is a modification of the PUP algorithm developed earlier. The reassembly algorithm is now described in greater detail.

Let there be k files to recover, then k file headers $(b_{h1}, b_{h2}, \dots b_{hk})$ are stored as the starting clusters in the reconstruction paths P_i for each of the k files. A set $S = (b_{s1}, b_{s2}, \dots, b_{sk})$ of current clusters is maintained for processing, where b_{si} is the current cluster for the ith file. Initially, all the k starting header clusters are stored as the current clusters for each file (i.e. $b_{si} = b_{\mathrm{hi}}$). The best greedy match for each of the k starting clusters is then found and stored in the set $T = (b_{t1}, b_{t2}, \dots, b_{tk})$ where b_{ti} represents the best match for b_{si}. From the set T of best matches the cluster with the overall best matching metric is chosen.

Assuming that this best cluster is b_{ti}, the following steps are undertaken:

1) Add b_{ti} to reconstruction path of ith file, (i.e. $P_i = P_i \| b_{ti}$).
2) Replace current cluster in set S for ith file (i.e. $b_{si} = b_{ti}$).
3) Sequentially analyze the clusters immediately after b_{ti} until fragmentation point b_{fi} is detected or file is built.
4) Replace current cluster in set S for ith file (i.e. $b_{si} = b_{fi}$).
5) Evaluate new set T of best matches for S.
6) Again find best cluster b_{ti} in T.
7) Repeat one until all files are built.

The enhancements to PUP are in Step 3 of the above algorithm. Step 3 will now be described in greater detail.

SEQUENTIAL HYPOTHESIS TESTING FOR FRAGMENTATION POINT DETECTION

In Pal et al's sequential fragmentation point detection method [5], the number of observations (weights) $W_1, W_2, W_3, \dots$, associated with a set of clusters, are not fixed in advance. Instead, they are evaluated sequentially and the test is ended in favor of a decision only when the resulting decision statistic is significantly low or high. Otherwise, if the statistic is in between these two bounds, the test is continued. Starting with the first data cluster of the base-fragment b_0, identified during collation, subsequent data clusters $b_1, b_2, \dots b_n$ are appended to b_0 and a weight conforming to, $W_1, W_2, \dots, W_n$ is obtained in sequence with each addition of a cluster. Accordingly, we define the hypotheses H_0 and H_1 as the following:

H_0: clusters $b_1, b_2, \dots, b_n$
belong in sequence to the fragment.
H_1: clusters $b_1, b_2, \dots, b_n$
do not belong in sequence to the fragment.

If the evaluated data clusters $b_1, b_2, \dots, b_x$ do not yield to a conclusive decision, the test continues with the inclusion of cluster b_{x+1} until one of the hypotheses is confirmed. When hypothesis H_0 is true, the evaluated clusters are merged to the base-fragment and a new test is started. Each time the test starts with a new data cluster in sequence, the weight is computed with respect to the recovered part of the base-fragment. The test procedure finalizes after one of the following conditions occur:

1) H_1 is achieved. (cluster does not belong to the fragment).
2) The file is completely recovered.
3) An error occurs because no data-cluster remains or remaining clusters are of a different file type.

Let W represent the weight sequence $W_1, W_2, \dots, W_n$, then in a sequential hypothesis test, a test statistic Λ is computed as the likelihood ratio of observing sequence W under the two hypotheses, which is expressed as the ratio of the conditional distributions of observed weights under H_0 and H_1 as

$$\Lambda(\mathrm{Y}) = \frac{Pr(\mathrm{W}|H_1)}{Pr(\mathrm{W}|H_0)}. \tag{3}$$

The actual probabilities are calculated by comparing the weight generated between two clusters against the probability models that are pregenerated. Finally, a decision is made by comparing Λ to appropriately selected thresholds as

$$\text{outcome of test} = \begin{cases} H_1, & \Lambda(\mathrm{W}) > \tau^+ \\ H_0, & \Lambda(W) < \tau^- \\ \text{inconclusive}, & \tau^- < \Lambda(W) < \tau^+. \end{cases} \tag{4}$$

WEAR LEVELING IS AN ALGORITHM BY WHICH THE CONTROLLER IN THE STORAGE DEVICE REMAPS LOGICAL BLOCK ADDRESSES TO DIFFERENT PHYSICAL BLOCK ADDRESSES.

That is, if the test statistic (likelihood ratio) is larger than τ^+, we assume that hypothesis H_1 is true and we have found the fragmentation region. If, on the other hand, test statistic is smaller than τ^-, hypothesis H_0 is true and all the fragments are merged and the test continues from the next sequential cluster. Finally, if neither case is true, testing continues until one of the thresholds is exceeded or end-of-file indicator is reached.

Ultimately, the success of the sequential fragment point detection method depends on two factors. The first factor is the choice of the weight whose design has to take into consideration different file types and to capture semantic or syntactic characteristics of the file. The second factor is the accurate determination of the conditional probability mass functions under the two hypotheses.

Pal et al. only provided results for jpeg reconstruction on the well-known test-sets of DFRWS 2006 [22] and DFRWS 2007 [23] and showed how they were able to recover files that no other known technique could recover even in the case of files fragmented into more than four pieces and with many different file types present. The major known issue with this technique is building accurate models to determine the thresholds and defining good weighting techniques to compare clusters from different file types. For images they weighted clusters by examining pixel boundary value differences as described in the section "Graph Theoretic Carvers," and for text they used PPM across a sliding window between two clusters. However, there is room for improvement in determining accurate weights that can be used to indicate the likelihood that two clusters belong together.

CONCLUSIONS

We have presented the evolution of file carving and described in detail the techniques that are now being used to recover files without using any file system meta-data information. We have shown the benefits and problems that exist with current techniques. In the future, SSDs will become much more prevalent. SSDs will incorporate wear-leveling, which results in files being moved around so as to not allow some clusters to be written to more than others. This is done because after a certain amount of writes a cluster will fail and, therefore, the SSD controller will attempt to spread the write load across all clusters in the disk. As a result, SSDs will be naturally fragmented, and should the disk controller fail the clusters on the disk will require file carving techniques to recover. There is a lot of research yet to be done in this area for data recovery. Finally, while Pal et. al's techniques are useful for recovering text and images, new weighting techniques need to be created for video, audio, executable and other file formats, thus allowing the recovery to extend to those formats.

AUTHORS

Anandabrata Pal (PashaPal@hotmail.com) is obtaining his Ph.D. degree in computer science at Polytechnic University, New York. He is specializing in file carving and data recovery. His research interests also include code obfuscation and file system forensics.

Nasir Memon (memon@poly.edu) is a professor in the Computer Science Department at Polytechnic University, New York. He is the director of the Information Systems and Internet Security lab at Polytechnic. His research interests include data compression, computer and network security, digital forensics, and multimedia data security.

REFERENCES

[1] *Pew Global Attitudes Project* [Online]. Available: http://pewglobal.org/reports/pdf/258.pdf

[2] S. Garfinkel, "Carving contiguous and fragmented files with fast object validation," in *Proc. 2007 Digital Forensics Research Workshop (DFRWS)*, Pittsburgh, PA, Aug. 2007, pp. 4S:2–12.

[3] A. Pal and N. Memon, "Automated reassembly of file fragmented images using greedy algorithms," *IEEE Trans. Image Processing*, vol. 15, no. 2, pp. 385–393, Feb. 2006.

[4] A. Pal, K. Shanmugasundaram, and N. Memon, "Reassembling image fragments," in *Proc. ICASSP,* Hong Kong, Apr. 2003, vol. 4, pp. IV–732-5.

[5] A. Pal, T. Sencar, and N. Memon, "Detecting file fragmentation point using sequential hypothesis testing," *Digit. Investig.*, to be published.

[6] G. G. Richard, III and V. Roussev, "Scalpel: A frugal, high performance file carver," in *Proc. 2005 Digital Forensics Research Workshop (DFRWS)*, New Orleans, LA, Aug. 2005.

[7] *Foremost 1.53* [Online]. Available: http://foremost.sourceforge.net

[8] K. Shanmugasundaram and N. Memon, "Automatic reassembly of document fragments via data compression," presented at the 2nd Digital Forensics Research Workshop, Syracuse, NY, July 2002.

[9] K. Wang, S. Stolfo, "Anomalous payload-based network intrusion detection," in *Recent Advances in Intrusion Detection*, (Lecture Notes in Computer Science), vol. 3224. New York: Springer-Verlag, 2004, pp. 203–222.

[10] W.J. Li, K. Wang, S. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *Proc. 6th IEEE Systems, Man and Cybernetics Information Assurance Workshop*, 2005, pp. 64–67.

[11] M. McDaniel and M. Heydari, "Content based file type detection algorithms," in *Proc. 36th Annu. Hawaii Int. Conf. System Sciences (HICSS'03)—Track 9*, IEEE Computer Society, Washington, D.C., 2003, p. 332.1

[12] B. Carrier, *File System Forensic Analysis*. Boston, MA: Pearson Education, Addison-Wesley Professional, 2005 .

[13] M. Karresand and N. Shahmehri, "Oscar file type identification of binary data in disk clusters and RAM pages," in *Proc . IFIP Security and Privacy in Dynamic Environments*, vol. 201, 2006, pp. 413–424.

[14] M. Karresand and N. Shahmehri, "File type identification of data fragments by their binary structure," in *Proc. IEEE Information Assurance Workshop*, June 2006, pp. 140–147.

[15] Cor J. Veenman, "Statistical disk cluster classification for file carving," in *Proc. IEEE 3rd Int. Symp. Information Assurance and Security*, Manchester, U.K., 2007, pp. 393–398.

[16] *STORAGEsearch.com. Data Recovery from Flash SSDs?* [Online]. Available: http://www.storagesearch.com/recovery.html

[17] A. Wald, *Sequential Analysis*. New York: Dover, 1947.

[18] Amiga Smart Filesystem [Online]. Available: http://www.xs4all.nl/ hjohn/SFS

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[20] L.W. McVoy and S.R. Kleiman, "Extent-like performance from a UNIX file system," in *Proc. USENIX, Winter '91*, Dallas, TX, 1991, pp. 33–43.

[21] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS file system," in *Proc. USENIX 1996 Annu. Tech. Conf.*, San Diego, CA, 1996, pp. 1–14.

[22] B. Carrier, V. Wietse, and C. Eoghan, *File Carving Challenge 2006* [Online]. Available: http://www.dfrws.org/2006/challenge

[23] B. Carrier, V. Wietse, and C. Eoghan, *File Carving Challenge 2007* [Online]. Available: http://www.dfrws.org/2007/challenge

[24] J. G. Cleary and W. J. Teahan, "Unbounded length context for ppm," *Comput. J.*, vol. 40, no. 2/3, pp. 67–75, 1997.