Resurrection: A Carver for Fragmented Files

Martin Lambertz(✉), Rafael Uetz, and Elmar Gerhards-Padilla

Fraunhofer FKIE, Friedrich-Ebert-Allee 144, 53113 Bonn, Germany
{martin.lambertz,rafael.uetz,elmar.gerhards-padilla}@fkie.fraunhofer.de

Abstract. The recovery of deleted files is an important task frequently carried out by professionals in digital forensics and data recovery. When carried out without information from the file system, this process is called file carving. The techniques implemented in today's file carvers are mostly sufficient for non-fragmented files. Fragmented files, on the contrary, are not well supported. In this paper we present a general process model for the recovery of fragmented files. This model is then applied to the JPEG file format which is the de facto standard for digital photographs. Moreover, we evaluate popular open source carvers and compare them with our proposed approach.

Keywords: File carving · Multimedia forensics · Fragmented files · JPEGs

1 Introduction

The discipline of digital forensics provides techniques to find evidence on digital devices. One task carried out to find digital evidence is the recovery of deleted data. When a file is deleted, the actual data making up this file is usually not removed from the storage device. Instead, only the corresponding entries in the file system metadata are removed or modified. This effectively marks the area that was occupied by the file as free. Therefore, deleted files are still present until overwritten by other data.

In most cases techniques used in traditional data recovery might suffice to restore the deleted files. These methods usually rely on information provided by the file system metadata. However, in cybercrime you often have a different situation. If no information from the file system is available, either because it is missing or corrupt, e.g. because a suspect tried to destroy evidence, those methods will most likely not be able to restore deleted files.

In such scenarios the files have to be restored by investigating their structure and contents rather than investigating the file system metadata. This process is called file carving or just carving. This gets even more difficult if the files are stored fragmented on a storage medium. Without file system information it is very hard to determine where the fragments of such a file begin and end. Thus, file carving can be a very time consuming process. Moreover, it possibly generates a large amount of data to be sifted by a forensic examiner. However, when dealing

P. Gladyshev et al. (Eds.): ICDF2C 2013, LNICST 132, pp. 51–66, 2014.
DOI: 10.1007/978-3-319-14289-0_5

with cybercrime, not all of this data is of interest during an investigation. For instance, files belonging to the operating system, when untampered with, are usually non-relevant. In most cases files generated by the user such as e-mails, text documents, pictures or videos are more important. On the one hand those files itself may prove or disprove a criminal act (e.g. when possession of a specific file is illicit). On the other hand the contents of a file may give important hints to the solution of a crime.

One type of user generated data are pictures in general and digital photographs in particular. Photographs can play central roles when investigating criminal acts. Consider cases of child pornography for example. Finding explicit photographs on the computer of a suspect may help the authorities to convict the suspect. Other crimes that possibly involve photographs are blackmailing, cyber-bullying, harassment, and falsification of documents.

The de facto standard algorithm for compressing digital photographs has been defined by the Joint Photographic Experts Group in [1] and is commonly known as the JPEG standard. Nowadays, most of the digital cameras produce JPEG-compressed images. This holds for low-end consumer cameras and cameras integrated into mobile phones as well as for high-end digital cameras.

Current file carvers are able to restore deleted files when they are stored contiguously on the storage medium. However, only very few carvers support the reconstruction of fragmented files and, if so, in a rudimentary manner only. To our knowledge the only product available with advanced support for fragmented images is Adroit Photo Forensics [2] developed by Digital Assembly. However, this product is proprietary and cannot be easily used by researchers for their own work.

Although modern file systems try to minimize fragmentation, it cannot always be avoided completely. In [3] Garfinkel presents detailed fragmentation statistics collected from more than 300 hard disks. An important conclusion that can be drawn from these statistics is that files of interest during a forensic investigation tend to fragment more likely than less interesting files. For instance, 16 % of the JPEG files were fragmented, 20 % of the AVI files, and 58 % of the PST files. These numbers may not sound very large, however, we think that they are too large for not considering fragmented files at all.

The rest of this paper is organized as follows: Sect. 2 presents related work and Sect. 3 formulates requirements for modern file carvers. In Sect. 4 we present our approach for carving fragmented JPEG files. Finally, in Sect. 5 we compare our carving approach with popular open source carvers and conclude this paper in Sect. 6.

A Note on Terminology. Throughout the rest of this paper we use the term *block* to denote the smallest unit of data that is processed by a file carver. This may be the smallest addressable unit of a storage medium such as a sector of a hard disk drive or a cluster of the file system.

2 Related Work

There exist various file carving approaches. In this section we introduce selected proposed techniques which our work is partially based on. Starting with approaches that do not consider fragmented files, we present fundamental works addressing fragmented files as well.

One of the most simple carving approaches is called header-to-footer carving. Using this technique a carver searches a given disk image for occurrences of certain byte sequences indicating the beginning (header) and end (footer) of a file. Afterwards the data between those signatures is restored. While being very fast, this approach is limited to contiguously stored files only. Nevertheless, it is one of the most widely used strategies in current file carvers such as Foremost [15] and Scalpel [16].

Motivated by his study on fragmented files, Garfinkel was one of the first to address the recovery of fragmented files. In [3] he introduces a technique called bifragment gap carving. This approach makes use of object validation, which means that a file candidate is validated before it is actually recovered. Depending on the file type a validation includes a verification of headers and footers, a verification of the file structure, and, if applicable, the decoding of the file. In case of a successful validation the file is recovered; otherwise, the file is deemed fragmented. Fragmented files are recovered by successively removing data between a found header and footer of a file and afterwards trying to validate the file. Starting with a gap size of one block, all possible positions of this gap are tested. If the file still does not validate, the gap size is increased by one block and again all possible positions of the gap are tested. This process is carried out for all gap sizes until the file is successfully validated or all combinations are exhausted. As the name implies this approach is limited to files split into two fragments; files split into more fragments are not recoverable.

In [4,5] a general process model for the reconstruction of fragmented files is proposed consisting of three steps: preprocessing, collation, and reassembly. In the preprocessing step encryption or compression possibly applied to the disk image is removed. The collation step is responsible for classifying the blocks of the disk image as belonging to one or more file types. Finally, in the reassembly step the files are reconstructed. Depending on the file type different reconstruction algorithms have to be applied in this step.

Moreover, the authors formulate the reassembly of fragmented files as a graph-theoretic problem where the blocks of a disk image form the set of nodes of a graph. The edges indicate the probability that two blocks are adjacent in the original file. The authors use different algorithms to approximate optimal paths within the resulting graph. This idea was subsequently refined in [6,7]. A problem with graph-theoretic carving is scalability. For large storage media the number of blocks to consider may exceed current computing and memory capabilities.

3 Requirements for File Carvers

The ever increasing capacities of today's storage media present a formidable challenge to file carving tools. Modern hard disk drives are capable of storing terabytes of data and even USB flash drives and mobile phones have capacities of several gigabytes. In order to be able to keep up with this trend a file carver should behave efficiently in terms of processing speed as well as in memory requirements.

Considering that a file carver is often part of a digital forensics investigation an even more important requirement is a correct and traceable functioning. If this cannot be granted, results produced by the file carver might not be usable as evidence in court.

Based on the aforementioned considerations we formulated three basic requirements for file carving tools. Moreover, we prioritized the requirements by their importance during a forensics investigation.

1. Correctness: A carver should be able to correctly recover as many files as possible. Moreover, it should not generate corrupted files that cannot be opened using a standard program for the corresponding file types.
2. Scalability: A carver should be able to handle today's and future high-capacity storage volumes.
3. Performance: A carver should be reasonably fast.

4 Fragmented JPEG Carving

Based on the general process model introduced in [4] and the requirements presented in the previous section we derived the file carving process model depicted in Fig. 1. We varied from the original process model because of the fragmentation statistics in [3] which imply that a large fraction of the files on a storage medium are stored contiguously. Such files can be restored comparatively easily and fast. Therefore, we restore the non-fragmented files first. On the one hand this approach can drastically reduce the amount of data to be considered in the more complex reconstruction of fragmented files. On the other hand there will already be a large number of recovered files for an examiner to sift.

4.1 Preprocessing Phase

As in [4], the preprocessing phase is responsible for converting the input into a format the file carver is able to process. This means that any encryption or compression has to be removed so that the following steps can operate on raw data. Moreover, if the storage medium under investigation contains a valid file system, the information stored in the file system metadata may be used to reduce the overall data the carver has to consider. This, however, requires the investigator to trust this information, which cannot always be assumed.

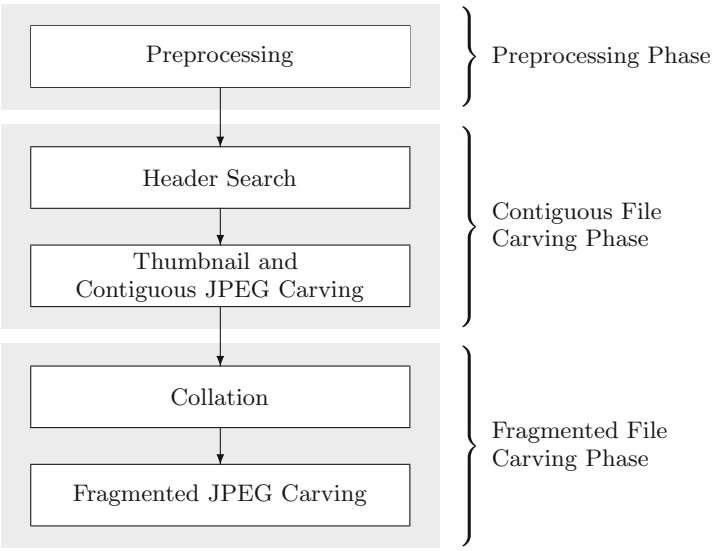

Fig. 1. Proposed carving model.

4.2 Contiguous File Carving Phase

As mentioned earlier, the first carving phase is used to restore non-fragmented files. Moreover, it is responsible for generating candidates for the reconstruction.

Header Search. In this step the storage medium is searched for occurrences of strings indicating the start of a JPEG file, thus generating candidates of files to be carved. Note that the results of this step will be used in all of the remaining carving steps, not only in this phase but also in the fragmented file carving phase. That is, only candidates generated in this step will be considered during the following carving processes.

Table 1. Byte sequences identifying JPEG headers. The question marks (?) represent wildcard characters representing one byte value each.

Header type	Byte sequence
JPEG/JFIF	`0xFF 0xD8 0xFF 0xE0 ? ? 0x4A 0x46 0x49 0x46 0x00`
JPEG/Exif	`0xFF 0xD8 0xFF 0xE1 ? ? 0x45 0x78 0x69 0x66 0x00`

Table 1 lists the byte sequences identifying the JPEG/JFIF and JPEG/Exif file formats. Whenever such a byte sequence is found, its starting position is stored in a sequence H which represents the candidates to be carved. We focus on these two formats here, as they are the predominant file formats for storing

JPEG-compressed files. However, support for further JPEG or even completely different file types may be added very easily.

Thumbnail and Contiguous JPEG Carving. We differentiate between regular JPEG files and thumbnails. Many JPEG file formats store a down-scaled version of the original image in the JPEG file. Moreover, a lot of digital cameras generate and store these thumbnails automatically in the photographs.

Thumbnails are interesting for several reasons. For one thing the forensic examiner can use them to get a rough overview of what images to expect. Depending on the contents of the thumbnail images the examiner may be in a better position to judge if more time-consuming techniques are worth being applied. For another thing the thumbnails themselves may already contain compromising material that can be used in the course of the investigation.

Thumbnails may appear in certain segments of a JPEG file only, located within a limited range after the header signature. We scan this area for possibly existent thumbnails and whenever a thumbnail is found, it is immediately carved.

After all thumbnails are restored, we carve the contiguously stored JPEG files. Based on Garfinkel's statistics provided in [3] these files will make up about 84 % of the JPEGs. Therefore, it is worthwhile to use appropriate carving strategies. That is, carving strategies that are simple and, more importantly, fast.

The obvious carving strategy would be header-to-footer carving. However, since correctness is our primary requirement, we have to make sure that only valid files are carved during this step. Header-to-footer carving alone provides no means to ensure the validity of the carved files. Hence, a second step to validate the files would be necessary.

In order to avoid having to first carve and afterward validate a file, we chose not to use the classical header-to-footer carving. Instead, we start decoding the JPEG from the header on until the JPEG is either completely decoded or until the decoder detects errors, such as an invalid file structure, invalid Huffman symbols or a premature end of the file for instance. Only files that are decodable without errors are finally carved and their headers are removed from the sequence H. Moreover, blocks that have been used to carve the non-fragmented files are marked as used and are not considered in the following carving phase anymore.

4.3 Fragmented File Carving Phase

In the second carving phase we try to restore the remaining JPEG files in set H, i.e. files that have not been restored in the first phase. At this point H contains candidates only that could not be carved using sequential decoding.

Collation. In the original process model the collation step is used to classify a given block as belonging to one or more file types. Especially for encoded data this multiclass classification problem is very hard and, in fact, not required at all. Since we focus on the recovery of JPEG files, we do not classify the blocks. Instead, in our case it is sufficient to determine if a data block belongs to a

JPEG file or not. This reformulation turns the multiclass classification problem into a binary classification problem, allowing the usage of more specialized methods, which are easier to compute and more accurate than general classification techniques most of the time.

Our preliminary assumption is that every block belongs to a JPEG file. Then, we subsequently perform tests on each block that may refute this assumption. As soon as one of these tests yields a negative result, we discard the block and do not consider it any further.

The first test exploits what is called byte stuffing in [1]. In a JPEG file a `0xFF` byte introduces a marker. In order to distinguish markers from `0xFF` bytes occurring in the encoded image data, each `0xFF` that does not introduce a marker has to be masked with a `0x00` byte. Blocks without correct byte stuffing can easily be identified and do not have to be considered anymore.

In the next test we determine the frequency of printable ASCII characters contained in a block. We count the number of bytes from the interval [`0x20`,`0x7E`] as well as the bytes `0x09` (horizontal tab), `0x0A` (line feed), and `0x0D` (carriage return). If the percentage of these characters exceeds a certain threshold, then we assume the block to belong to some kind of text file rather than to a JPEG. We evaluated various thresholds and found 70 % to yield the best results which are presented in Table 3.

Finally, we check whether a block consists of `0x00` bytes only. If this applies, then we assume that the block has not been used to store any data yet or has been sanitized. Such blocks are discarded as well.

A block that passes all of the tests outlined above will be inserted into a set D. Along with the headers H, only these blocks are considered in the reassembly of fragmented JPEGs following the collation step. Therefore, it is important not to separate out blocks that actually belong to JPEG files.

Note that we considered implementing more tests in this step, for instance by using the information entropy or byte frequency distribution of a block. However, based on [8], where the authors assessed the probability of being able to distinguish a non-JPEG block from a JPEG block by checking the correctness of the byte stuffing, we know that with increasing block size this test alone is sufficient to eliminate more than 90 % of the non-JPEG blocks. For a block size of 4096 bytes, which is a common block size in many file systems, the probability to find an invalid byte sequence is even larger than 99 %. Therefore, we chose to omit further test, especially because tests based on entropy and byte frequency distribution are computationally more expensive.

Fragmented JPEG Carving. At this point we have completely classified every block of the disk image that has not yet been used to restore a JPEG. That is, each of the available blocks has either been included in H or D or has been determined not to belong to a JPEG file. Based on the blocks in the aforementioned sets the reassembly of fragmented JPEGs is performed in this step.

In order for the carving algorithm to be functioning, an essential assumption has to hold: the marker segments and the first few scanlines of the JPEG to

be recovered have to be stored contiguously in the first fragment of the file. This is because our approach does not yet include mechanisms to reassemble fragmented marker segments and the first scanlines are required as a starting point for the following reassembly steps. Although this requirement may sound restricting, we believe that it is not in realistic scenarios. Since file systems try to avoid fragmentation, it is unlikely that fragmentation occurs that early in a file already.

The following pseudocode lists the basic steps of the recovery of a fragmented JPEG file:

```
FUNCTION carve-fragmented-jpeg(h)
    B, b_a := initialize carving process
    WHILE JPEG is recoverable and JPEG is not complete DO
        b_safe := FastForwardStep(b_a)
        b_z := FragmentationPointDetection(b_safe)
        B := (B, b_a, b_a+1,..., b_z)
        b_a := FindNextFragment(b_z)
    END WHILE
END FUNCTION
```

Given a header $h \in H$, the carving process is initialized by reading the metadata (i.e. the marker segments) of the JPEG. This provides us with important information such as the number of pixels in vertical and horizontal direction and the number of color components. These values are essential for the further reconstruction steps. Moreover, the sequence of blocks B, which denotes the data blocks of a JPEG under reconstruction, is initialized with the blocks used to read the marker segments. The block b_a denotes the last block that has been used so far. This block serves as a starting point for the reassembly algorithm and is added to B not until later in the reassembly.

Decoding data not belonging to a JPEG typically causes corrupted image regions. These corruptions can visually be easily distinguished from valid image regions as they form sharp edges to the valid image regions in most cases. Thus, corrupted regions may be detected using edge detection algorithms known from the field of computer vision. In the *fast forward* step we try to find the maximum number of complete scanlines of an image that are not corrupted. We denote these scanlines as *safe scanlines*.

Given the first block of a fragment, b_a, we start to decode the image line by line. In order to detect image corruptions we use a simple edge detection technique similar to the one proposed by Cohen in [9]. This technique is based on Laplacian zero-crossings and is a well known approach in the field of computer vision. For every decoded scanline we compute an edge value according to Eq. 1:

$$e(y) = \frac{1}{X \cdot C} \cdot \sum_{x=0}^{X-1} \sum_{c=0}^{C-1} |I_c(x, y-1) - 2 \cdot I_c(x, y) + I_c(x, y+1)| \; . \qquad (1)$$

Here, X denotes the number of pixels in horizontal direction and C denotes the number of components per pixel. A grayscale image has only one component, an RGB image three. $I_c(x, y)$ denotes the cth component of pixel x in scanline y.

The edge value is subsequently compared to a dynamically computed threshold Θ. Such a threshold has to adapt to changing image characteristics without too much delay. Hence, we chose to compute the threshold based on a linear weighted average which on the one hand reacts faster to changing image characteristics than a median for example and on the other hand assigns higher weights to more currently processed scanlines.

A JPEG-compressed image is decoded in terms of blocks called minumum coded units (MCUs) of certain dimensions. Therefore, we do not have to check every image line for corruptions. Instead, we only examine the scanlines on the horizontal MCU block boundaries. Consider for example an MCU size of 8 x 8 pixels. Then we would check for image corruptions every 8 scanlines only.

The threshold Θ is computed by

$$\begin{aligned} \Theta_0 &= \alpha \\ \Theta_i &= 0.5 \cdot (\alpha + \mu) + \beta \cdot \Theta_{i-1} \end{aligned} \tag{2}$$

with α and β being predefined constants. μ is the linear weighted average of the last n edge values values computed by

$$\mu = \frac{2}{n \cdot (n+1)} \cdot \sum_{j=1}^{n} j \cdot s_j \tag{3}$$

where n denotes the number of edge values and s the ordered sequence of edge values computed in the current instance of the fast forward step. That means, $s = s_1, s_2, \ldots, s_n = e(y_1), e(y_2), \ldots, e(y_n)$, where y_1 is the first scanline read in this fast forward step and y_n the scanline just before the computation of μ.

Note that the weighted average and the threshold are computed only before the threshold is used in a comparison with an edge value. That is, the computations are carried out if the current scanline processed is at the boundary of two MCU blocks. Moreover, Θ and μ are only valid for one instance of the fast forward step. If the fast forward step is entered in the next round of the reconstruction algorithm, the computation of the two values starts over again.

Besides detecting image corruptions, the fast forward step serves another purpose. We do not only compute the linear weighted average, but also a long-term model of the image consisting of the arithmetic mean of the edge values and their standard deviation. This model is used in the following steps of the reassembly algorithm.

The fast forward step only determines the approximate region of the fragmentation point. The exact fragmentation point is identified in the *fragmentation point detection* step. Here, we try to find the last block belonging to the current file fragment.

The fragmentation point detection receives the last safe block, b_{safe}, as input which serves as a starting point for this step. In order to find the exact fragmentation point, we append the blocks following b_{safe} one by one and check

whether an image corruption is detected. The detection of a corruption is again performed by using formula 1 and a threshold θ computed by

$$\theta = \kappa \cdot M + \lambda \cdot \sigma \ . \tag{4}$$

In formula 4, M is the arithmetic mean and σ the standard deviation of the long-term model computed during the fast forward step. κ and λ are two predefined constants.

If the edge value computed exceeds the threshold θ, an image corruption is assumed and the block causing this corruption will be considered as the first block not belonging to the current file fragment anymore.

Note that we only append blocks which are in D in this step. When a block following b_{safe} is not in D, we assume a fragmentation point to be detected. This is why we need a de facto perfect sensitivity of the block classifier used in the collation step.

After the last block of the current fragment, b_z, has been identified, we have to find the first block of the next fragment belonging to the file under reconstruction. This is accomplished in the *find next fragment* step.

Based on the fragmentation statistics provided in [3], we know that typically file fragments are not scattered randomly across the storage medium, but are stored relatively closely to each other. This suggests that most of the time the beginning of the next fragment can be found within a close range after the block b_z. In [3] Garfinkel lists the gap distribution for JPEG files split into two fragments with the largest common gap size observed being 636 KiB. However, we do not want to limit the search for the next fragment to a predefined range in order to be able to carve files in more complicated fragmentation scenarios, too. Therefore, this step is further subdivided. First, we check the blocks within a predefined range after b_z. If we already find a good candidate within this range, we stop here and return this block. If we do not find such a candidate, we exhaustively search the remaining blocks in D for a candidate.

To grade a candidate block, we append the block to the blocks already considered as belonging to the JPEG and decode a complete set of scanlines. Note that we might have to read further blocks in order to completely decode these scanlines. After the decoding, once again we use formula 1 to compute an edge value between the scanlines that have been determined to belong to the file earlier and the newly decoded ones. The grade of a block is then calculated by

$$\omega = \begin{cases} 1, & \text{if } e(y_{b_i}) = 0 \\ 1/e(y_{b_i}), & \text{else} \end{cases} \tag{5}$$

where y_{b_i} is the first of the newly decoded scanlines after block b_i has been appended.

We compute the value ω for all blocks within a predefined range of 2500 blocks. If the best weight is larger than a threshold ϕ, the candidate is considered as good and this block will be returned as the beginning of the new fragment. If no block yields a weight larger than the threshold, we exhaustively check all

remaining blocks in D. We start right after the predefined range and proceed until the end of the disk image. Afterwards we start from the beginning of the disk image until we reach the header of the current file. Finally, we return the block with the best ω as the beginning of the next fragment and start another round of the algorithm until the JPEG is completely recovered or we detect that it is not recoverable at all.

5 Evaluation

To evaluate our proposed carving approach we implemented a prototypic carver and compared its capabilities with available open source carvers. The evaluation was performed on a machine with an Intel Core i7-3930K hexa-core CPU clocked at 3.2 GHz with 12 logical cores.

Table 2. Summary of the test sets.

Test set	Size in MiB	Non-fragmented JPEGs	Fragmented JPEGs	Thumbnails
Simple#1	≈50	2	0	3
Simple#2	≈50	1	1	3
Simple#3	≈50	0	2	3
Simple#1-notn	≈50	2	0	0
Simple#2-notn	≈50	1	1	0
Simple#3-notn	≈50	0	2	0
DFTT#8	≈10	6	0	0
DFTT#11	≈62	3	0	3
DFTT#12	≈124	1	2	1
DFRWS-2006	≈48	7	7	5
DFRWS-2007	≈331	1	13	13
nps-2009-canon2-gen6	≈32	30	6	38

Table 2 lists the test sets we used in our evaluation. The Simple#n test sets have been created by us to evaluate basic carving capabilities of the carvers. They consist of 50 MiB of random data and two JPEG files implementing fragmentation scenarios of different levels of difficulty. The Simple#n-notn test sets are the same as the Simple#n test sets but with the thumbnails removed from the JPEGs. The remaining test sets are publicly available standard test sets for file carvers frequently used in the literature. The DFTT test sets have been created by Carrier and Mikus specifically for the evaluation of file carvers. The test sets themselves and detailed descriptions are available at [10]. The DFRWS test sets were taken from the DFRWS challenges from the years 2006 and 2007

which were dedicated to file carving with a focus on fragmented files. Again, the test sets and detailed descriptions are publicly available [11,12]. Finally, we used the nps-2009-canon2-gen6 test set which is the image of a memory card used in a digital camera [13,14].

We used the test sets of the DFRWS challenges to evaluate our tests to differentiate between JPEG and non-JPEG blocks. Both test sets were processed with a block size of 512 bytes. The classification results with regard to sensitivity and specificity are presented in Table 3. Sensitivity and specificity are two standard metrics when measuring the quality of a classifier. In our case the former denotes the fraction of JPEG blocks which have been correctly classified as such and the latter denotes the fraction of non-JPEG blocks which have not been classified as JPEG blocks.

Table 3. Classification results for the DFRWS test sets.

		DFRWS'06	DFRWS'07
JPEG headers	Sensitivity	1.00	1.00
	Specificity	1.00	1.00
JPEG data blocks	Sensitivity	1.00	1.00
	Specificity	0.82	0.86

The first thing standing out is the perfect sensitivity of our classifier. For JPEG header blocks the specificity is perfect as well. For JPEG data blocks the specificity is notably lower. In the DFRWS-2006 test set the false positive rate is 18 %, in the DFRWS-2007 test set 14 %. These results correspond to the results of [8] mentioned in Sect. 4.3. Hence, we can expect the specificity of our classifier to become better with larger block sizes.

We evaluated various different combinations of the constant values α, β, κ, λ, and ϕ. We found that the carver achieved the best results with the values set to $\alpha = 1000$, $\beta = 0.32$, $\kappa = 1.9$, $\lambda = 4$, and $\phi = 0.001$. Therefore, we present the results for these values in our evaluation.

As already mentioned, we did not only evaluate our carver but also popular open source carvers. We included Foremost [15], Scalpel [16], and PhotoRec [17] in our evaluation each in at least two different configurations.

Figure 2 presents the results of the carvers with regard to the number of reconstructed files. The plot is tripartite: the top part shows the ratio of carved non-fragmented JPEGs, the part in the middle the ratio of carved fragmented JPEGs, and the bottom part the ratio of carved thumbnails. Each bar is divided into correctly carved files, partially carved files, and files not carved at all. In our evaluation a file is rated as correctly carved if it completely corresponds to the original file. A file is graded as partially carved if the file is not complete but the subject depicted is still identifiable. Finally, files are rated as not carved if the file is either missing completely or the subject is not identifiable.

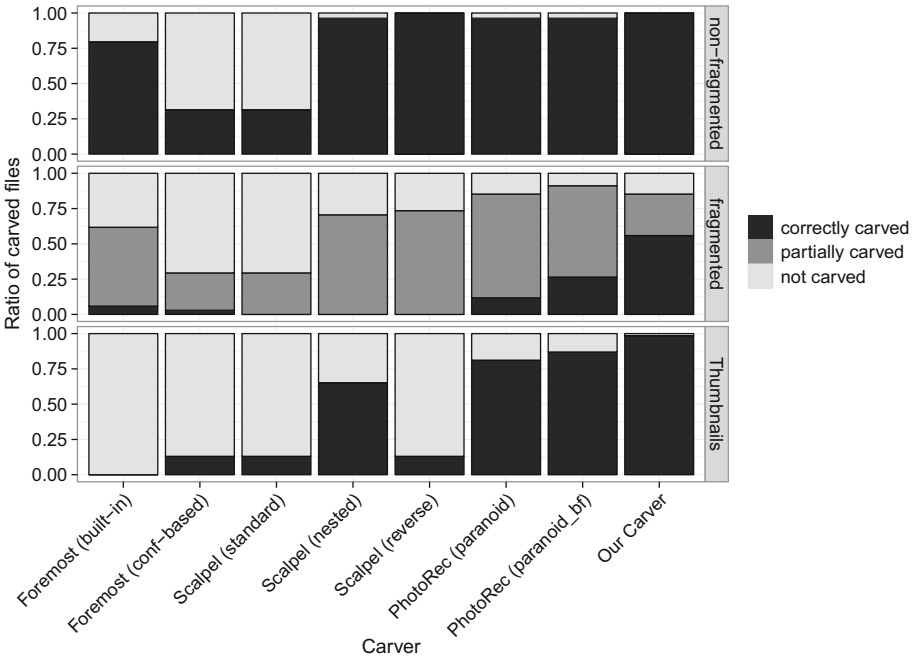

Fig. 2. Correctness of the file carvers.

Looking at the top part reveals that two configurations of Scalpel, PhotoRec, and our carver achieve very good reconstruction rates of more than 96 %, with our carver and one configuration of Scalpel being able to restore all files correctly.

When it comes to fragmented JPEGs, the number of correctly carved files significantly drops for all carvers evaluated. Especially Foremost and Scalpel, which have no dedicated support for carving fragmented files, are not able to completely recover a single fragmented JPEG. An exception is Foremost (built-in), which implements a heuristic to restore certain fragmented files from ext2/ext3 file systems [15].

PhotoRec implements basic mechanisms to carve fragmented files which is also reflected in our evaluation. Depending on the carving mode, PhotoRec is able to correctly restore about 12 % and 26 % respectively.

Finally, our carver is able to recover approximately 56 % of the fragmented JPEGs correctly which clearly outperforms the other carvers tested.

If we include the partially carved files, PhotoRec achieves slightly better results than our carver. The heuristic for carving fragmented files implemented by PhotoRec is comparatively simple and of minor computational complexity. Hence, it might be beneficial to adapt it to our carver as well.

Finally, the lower part shows that our carver is capable of restoring more thumbnails than every other carver tested. Only one thumbnail is not recovered because the header of the actual JPEG had already been overwritten. PhotoRec also differentiates between thumbnails and regular JPEGs and consequently

achieves good results as well. The other carvers do not make this differentiation and do not perform as well as our carver or PhotoRec.

Moreover, thumbnails were often the cause for corrupted files. Figure 3 depicts the absolute number of corrupted files generated by the carvers. Scalpel and Foremost (conf-based) generate a large amount of corrupted files mainly caused by JPEGs containing thumbnails. Foremost using the built-in carving heuristics, PhotoRec and our carver, on the contrary, do not generate such files. This is because all of these carvers perform some kind of validation during the carving process.

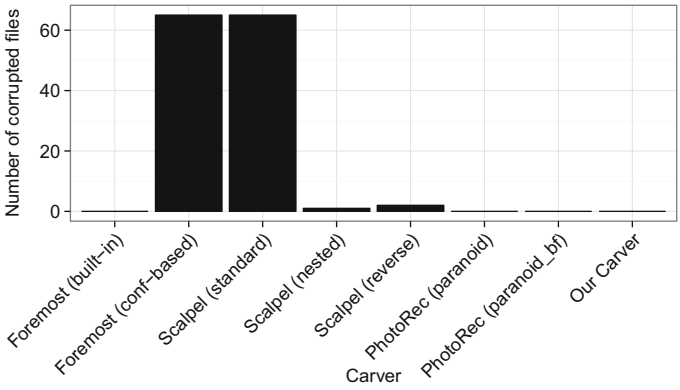

Fig. 3. Corrupted files generated by the file carvers.

Figure 4 illustrates the time the carvers took to process the individual test sets. The data points represent the arithmetic mean of 50 replications. The corresponding standard deviation is also plotted, but hardly visible due to virtually no variance in the results. In order to keep the plot clear, we only present a subset of the carvers, which generated the best results in terms of correctness.

The x-axis depicts the test sets sorted ascending by their size. The y-axis depicts the time the carvers took to process the test sets in seconds. Please note that this axis is scaled logarithmically in order to render all results visible.

The first thing to observe is that Foremost and Scalpel take less than two seconds for each of the test sets. Moreover, the time these two carver take seems to depend mainly on the size of the given input data.

PhotoRec exhibits a different behavior. Here, the input size is not the main factor influencing the runtime but rather the number of JPEGs and the complexity of the fragmentation scenarios contained in the test sets. In the paranoid mode, PhotoRec is nearly as fast as Foremost and Scalpel. All test sets have been processed in less than five seconds. The paranoid_bf mode is able to process half of the test sets in less than one second. The test sets DFRWS-2006 and nps-2009-canon2-gen6 took less than one minute. However, the time required for the test sets Simple#3 (>2.5 h), Simple#3-notn (12 min.), and DFRWS-2007 (25 min.) are significantly higher. Another thing to notice is the missing result

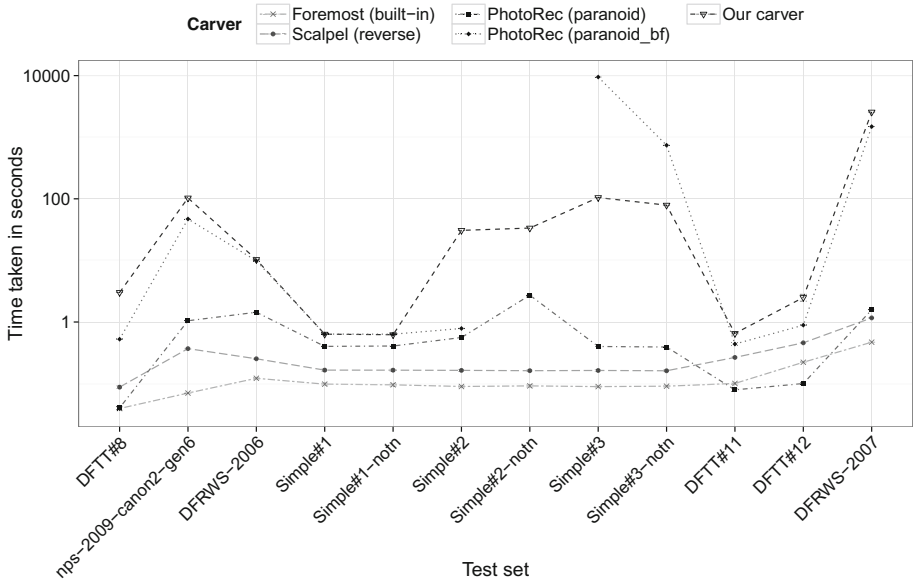

Fig. 4. Runtimes of the file carvers.

for PhotoRec (paranoid_bf) for the Simple#2-notn test set. Here, we stopped the execution after a time frame of more than 40 hours.

The trends of our carver is comparable to the execution times of PhotoRec. Again, the input size does not impact the runtime as much as the complexity of the fragmentation scenarios. Our carver is able to process all but the DFRWS-2007 test set in less than two minutes, some of them in less than one second. The DFRWS-2007 test set takes about 42 min to complete.

6 Summary and Future Work

In this paper we presented a carving approach for fragmented JPEG files. After formulating three key requirements for modern file carvers, we subsequently derived a process model for such a carver. Although we focused on fragmented JPEGs here, the general approach is suitable for different file types as well.

Afterwards we presented an algorithm to recover fragmented JPEG files following our process model. The evaluation revealed that our approach is capable of carving more files correctly than popular open source carvers, while still retaining good runtimes.

We are currently investigating techniques to further increase the correctness of our approach. For instance, exploiting features of the encoded image data might lead to better detection of corruptions and might as well increase the performance of our carver. Furthermore, a reconstruction of JPEG files which have no valid header anymore is on our agenda.

Moreover, we are evaluating the impact of parallelizing our carver. A parallelized prototype implementing our carving approach already yields promising results gaining a speedup of a factor greater than seven. In the course of this, we also consider modern graphics cards which allow massively parallel processing.

Finally, we are planning to extend our approach to further file types. Besides more image formats, the reconstruction of fragmented videos, PST files, and SQLite databases would be valuable for instance.

References

1. CCITT Recommendation T.81: Information technology - Digital compression and coding of continuous-tone still images - Requirements and guidelines. ISO/IEC 10918–1 (1992)
2. Digital Assembly - A smart choice for photo forensics. http://digital-assembly.com/products/adroit-photo-forensics/
3. Garfinkel, S.L.: Carving contiguous and fragmented files with fast object validation. Digital Inv. **4S**, 2–12 (2007)
4. Shanmugasundaram, K., Memon, N.: Automatic reassembly of document fragments via context based statistical models. In: 19th Annual Computer Security Applications Conference, pp. 152–159. IEEE Computer Society, Las Vegas (2003)
5. Pal, A., Memon, N.: The evolution of file carving. IEEE Signal Process. Mag. **26**(2), 59–71 (2009). IEEE
6. Memon, N., Pal, A.: Automated reassembly of file fragmented images using greedy algorithms. IEEE Trans. Image Process. **15**(2), 385–393 (2006)
7. Pal, A., Sencar, H.T., Memon, N.: Detecting file fragmentation point using sequential hypothesis testing. Digital Inv. **5**, 2–13 (2008)
8. Roussev, V., Garfinkel, S.L.: File classification fragment - the case for specialized approaches. In: Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, pp. 3–14. IEEE Computer Society (2009)
9. Cohen M.I.: Advanced JPEG carving. In: 1st International ICST Conference on Forensic Applications and Techniques in Telecommunications, Information and Multimedia (2008)
10. Digital Forensics Tool Testing Images. http://dftt.sourceforge.net
11. DFRWS 2006 Forensics Challenge. http://www.dfrws.org/2006/challenge/
12. DFRWS 2007 Forensics Challenge. http://www.dfrws.org/2007/challenge/
13. Digital Corpora: Disk Images. http://digitalcorpora.org/corpora/disk-images
14. Garfinkel, S.L., Farrell, P., Roussev, V., Dinolt, G.: Bringing science to digital forensics with standardized forensic corpora. Digital Inv. **6**, S2–S11 (2009)
15. Mikus, N.: An Analysis of Disc Carving Techniques. Master's thesis, Naval Postgraduate School, Monterey, California (2005)
16. Richard III, G.G., Roussev, V.: Scalpel: a frugal, high performance file carver. In: 2005 Digital Forensics Research Workshop (2005)
17. PhotoRec - CGSecurity. http://www.cgsecurity.org/wiki/PhotoRec