Forensic Science International: Digital Investigation 38 (2021) 301266

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

Timestamp prefix carving for filesystem metadata extraction

Kyle Porter [a, *], Rune Nordvik [a, b], Fergus Toolan [b], Stefan Axelsson [a, c]

[a] Norwegian University of Science and Technology, Norway
[b] Norwegian Police University College, Norway
[c] DSV, Stockholm University, Sweden

ARTICLE INFO

Article history:
Received 3 October 2020
Received in revised form
19 July 2021
Accepted 22 July 2021
Available online 7 August 2021

Keywords:
Digital forensics
Carving
Metadata
Filesystems

ABSTRACT

While file carving is a popular and effective method for extracting file content from unallocated space in a forensic image, it can be time consuming to carve for the wide variety of possible file signatures. Furthermore, file carving does not connect the discovered file to its filesystem metadata. These limitations of file carving are the advantages of *Generic Metadata Time Carving*, in which filesystem metadata is searched for by first finding repeated co-located timestamps using a potential timestamp carving algorithm. The potential metadata is verified by a filesystem specific parser, and the pointer within the metadata to the file data may allow for full file recovery. Currently, a limitation of the Generic Metadata Time Carving method is that it will only find metadata records that have multiple equivalent timestamps, thus missing metadata records and files with differing, but very similar, timestamps. Therefore, in order to improve the recall of the Generic Metadata Time Carving methodology, we have designed and implemented a prefix matching potential timestamp carving algorithm. We apply our experiments to realistic NTFS and Ext4 forensic images, in which we compare the precision and recall results for differing prefix lengths. Our results indicate that using prefix-based potential timestamp carving can yield significantly greater recall for extracting filesystem metadata records, with little to no reduction in precision as compared to the original exact potential timestamp carving method.

1. Introduction

File carving is an established digital forensics method for extracting files that cannot be found using the filesystem, and while extremely useful it is not without its faults. When applying popular file carving tools such as Scalpel (Richard and Roussev, 2005), one must attempt to search for all possible file signatures that are relevant to the case, which not only makes the search process more time consuming,[1] but more importantly, the file signature database may be incomplete. Omitted file signatures means missing files when carving.[2] File carving also does not have an automated method for connecting filesystem metadata to the discovered file (assuming the metadata still exists on disk) (Dewald and Seufert, 2017; Nordvik et al., 2019), and has difficulty dealing with fragmentation (Garfinkel, 2007).

These limitations of file carving are the advantages of *Generic Metadata Time Carving* (GMTC) by Nordvik et al. (2020). GMTC is a metadata carving method that uses a simple string matching algorithm, a *potential timestamp carver*, to search for equivalent and closely co-located byte sequences in order to find potential filesystem metadata record timestamps. After the locations of the potential timestamps are found, a filesystem specific parser either accepts or rejects the surrounding content as filesystem metadata. The filesystem metadata record may then be used to retrieve the associated file whether or not the file is fragmented, where the ability to do so is dependent on the filesystem and its policy for deleted files. This technique thus connects filesystem metadata to the file data, enabling at least *metadata and content recovery* (Casey et al., 2019). Furthermore, the method does not depend on file signatures, since timestamps are essentially a dynamic signature

* Corresponding author.
E-mail address: kyle.porter@ntnu.no (K. Porter).

[1] While Scalpel uses a modified version of the single-pattern string matching algorithm Boyer-Moore (Boyer and Moore, 1977) (as of 2018 (Bayne et al., 2018)) and causes header-footer matching to run in $O(sn)$ time where s is the number of header-footer signatures and n is the length of the data being processed, Zha and Sahni (2010) created *FastScalpel* which uses the multi-pattern string matching algorithm Aho-Corasick (Aho and Corasick, 1975) that performs header-footer matching in linear time.

[2] In the case of general data carving methods (for example, the Bulk Extractor (Garfinkel, 2013)), omitted regular expressions may mean missed data such as credit card numbers, social security numbers, etc.

https://doi.org/10.1016/j.fsidi.2021.301266

for all filetypes.

Use-cases of GMTC and other metadata carving methods include scenarios where the filesystem has been severely damaged or overwritten. Moreover, GMTC can also be used to find metadata records hidden in perfectly functioning filesystems.

Generic Metadata Time Carving has several limitations as well. The largest of which is that the method can only find metadata records that contain precisely equivalent timestamps, thus limiting the recall of discovered files in an image to the same number of metadata records that contain at least 2 or more equal timestamps. In order to improve this limitation, we have created and implemented a new potential timestamp carving algorithm that performs timestamp prefix matching. Allowing for some minor tolerance of difference between potential timestamps, we aim to improve filesystem metadata record recovery recall for GMTC without significantly reducing its precision. We formalize our research questions below:

1. How does the value of the prefix parameter affect the precision and recall of the Generic Metadata Time Carving method?
   (a) How does the original Generic Metadata Time Carving method compare with the prefix matching implementation?
2. Do the experimental results indicate that Generic Metadata Time Carving, prefix matching or otherwise, may be used in realistic digital forensic scenarios?

We hypothesize that timestamp matching using shorter prefixes will result in more potential timestamp matches, thus increasing recall, while reducing precision, as is often seen in precision-recall trade-offs.[3] Furthermore, due to the time complexity of potential timestamp carving, previous work on this subject, and the size of our data we hypothesize that even relatively large images can be processed in a reasonable amount of time. To test our hypotheses, our prefix matching method of Generic Metadata Time Carving is applied to two NTFS images and one Ext4 image, where the sizes of the images range from 1 GB to 476 GB. For each image, we apply all possible prefix length parameters and record the runtime for the timestamp carver and filesystem specific parser, and also record the number of potential timestamp hits.

We perform a location-based data recovery evaluation to test the performance of our tools' abilities to carve for filesystem metadata records, wherein we measure their precision and recall for extracting records from specific files or regions of the disk (such as from the $MFT, $LogFile, and the inode table). Note that we are running our tools on the entire disk images, as the tools are intended to be used, but we only calculate precision and recall for identifying filesystem metadata record hits for specific regions of disk, on a particular partition. We would have liked to obtain the precision and recall of our tools' ability to carve for filesystem metadata records across an entire disk image or partition, but since we did not create the test images we have no way of knowing the ground truth information regarding the locations of all filesystem metadata records on any partition. We additionally determine the files containing any hits for file system metadata records found outside these test spaces.

The prefix-based potential timestamp carver, the filesystem specific parsers, and instructions on how to use the tools are provided on a GitHub repository.[4]

[3] In short, precision is the percentage of returned hits that are relevant to the user's search, and recall is the percentage of the total amount of relevant items that were returned.

[4] Timestamp Prefix Carving for Filesystem Metadata Extraction code https://github.com/TimestampPrefixCarving/Peer-Review.

The paper is organized as follows. The Introduction section has described the objectives and experiments of this paper and the Related Work section covers past work, such as prominent metadata carving methods and timestamp carving methods. The Methodology section describes our prefix matching potential timestamp carving algorithm, how it fits into the greater Generic Metadata Time Carving methodology, a description of the experiments, and a description of the location-based data recovery evaluation. The Results section covers the outcomes of our experiments. The Discussion section analyzes and synthesizes our results, as well as looks at the limitations of this study. Lastly, we conclude in the Conclusion and Further Work section.

2. Related work

Metadata carving is a niche field in digital forensics, as opposed to file carving which is better studied, so we have attempted to cover the subject in full. In summary, these methods do not depend on critical filesystem data such as the $MFT record, superblocks, or group descriptor tables. Carving for metadata is done in a byte-wise manner, and if the pointers in the metadata records to their files have not been deleted, then there is a possibility of full file recovery.

In Fig. 1, we show an abstraction of a disk image with two partitions, which also shows a simplified abstraction of filesystems. The image is a representation of how critical filesystem data such as the MFT table or superblocks keeps track of the filesystem metadata records. For details on traditional file carving, see the Forensics Wiki (forensicswiki.xyz, 2012).

2.1. Metadata carving

One of the first works that scientifically studied metadata carving was done by Dewald and Seufert (2017). They exclusively carve for inodes in Ext4, where they intentionally made their images' superblocks and group descriptor tables unusable. Their method of byte-wise search uses search patterns that are expected to be found in inodes such as file flags, extent header magic numbers, and tests the relationships between the timestamps

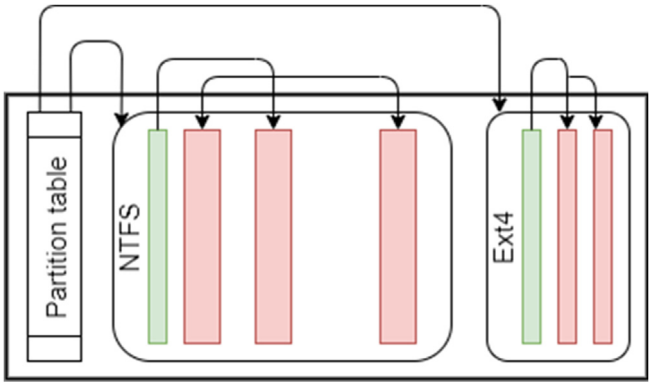

Fig. 1. An abstraction of a simple disk image, partitions, and filesystems. The large encompassing rectangle is the entire disk image, the furthest left rectangle with internal lines is the partition table that points to the partitions, and the other rectangles with rounded corners are partitions. Each partition has a filesystem, where the green rectangles represent filesystem critical data structures such as the $MFT record (and its mirror), superblock, or group descriptor table. These help keep track of the filesystem records (for example, inodes or MFT records), which are represented by the red rectangles. *Generic Metadata Time Carving* (Nordvik et al., 2020), and our work, attempts to find the red blocks without help from the green blocks. For a more complete picture of the general filesystem structure, see the work by Carrier (Carrier, 2005). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

ensuring their validity. In their experiments, they tested a variety of combinations of the inode attributes to search for, where the effectiveness of the combination of patterns is dependent on the case. Similar work was performed by Plum and Dewald (2018), where they carved for nodes using different combinations of object type and subtype.

Our work extends the work by Nordvik et al. (2020), wherein they created the Generic Metadata Time Carving method. The method considers every non-overlapping m length sequence of bytes as a search keyword, where this keyword is checked for equivalency against every non-overlapping m length sequence of bytes in a k length search window directly following the keyword. If the keyword matches one or more of the sequences of bytes in the search window, then the location of the potential timestamp is recorded. One notable aspect of their byte-wise search approach to timestamps is that it does not require the user to set a minimum or maximum date they are searching for, only a guess as to what the length of the timestamp is and the filesystem that is suspected. For instance, it is required that the length m of the timestamp must be defined as either 4 or 8. After a list of potential timestamp locations are compiled, another scan over the disk image is performed that is filesystem specific. Using the list of potential timestamp locations, they directly access each location on the image and check if specific byte offsets relative to the timestamp location fit the profile of the metadata record being searched for. Examples of such expected features are Standard Information Attribute (SIA) or Filename Attribute (FNA) flags for NTFS, or the extent header magic number for Ext4. Once the metadata record is identified as a positive hit, the metadata information can easily be read out, including resident files for NTFS records, dataruns from NTFS Data Attributes, and extents and block pointers from Ext4.

The potential timestamp carver by Nordvik et al. (2020) works generally, but to date, their filesystem specific parsers only support NTFS and Ext4. For their experiments, they created disk images with a known set of files for each filesystem, where each filesystem was then damaged by reformatting the image with a different filesystem. Their results for the NTFS experiment achieved greater than 99% precision in identifying MFT records and full recall in retrieving files known to the original filesystem. For Ext4 they obtained 100% precision in identifying inodes, but only retrieved about 23% of the inodes known to the original filesystem. Their experiments however only included metadata records that had multiple timestamps that were exactly the same, thus our work intends to apply a prefix matching version of their approach to more realistic datasets.

The first notable reference to byte-wise timestamp carving appears to be from McCash (McCash and 5, 2010), wherein he used an EnScript from Mueller (2008) to discover MFT records, indexes, and registry keys. The timestamp carving methodology essentially allows the user to input a date or range of possible dates, the EnScript converts the dates into their NTFS byte format, and the possible byte sequences are then searched for. To improve the precision of the tool, there is an option to search for contiguous potential timestamps.

2.2. Related methods of data retrieval

We briefly touch upon some tools that do not strictly use filesystem metadata extraction, but whose functionality is similar.

One tool that focuses on Ext4 file recovery via non-traditional means is Ext4Magic (Maar, 2014). The basics of the tool is that it uses journal blocks with an old but functional deleted inode. The inode will hopefully point to datablocks which have not been reused for a different file.

Bulk Extractor by Garfinkel (2013) gathers relevant forensic features such as email addresses, phone numbers, credit card numbers, and more by parsing through a disk image in a single scan block by block. A benefit of the tool is that it truly is filesystem agnostic, and can analyze different parts of the disk in parallel.

3. Methodology

In this section we first describe our prefix-based potential timestamp carving algorithm and how it fits into Nordvik et al.'s (Nordvik et al., 2020) Generic Metadata Time Carving workflow, and then describe our experimental and evaluative methodologies.

3.1. Prefix-based potential timestamp carving algorithm

The prefix-based potential timestamp carving algorithm, like the exact matching version by Nordvik et al. (2020), is used to identify the byte offset locations of potential filesystem metadata record timestamps from across an entire disk image. The algorithm outputs the offsets from the beginning of the image to a text file. Our prefix-based potential timestamp carving algorithm adds unique elements to the timestamp carving algorithm and source code from Nordvik et al. (2020). We briefly review the original algorithm, as understanding their work is imperative for understanding our own contributions. Their basic assumption is that timestamps within filesystem metadata records are typically co-located close to each other, and often two or more timestamps are identical.[5]

The search procedure for these algorithms is based on the *sliding window* approach, where we have some byte-stream T representing the forensic image being searched, and we let m be the length of a timestamp. Potentially, almost every non-overlapping m bytes of the forensic image is tested as a *candidate timestamp*, and used as a *keyword*. This test requires a user defined value k, which is the length of bytes for a search window following each candidate timestamp. If the candidate timestamp passes the test, then it is considered to be a *potential timestamp*. The search begins at $T[0]$, with the first candidate timestamp being $T[0 : m - 1]$. This candidate timestamp is then compared to each non-overlapping m byte sequence within the k length window for equivalency. We refer to this process as the *timestamp equivalency test*, and these byte sequences as *test sequences*. If the number of matches is greater than or equal to the threshold $h - 1$, h being the number of required matching timestamps within a metadata record set by the user, then the position of the candidate timestamp (now potential timestamp) is recorded, the search skips ahead by k, and repeats the search procedure. The definitions given in this paragraph and

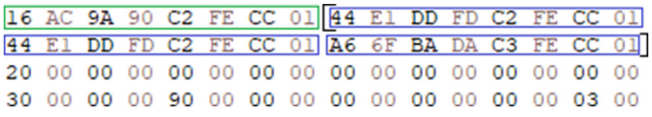

Fig. 2. For 8 byte timestamps, the *candidate timestamp* is highlighted with green, and the *test sequences* are highlighted in blue. The search window is indicated by the brackets. The *timestamp equivalency test* simply checks how many times the candidate timestamp matches the test sequences. If the number of matches is greater than or equal to the threshold $h - 1$, where h is the number of required matching timestamps within a metadata record set by the user, the candidate timestamp is deemed a *potential timestamp*. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

[5] Filesystem metadata records often record their timestamps consecutively, and oftentimes actions on a file (creation, access, modification, etc.) update several of its timestamps simultaneously.

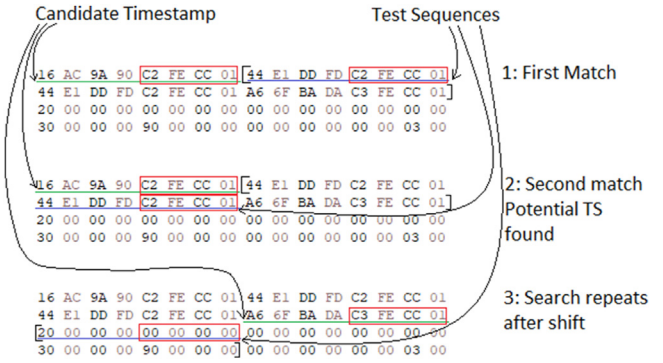

Fig. 3. Hex dump with highlights to illustrate the timestamp prefix matching search procedure. The byte sequence underlined in green represents the current candidate timestamp, and those underlined with blue are test sequences. The brackets represent the candidate timestamp's search window. The red boxes represent the little-endian prefixes that are being compared for equivalency. The first two examples show matches, despite the fact the candidate timestamp does not equal the subsequent ones. If three matching timestamps are required ($h = 3$), the third example shows the advancement of the search by k bytes, and begins to repeat the entire procedure. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

the timestamp equivalency test are illustrated in Fig. 2. If the candidate timestamp is not found to be a potential timestamp, then the search only skips ahead by m, and the search procedure repeats. The search continues until the last k bytes of T. Full details of the algorithm can be found in Nordvik et al. (2020).

Our major contribution in this work is the modification of the timestamp equivalency test. In most cases, timestamps that are stringologically similar should also be temporally similar. Our implementation of the timestamp equivalency test simply tests if the p most significant bytes, which we refer to as the prefix, of a candidate timestamp is equivalent to the prefixes of the test sequences.

An example of such a search is shown in Fig. 3. The prefix of the timestamp (the most significant bytes) is least likely to change when a timestamp is updated, and typically holds information regarding the timestamp's month and year. We argue this form of prefix matching is more suitable as an approximation metric than other popular metrics such as the Hamming (1950) or edit distance (Levenshtein, 1966), since they do not consider the order in which the matching errors occurred. Furthermore, the method for prefix matching is algorithmically simple.

For testing if a candidate timestamp is a potential timestamp, the original algorithm by Nordvik et al. (2020) converted the candidate timestamp byte sequence and the test sequences to their big-endian forms for using them as `unsigned long longs`, and we do this as well. For most operating systems and filesystems, timestamps are recorded in little-endian, but utilizing them in a program in big-endian is generally more useful. In order to test whether the candidate timestamp and the test sequences have an equivalent prefix of length p (that the p most significant bytes of the timestamps are the same) we need only XOR the timestamps in their big-endian form, and shift the resulting value to the right by 8*(m − p) bits. The shift to the right removes the m − p least significant bytes from the result of the XOR, and if the remaining value is 0, then the prefixes must match. If the prefixes match, the count of matching timestamps for the candidate timestamp is increased by 1. Algorithm 1 explicitly describes this process. The prefix matching algorithm is only a component within the prefix-based potential timestamp carving algorithm (see Algorithm 2), which is shown in Appendix A.

Algorithm 1. Prefix matching algorithm.

Input: Big-endian `unsigned long long` forms of candidate timestamp x and test sequence y
Output: Number of matches for the candidate timestamp either increases or stays the same
m # Length of timestamp;
p # Size of prefix;
$xorResult = x \bigoplus y$;
if *((xorResult >> (8*(m-p))) == 0)* **then**
  matchCount += 1;
end

Algorithm 1: Prefix matching algorithm.

We add one extra condition that must be met when applying prefix-based timestamp carving. The original method requires that candidate timestamps cannot be sequences of repeated bytes, as this filters out very common byte sequences such as 0x0000 and 0xFFFF. These common byte sequences would otherwise generate many false positive timestamp matches. We keep this condition, but add that a candidate timestamp's most significant bytes cannot be 0. This is due to the fact that there are many byte sequences which in big-endian start with non-zero values, but end with zeros, which will cause our algorithm to identify many potential timestamps where the candidate timestamp is a non-zero byte sequence, but the prefixes being tested are not. We consider this to be a fair assumption to make, as most timestamps' most significant bytes are non-zero. Both conditions can be found in Algorithm 2 within A.

Since the modification of the potential timestamp carving algorithm by Nordvik et al. (2020) primarily only changes the timestamp equivalency test by a constant number of steps, it implies that the time complexity of the prefix-based potential timestamp carving algorithm must remain the same. While Nordvik et al. showed that a general potential timestamp carver would run in nonlinear time (their worst case scenario omits the repeated byte timestamp check and conversion from a string of characters to a 64-bit data type), the implementation of the potential timestamp algorithm effectively runs in linear time, dependent on the length $|T|$ of the disk image. This is because length m is limited to a choice of 4 and 8 (timestamps are stored in an unsigned long type), and because the search window length k in most practical situations would never exceed the size of a block or cluster.

Note, the potential timestamp carving algorithm assumes that timestamps fall on 4 or 8 byte boundaries, so metadata that is unaligned will be missed. We also note that if timestamps are recorded in big-endian, prefix-matching will not work.

3.2. Generic Metadata Time Carving and the filesystem specific parsers

The potential timestamp carving algorithm described in the previous section is only the first step of the Generic Metadata Timestamp Carving (GMTC) methodology. The produced list of potential timestamp locations in general will be extremely large, and thus referring to many “false positive timestamps” identified across the full disk image. In the GMTC method, the list of potential timestamp locations is then fed into a filesystem specific parser, and the parser checks the offsets on the disk image given by the potential timestamp list in an attempt to verify if the potential timestamp is contained within a filesystem metadata record. The output of the parser is a .csv and .txt file containing data from the suspected records. More or less, a filesystem specific parser acts as a filter.

We briefly describe the strict verification tests performed by the NTFS and Ext4 parsers. The majority of stated parser tests were in

the original work by Nordvik et al. (2020)

The NTFS parser first checks if the date of a potential timestamp falls within the year range of 1970 and 2100. If so, it then checks all possible offsets behind the potential timestamp location for Standard Information Attribute (SIA) or Filename Attribute (FNA) attribute header flags. If one of these flags are found, we make an assumption of where the attribute begins. We can use the identified attribute lengths to navigate from one attribute to the next, where we require to start from the SIA, then hop to an FNA, and then if possible the Data attribute. Furthermore, the MFT attributes encountered must be in numerical order (as given by their header flags). Any attributes encountered for an assumed record must occur within a 1024 byte space. The parser also only reports an MFT record if the identified filetype extracted from an FNA is one of four possible types. The accepted filetypes are: “File”, “Directory”, “Index View”, or “Directory and Index View”. More information regarding MFT data structures can be found in the work by Carrier (2005).

The Ext4 parser is more complicated due to the fact that inodes are small, have far less features to strictly identify than MFT records, and that the parser attempts to connect the inodes to a filename and inode number. Both the inode number and filename are in a different data structure than the inode. The first step for validating potential timestamps is to check the possible offsets from the potential timestamp to the filetype nibble at the start of the inode. We only allow for three different types: “Regular Files”, “Directories”, and “Symbolic Links”. The offset to one of these values dictates our guess to where the inode begins. If the extent flag is set, and the offset to the extent header magic number is 0xF30A, or if there is no extent flag and the offsets from the beginning of the inode 0x24 to 0x27 are 0, we continue our validation tests. The total size of the file is checked to see if it corresponds to the total amount of blocks it is occupying, if the total size of the file is less than the size of the image, and if the relationships between the timestamps are valid. For instance, we check if the deleted value is not 0, then it must be greater or equal to both the modified and created time. The steps so far are the validation checks done in the preprocessing phase, as we need to gather information to try to connect inodes to their filename and inode numbers. If the inode passes the initial preprocessing and validity checks, it is fully processed. The timestamps are checked to ensure they fall within the years 2000 and 2020 (the deletion timestamp being the exception). For more information about Ext4 data structures and inodes, see (Ext4 (and Ext2/Ext3) Wiki, 2019).

3.3. Experimental methodology

We use our prefix-based Generic Metadata Time Carving (GMTC) method on three realistic forensic images. We first apply our novel prefix-based potential timestamp carving algorithm on the images, where the output of the algorithm creates a text file list of the locations of the potential timestamps in byte offsets from the beginning of the image. This list is then input into one of the two pre-existing but modified filesystem specific parsers (NTFS or Ext4), where the output of the parser is a .csv and .txt file with data from the discovered metadata records. We identified a few bugs in the original filesystem specific parser scripts by Nordvik et al. (2020), so we updated them so that we may achieve more complete and accurate results.

The three images being tested are a 1 GB NTFS image, one 59.5 GB Ext4 image from a real device, and one 476 GB synthetic NTFS image. The small NTFS image is from NIST's Deleted File Recovery page (DFR-13) (NIST, 2017), the Ext4 image was extracted by the authors from a real Samsung S8 mobile phone, and the large NTFS image is the “Lone Wolf” forensic image, available from Digital Corpora (Moore et al., 2018). Notably, these images are not guaranteed to have at least two equivalent timestamps per metadata record, unlike the work by Nordvik et al. (2020).

For each image we try all possible sizes, p, of the prefixes of the candidate timestamps that are required to be equivalent to the prefixes of the test sequences.

Since it is possible that the prefix of the candidate timestamp can be the length of the timestamp itself, we are also comparing the precision-recall performance of the original GMTC method to our method.

We clarify some items regarding our testing and evaluation methods. The prefix-based GMTC methodology (the potential timestamp carving followed by the filesystem specific parser) is applied to the entire disk image for our tests. Thus we obtain potential timestamp locations and filesystem metadata records from across entire disk images. However, since we have no ground truth information regarding the number and location of all file system metadata records across the disks, we cannot evaluate the precision and recall of our tools across an entire disk. Thus, we limit our precision and recall evaluations to specific areas or files on the disk, where we can easily retrieve the number and location of records. Examples of such files or regions of disk include the MFT table or inode table for a particular partition. This is explained more in depth in the next subsection.

Other data we record are the number of potential timestamps logged by the prefix-based potential timestamp carving algorithm, the time required by this algorithm and the filesystem specific parsers, and the number of metadata records extracted that were outside the $MFT, $LogFile, or inode table and which file they were found in.

3.4. Precision-recall location-based data recovery evaluation

Ideally, our experiments would measure the precision and recall of our tools' ability to carve all filesystem metadata records from an image or partition, but this is infeasible since we have no reliable method of obtaining ground truth knowledge of every single offset of every single file system metadata record. Thus, while we still run our tool on entire disk images, we focus on our tools’ precision and recall for carving filesystem metadata records from specific files or regions of disk where record offsets are more easily obtainable. We also determine the files on the same partition that contain the hits for file system metadata outside the precision-recall evaluated files or data structures.

For NTFS, we measure the precision and recall for carving MFT records from the $MFT of the partition of interest, and we perform another precision and recall evaluation for carving MFT records from the $LogFile. To obtain ground truth knowledge of the offsets to MFT records in each file, we used The Sleuthkit to export these files from the NTFS image with icat, search the file for FILE signatures, and check if the 0x38 byte offset from the FILE signature equals the Standard Information Attribute flag (0x10000000). For the $LogFile, we also check the 0x78 byte offset from the FILE signature for the cases that an MFT record is divided between $LogFile pages, where the beginning of each page has header that begins with the signature “RCRD”. Using The Sleuthkit's istat command we also obtained the clusters that the $MFT and $LogFile occupy, so that we can translate the files' logical offsets to MFT records into physical offsets on the disk. All the discovered offsets for MFT records in the $MFT matched possible locations of records given by the clusters output from the istat command on the $MFT.

We refer to these ground truth offsets to MFT records as *Condition Positives*. Formally, a Condition Positive is the knowledge that at address A, there exists a filesystem metadata record. It is *conditioned* on the fact that we are limiting our precision-recall evaluations to the regions of the disk occupied by a specific file or ranges of disk space. By running our prefix-based GMTC method on the

entire disk image, we obtain a large set of byte offset locations that our tools detect as the locations of filesystem metadata records.[6] We refer to the offsets identified by our tools as *Test Positives*. One can think of our filesystem specific parsers similar to that of a classifier when they filter potential timestamp locations, which declares at address A we detect a filesystem metadata record of minimum length L. The value L for inodes and MFT records is 256 bytes (the minimum length of an MFT record includes the record header of length 0x38, Standard Information Attribute of length 0x60, and a minimum length Filename Attribute of 0x68). A Test Negative is simply that our tools do not detect a filesystem metadata record at address A.

For Ext4, we measure the precision and recall for carving inodes from the inode table of the partition of interest. To determine the Condition Positives in the inode table, we use the Sleuthkit's fls −r command to dump all files to a list (wherein we add the root directory and Journal with inode numbers 2 and 8 respectively). Using The Sleuthkit's fsstat command, we determine which blocks the inode table occupies, and which inodes are in which fragment of the inode table. Using this information (inode numbers provide a 256 byte multiple offset into their respective inode table fragment), we can calculate the physical positions of each inode offset from the beginning of the disk. We would have liked to perform similar tests on the Ext4 Journal, but we would need a more certain method of identifying Condition Positives other than performing a string search for the extent signature 0xF30A.

We reiterate that we limit the Condition Positives to the regions of disk where the precision and recall is being measured. If a Test Positive is also a Condition Positive, then the result produced by the GMTC tools is a true positive. That is, our tools detected a filesystem metadata record at address A with minimum length L, and the beginning of a record truly begins at address A. A false positive occurs if we obtain a Test Positive at some address B within the region of disk under examination, where according to our list of Condition Positives no record exists. If there are Condition Positive addresses that do not have a matching Test Positive address, then our tools have produced a false negative, a miss.

We use the typical precision and recall measures for our analysis, as seen in the equations below.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

When calculating precision and recall, it is possible that after accounting for all the Test Positives located in the disk image regions such as the $MFT, $LogFile, or inode table that there may still be a large number of Test Positive hits that are still unaccounted for elsewhere on a partition. To find where these extra records come from, we use The Sleuthkit's istat command to list the blocks/clusters (we refer to these as "blocks" from here on out) allocated to files known to the inode or MFT table, where the files' records are filtered with respect to our calculated Condition Positives. For each list of blocks extracted from the istat output, we create a list of block ranges that a file occupies, which also accounts for fragmentation. We then build a Python dictionary of such values where the key is the record number, and the values associated with a key are the block ranges of the file, and the file's name. It is then possible to create a derived version of this dictionary, where the key is a starting block of a particular file fragment, and the values associated with the key are the ending block of the file fragment, as well as the file's name and number. When this dictionary is ordered numerically, and our Test Positives are ordered by their offsets numerically, we can quickly search through all the file fragments to identify where our remaining hits lie.

3.5. Specifics of NTFS experiments

The 1 GB NTFS image is the 13th test case (dfr-13-ntfs.dd) from NIST's Deleted File Recovery page (NIST, 2017). This test case has performed random filesystem activity, so the timestamps of the MFT entries are rarely all equal. When running our prefix-based timestamp carving algorithm we set the length of the timestamps $m = 8$, the search window $k = 24$, and the required number of matching timestamps to $h = 3$ (the same parameters used by Nordvik et al. (2020)). We carved for timestamps for all possible prefixes p, from 1 to 8.

For this image, we only performed the location-based data recovery evaluation on the $MFT of the partition starting at sector 128, as we did not identify any MFT records in the $LogFile. We verified the lack of full MFT records in the $LogFile by running the LogFileParser by Schicht[7] on the file. Transactions in the LogFile where the Redo Operation or Undo Operation has the status of "InitializeFileRecordSegment", and the other Redo or Undo Operation has the status of "Noop" indicates that the transaction contains an entire MFT record (Cowen and Seyer, 2013). We found no such transactions.

The experiment using the 476 GB Lone Wolf forensic image (available from Digital Corpora (Moore et al., 2018)) focuses on the "Basic Data Partition" for the precision and recall evaluations, the largest partition on disk. Our timestamp carving experiments for the Lone Wolf image use the same parameters as the DFR-13 image.

We performed the location-based data recovery evaluation on the $MFT and the $LogFile on the LoneWolf image's partition.

3.6. Specifics of Ext4 experiments

The Ext4 experiment uses a dump of a Samsung S8 mobile phone running Android, where we specifically focus on the "SYSTEM" partition's inode table for the precision and recall calculations. The User partition was encrypted, and SYSTEM partition was the second largest partition on the image. The image was created by first flashing the recovery partition using the TWRP Recovery image (TWRP, 2019), and then using an ADB bridge executing a combination of netcat and dd commands in order to acquire the raw image. The recovery image method is described in detail by Son et al. (2013) and Vidas et al. (2011). We ran our prefix-based timestamp carving algorithm on the image with the same parameters as those used by Nordvik et al. (2020), where the length of the timestamps m was set to 4, the search window $k = 12$, and the required number of matching timestamps to $h = 2$. We carved for timestamps for all possible prefixes p, from 1 to 4.

The Ext4 parser also requires a few additional parameters, which are assumptions that assist in attempting to connect inodes to their filename and inode number. Using the Sleuthkit, we obtained the blocksize of 4096 bytes (which is the default blocksize (Ext4 (and Ext2/Ext3) Wiki, 2019)), and the byte offset of 225968128 to the partition. Thus, the parser only examines the disk from this offset onward.

[6] The Ext4 parser reports the byte offsets to the beginning of the inode, whereas the NTFS tool reports the byte offsets to the potential timestamp identified by the potential timestamp carver. Thus for MFT records, we have to consider the set of all possible locations of the beginning of the record with respect to the identified Standard Information Attribute timestamp.

[7] https://github.com/jschicht/LogFileParser.

3.7. Computer specifications

A Mac with the following specifications was used to run the timing experiments.

- OS: MacOS Catalina v 10.15.4
- Processor: 4.2 GHz Quad-Core Intel Core i7
- Memory: 64 GB 2400 MHz DDR4
- Storage: APPLE SSD SM0128L 3.12 TB, PCI-express, a hybrid, where 128 GB is pure SSD, and 3 TB is SATA. Sequential Read: 952 MB/s, sequential write 57 MB/s. Random read 0.9 MB/s, and random write 50 MB/s.

While running the tools we did not activate any other resource demanding processes. However, it is always possible that the OS performed additional scheduled tasks. We used the tool DiskMark[8] v2.2 to measure the read/write speed.

4. Results

Overall, our results show that by reducing the size of the prefix p of a timestamp in the timestamp equivalency test, a much higher recall for filesystem metadata record extraction can be achieved using the Generic Metadata Time Carving (GMTC) method as compared to the exact timestamp matching approach. To our surprise, the precision of the metadata extraction was not reduced by decreasing the size of the matching prefix p, and remained at 100% for all experiments. We go through each disk image we tested, showing the results of the individual precision-recall tests over specific areas of the disk, and the timing results for the potential timestamp carver and parser for that particular image. All timing experiments were run twice, and the listed runtimes are their averages. We also describe the files on the evaluated partition that contained filesystem metadata records that were outside the $MFT, $LogFile, and inode table.

4.1. Small NTFS image

The results for carving MFT records from the $MFT from the 1 GB NTFS image's partition beginning at sector 128, as seen in Table 1, show that applying prefix matching of timestamps greatly increases the recall, and appears to maintain the exact matching Generic Metadata Time Carving method's 100% precision. The exact matching GMTC ($p = 8$) only obtained 8.8% recall for finding MFT records, whereas decreasing p to 3 and less achieved a 97.9% recall. The number of Test Positives identified over the entire partition for different values of p is shown in Table 2. The true positives account for most of the Test Positives found over the entire partition, but three had gone unaccounted for. It transpired they were the $MFTMirr, $LogFile, and $Volume records found in the $MFTMirr file.

This increase in recall was not without its trade-offs, as seen in Table 3. Upon decreasing p from 8 to 4, the number of identified potential timestamp locations increased by three magnitudes. While this did not appear to unduly influence the timestamp carving algorithm, the time required for the filesystem parser increased more than 100 fold.

4.2. Ext4 Samsung S8 image

The results for carving inodes from the Ext4 image's SYSTEM partition's inode table are shown in Table 4. Like the small NTFS image, we achieved 100% precision in identifying inodes, where the recall increased for carving inodes from the inode table as the prefix length value of p decreased. However, the increase in recall was quite minor, only increasing by about 3%. We discuss our theories of why the precision and recall were so high for the Ext4 experiment in the Discussion section.

Table 5 shows the Test Positive counts of detected inodes found over the entire partition, with respect to the prefix length p being used. All test positive hits that were not discovered in the inode table were discovered in the Journal where, when the timestamp prefix length $p = 1$, we detected 1924 inodes.

In terms of computational performance, Table 6 exposed trends regarding the timestamp carving program when working with large files. Larger prefixes p caused the timestamp carver to take longer to complete, but not by too much. Unlike the small NTFS image experiment, the number of potential timestamp locations only increased by about one magnitude going from $p = 4$ to $p = 1$. The time required to run the filesystem parser appears to have an approximately linear relationship between the number of potential timestamp carving locations, as the time required to run at $p = 1$ is about 10 times as slow as using a prefix size of $p = 4$.

Table 1
Precision and recall for carving MFT records from the $MFT from the 1 GB NTFS image's partition beginning at sector 128 with $p = 1, 2, \ldots, 8$. The $MFT had 239 Condition Positives.

p	True Positives	False Positives	False Negatives	Precision	Recall
8	21	0	218	1	0.088
7	21	0	218	1	0.088
6	126	0	113	1	0.527
5	164	0	75	1	0.686
4	219	0	20	1	0.916
3	234	0	5	1	0.979
2	234	0	5	1	0.979
1	234	0	5	1	0.979

Table 2
Test Positive count over entire partition from the 1 GB NTFS image, where $p = 1, 2, \ldots, 8$.

P	8	7	6	5	4	3	2	1
Test Pos. Count	24	24	129	167	222	237	237	237

Table 3
Generic Metadata Time Carving performance for the entire 1 GB NTFS image with $p = 1, 2, \ldots, 8$. PTS stands for "Potential Timestamp".

p	# PTS Locations	TS Carve Time (s)	Parser Time (s)	Total Time (s)
8	892	8.177	0.049	8.226
7	3143	8.132	0.082	8.214
6	4311	8.128	0.104	8.232
5	3638	8.131	0.106	8.237
4	2056322	8.451	6.59	15.042
3	2056629	8.413	6.689	15.102
2	2056636	8.459	6.659	15.117
1	2061768	8.4	6.622	15.019

Table 4
Precision and recall for carving inodes from the inode table from the SYSTEM partition in the 59.5 GB Ext4 Samsung S8 image with $p = 1, 2, 3, 4$. The inode table had 7436 Condition Positives.

p	True Positives	False Positives	False Negatives	Precision	Recall
4	6766	0	670	1	0.910
3	6766	0	670	1	0.910
2	7004	0	432	1	0.942
1	7004	0	432	1	0.942

[8] https://inchwest.com/diskmark/.

Table 5
Test Positive count over entire SYSTEM partition from the Ext4 Samsung S8 image, where $p = 1, 2, 3, 4$.

p	4	3	2	1
Test Pos. Count	8470	8470	8928	8928

Table 6
Generic Metadata Time Carving performance for the entire 59.5 GB Samsung S8 image with $p = 1, 2, 3, 4$. PTS stands for "Potential Timestamp", *m* for minutes, and *s* for seconds. Note, the Ext4 parser skips the first approximately 210 MB.

p	# PTS Locations	TS Carve Time (s)	Parser Time (s)	Total Time (m:s)
4	10630945	1401.78	45.73	24:07.51
3	26228387	1397.36	115.25	25:12.60
2	41610448	1369.78	186.43	25:56.21
1	97555603	1266.81	452.42	28:39.22

4.3. Large NTFS image

The results for carving MFT records from the $MFT and $LogFile from the LoneWolf image's Basic Data Partition are seen in Tables 7 and 8 respectively. Again, we achieved 100% precision in identifying MFT records, both for the $MFT and $LogFile (we encountered no false positives with respect to our Condition Positive lists). The recall results reflect previous trends. Using exact matching timestamp carving we only achieved 41.6% recall for carving MFT records from the $MFT, and allowing for smaller timestamp prefix matching caused increasingly higher recall. The point of diminishing returns appeared to have occurred at $p = 2$, where about 97.2% recall was achieved. The recall results are quite different for carving MFT records from the $LogFile, as the recall hovered around 87% despite the value of *p*.

Table 9 shows the total number of test positives found over the entire partition, and after filtering out the Test Positive hits found in the $MFT and $LogFile, there were still a large number of hits left unaccounted for. When discussing where these hits were found on the partition, we focus on the results for $p = 1$, since each value of *p* larger than this should be a subset of the $p = 1$ results. In total, there were 91157 test positive hits that were yet to be accounted for. Using the dictionary we created that contained the allocated cluster ranges of all known files on the partition, we were able to discover where these potential MFT records were coming from, as seen in Table 10. The $MFTMirr contained the usual records of $MFT, $MFTMirr, $LogFile, $Volume. Four different boot.sdi files (with the filenames "boot.sdi, boot.sdi") each contained 42 filesystem metadata record hits, where a boot.sdi file is essentially a small partition of its own with completely irrelevant MFT records. It is used as a Ramdisk which can be shown with the *bcdedit* command (KillDisk, 2021). Lastly, we have the two Volume Shadow Copies[9] that contained 90985 detected MFT records.

In terms of timing performance, the large NTFS experiment mostly behaved as expected (see Table 11). Like in the Ext4 timestamp carving experiment, the run-times for all values of *p* were similar, but experiments with lower values of *p* took less time. What was rather surprising was the relatively small increase in potential timestamp locations that were found by $p = 1$ versus $p = 8$, given the size of the image. The increase was only by a factor of about 4.77, quite a deal less than the increase of magnitudes we saw before. A possible reason for this is that the Lone Wolf image is a synthetic image that was only being used for some months. Stranger still, were the parser times over all possible values of *p*. Given the previous results, we should have seen parser times drastically increase as *p* decreased. This did not happen, as seen by the fact that the parsing time for $p = 1$ was on average less than most other values of *p*, and this is despite the fact that the experiment for $p = 1$ had about 46 million more potential timestamps to check than the $p = 8$ experiment. Since both runs of the parser produced such similar results, at the moment we can only guess that some aspect of the parser script handles things inefficiently. A major difference between the NTFS parser and the Ext4 parser is that the Ext4 parser uses a Python memory mapping library[10] to handle the parsing of large files, while the NTFS parser has handcrafted code to handle large files.

We note that our tools found no Test Positives (detected hits of filesystem metadata records) in unallocated space for any of the disk images.

Table 7
Precision and recall for carving MFT records from the $MFT of the Basic Data Partition from the 476 GB LoneWolf NTFS image with $p = 1, 2, \ldots, 8$. The $MFT had 142960 Condition Positives.

p	True Positives	False Positives	False Negatives	Precision	Recall
8	59422	0	83538	1	0.416
7	59422	0	83538	1	0.416
6	72284	0	70676	1	0.506
5	95193	0	47767	1	0.666
4	120482	0	22478	1	0.843
3	129220	0	13740	1	0.904
2	139022	0	3938	1	0.972
1	139082	0	3878	1	0.973

Table 8
Precision and recall for carving MFT records from the $LogFile of the Basic Data Partition from the 476 GB LoneWolf NTFS image with $p = 1, 2, \ldots, 8$. The $LogFile had 2604 Condition Positives.

p	True Positives	False Positives	False Negatives	Precision	Recall
8	2251	0	353	1	0.864
7	2251	0	353	1	0.864
6	2251	0	353	1	0.864
5	2251	0	353	1	0.864
4	2263	0	341	1	0.869
3	2263	0	341	1	0.869
2	2267	0	337	1	0.871
1	2267	0	337	1	0.871

5. Discussion

Here we analyze our results, consider why we may have missed extracting some metadata records, the limitations of our research, and finally answer our research questions.

5.1. Analysis: small NTFS image

The small image from NIST (NIST, 2017) purposefully created

[9] Using The Sleuthkit's (versions 4.4.1 and 4.10.1 tested) istat command for Volume Shadow Copies (VSC) will show that the file only occupies a single cluster, having a large non-zero size, and an init_size of 0. This error has been seen before: https://github.com/sleuthkit/sleuthkit/issues/466. Why we bring this up is that relying on the Python dictionary we created for cluster ranges of files will be incorrect for the VSCs. To address this, we used the given cluster as the start of a VSC's range, and added the size of the file to obtain the end of its range. To ensure this unfragmented region of disk was truly a VSC file, we performed the following. icat -s will output a VSC entirely, and we took the MD5 hash of the VSC files. We then took MD5 hashes of the unfragmented disk regions defined by the byte ranges we were using for the VSCs. The hashes of the files and the regions of disk were identical.

[10] https://docs.python.org/3/library/mmap.html.

Table 9
Test Positive count over entire Basic Data Partition from the large LoneWolf NTFS image, where $p = 1, 2, \ldots, 8$.

p	8	7	6	5	4	3	2	1
Test Pos. Count	108852	108852	134227	169588	204467	218559	232425	232506

Table 10
Files containing the remaining Test Positives not found in the $MFT or $LogFile of the Basic Data Partition, where $p = 1$. The number associated to each file indicates how many MFT records were found in that particular file.

File	Record Number	Test Positive Count
$MFTMirr	1	4
Boot.sdi	21992	42
Boot.sdi	21993	42
Boot.sdi	21994	42
Boot.sdi	21995	42
Volume Shadow Copy 1	96066	51795
Volume Shadow Copy 2	123530	39190

Table 11
Generic Metadata Time Carving performance for the entire 476 GB NTFS image with $p = 1, 2, \ldots, 8$. PTS stands for "Potential Timestamp", *hr* for hours, *m* for minutes, and *s* for seconds.

p	# PTS Locations	TS Carve Time (s)	Parser Time (s)	Total Time (hr:m:s)
8	12235330	7324.28	1761.83	2:31:26.11
7	17426880	7322.61	2177.43	2:38:20.04
6	23192352	7291.95	2793.94	2:48:05.89
5	30135982	7279.97	2266.49	2:39:06.45
4	32353209	7286.00	2218.84	2:38:24.83
3	33625310	7296.31	2596.24	2:44:52.55
2	46791325	7298.88	1617.81	2:28:36.68
1	57934625	7137.19	1707.40	2:27:24.59

chaotic actions on the system, thus creating MFT records with erratic timestamps. The missed MFT records from the MFT table when $p = 1$ were the $MFT, as the Standard Information Attribute timestamps were 0, and 4 other records that did not contain File Name Attributes. The NTFS parser requires a File Name Attribute to be present.

The only other interesting item to note is explaining why the number of potential timestamp locations jumped drastically from $p = 5$ to $p = 4$. The *dfr*-13-ntfs.dd image fills sectors not occupied by an MFT entry with repeated byte sequences of either 0x2A or 0x5A, and the beginning of each sector has a message describing how the sector is or is not used. The combination of this message and the repeated byte sequences creates a large occurrence of valid potential timestamps. Such a situation would be unusual for more realistic images.

5.2. Analysis: Ext4 Samsung S8 image

The location-based data recovery evaluation for carving inodes from the inode table of the Ext4 Samsung S8 image performed suspiciously well, having 100% precision and 91% or greater recall. The high recall for extracting inodes from the inode table may indicate that the SYSTEM partition had fairly static files. Again, we would have liked to run the tests on the User partition, which would have included real user behavior, but it was encrypted.

The 432 false negative inodes from the inode table were entirely comprised of Symbolic Links. While the GMTC method by Nordvik et al. (2020) and our work is said to consider symbolic links (and will catch some), the Ext4 parser always assumes that at offset 0x28 from the start of the inode will be direct blocks or the start of the extents. However, this is an incorrect assumption, as a symbolic link will be stored at this offset if the string is less than 60 bytes long (Ext4 (and Ext2/Ext3) Wiki, 2019).[11]

5.3. Analysis: large NTFS image

Other than the strange runtimes of the NTFS parser, the results from the experiments on the large NTFS image were in line with what we had seen in the previous images. Three items of interest are worth discussing: The high recall of MFT records carved from the $LogFile, false negative MFT records, and Test Positives found outside the $MFT and $LogFile.

The high recall of at least 86.4% of MFT records found in the $LogFile can be attributed to the fact that full MFT records only occur in $LogFile transactions with "InitializeFileRecordSegment" operations, which means a new file is being created (Schicht, 2018). When a new file is created, all the timestamps for the Standard Information Attribute (SIA) are updated (Knutson and Carbone, 2016). This implies all the timestamps for the MFT records should be the same or nearly the same. Decreasing the timestamp prefix length p from 8 to 1 increased recall by less than 1%, which shows that some of the timestamps were indeed slightly different. The implications of these results is that the exact matching GMTC method will work well for carving MFT records from the $LogFile. However, as Nordvik et al. (2020) previously observed, the majority of records recovered from the $LogFile contained no datarun information, where we only identified 11 records that did.

It would appear that most of the 3878 false negative MFT records in the $MFT were those that needed to have non-resident attributes. This was expected, as the NTFS parser does not handle MFT records that are larger than 1024 bytes. Most of the 337 false negative MFT records in the $LogFile were records that crossed log

[11] https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Symbolic_Links.

pages, where a log page header (containing the magic number 'RCRD') split the MFT record somewhere after the Standard Information Attribute. We do however still find MFT records where they are split by a log page header after the MFT record header, and before the Standard Information Attribute.

Of the 91157 Test Positive hits for MFT records that were found in neither the $MFT or the $LogFile (see Table 10), the 90985 hits in Volume Shadow Copies are the most interesting. This is because the Volume Shadow Copies are snapshots of previous states of the partition, and thus may either contain previous states of files and their MFT records, or they may contain MFT records that are now deleted. Furthermore, a large number of Test Positive hits outside the MFT table or LogFile, but within specific regions of disk, may indicate that Volume Shadow Copies even exist on a partition in the first place.

5.4. Limitations

The purpose of this work was to show that the GMTC method can be used on realistic images and timestamps, and that the use of a timestamp prefix matching method could greatly improve the method's ability to extract filesystem metadata records. Here, we address the issues we believe to be the primary limitations of our work.

The first issue is that we are applying the GMTC method to data that does not fit its more typical use-case. The GMTC method should be applied if the filesystem is damaged or otherwise inaccessible. We are applying the method to perfectly working images, and undamaged filesystems. The reason for this is to understand what the prefix-based GMTC method can *potentially* recover.

Another limitation of this work is that we were not using a user partition for the Ext4 experiments, so the filesystem we were analyzing was likely more static and not as realistic as we would have liked it to be.

The last large limitation of our work is our experiments' unsatisfying explanation of the unflagging 100% precision. However, by looking at our results, we can see that low timestamp prefix lengths do indeed produce many more false positive potential timestamps. For example, reducing the prefix length p from 4 to 1 in the Ext4 experiment increased the number of potential timestamps from approximately 11 million to 98 million. For the Large NTFS experiment, reducing p from 8 to 1 increased the number of potential timestamps from about 12 million to 58 million. The effect of prefix length p on the number of potential timestamps identified for the large NTFS and Ext4 images is shown in Figs. 4 and 5 respectively, where we also show the number of Condition Positives to illustrate that the count of Condition Positives is only a fraction of the number of false positive timestamps we may be encountering. According to our filesystem specific parsing experiments, it would seem that the filesystem specific parsers are extremely strict since we encountered no false positive records despite checking for millions of more offsets on the disk image. Likewise, it appears that the roll of the potential timestamp carver is to control the total number of byte offsets that a filesystem specific parser must verify when looking through a disk image for filesystem metadata records (affecting recall), and that the filesystem specific parser ultimately controls the precision of the GMTC method.

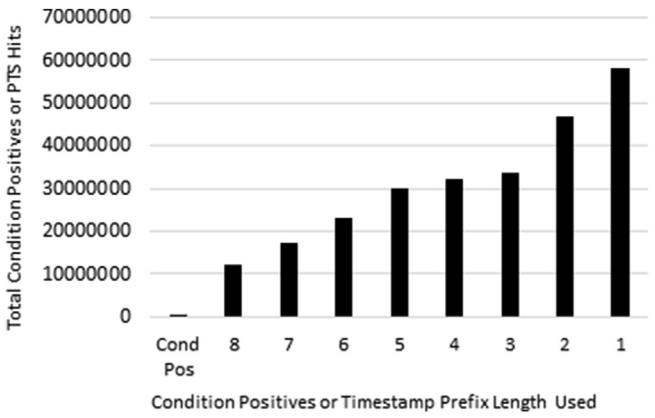

Fig. 4. Histogram comparing the number of Condition Positives we account for on the Basic Data Partition of the 476 GB Lone Wolf image and the number of potential timestamp (PTS) locations identified after carving for all possible prefix lengths.

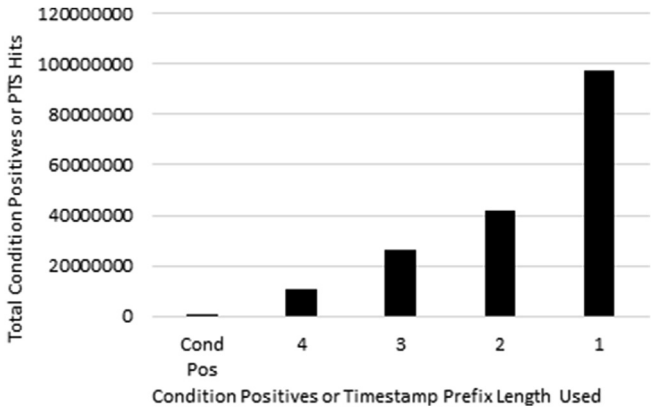

Fig. 5. Histogram comparing the number of Condition Positives we account for on the SYSTEM partition of the 59.5 GB Samsung S8 image and the number of potential timestamp (PTS) locations identified after carving for all possible prefix lengths.

We wanted to observe these suspected rolls of the prefix-based potential timestamp carver and filesystem specific parsers empirically, so we conducted a short experiment. This experiment obtains the results of performing the prefix-based GMTC method on an encrypted image, as the data is essentially a large string of random bytes, and also obtain the results of applying the filesystem specific parsers directly on the encrypted image without first performing timestamp carving. Results we were interested in included the number of false positive filesystem metadata records the experiments would encounter, and how long the runtimes for the different experiments were. Our hypothesis was that we would not encounter any false positive records for any experiment, and that Generic Metadata Time Carving should be faster than applying the parsers directly on the image.

We encrypted the 59.5 GB Ext4 image with Kleopatra[12] and carved for potential timestamps on the encrypted image using the same parameters as our previous experiments, but only searched for timestamps based on a prefix size of 1 byte. Then both filesystem specific parsers were ran on the encrypted image using their respective potential timestamp locations from the potential timestamp carving. Next, we ran the NTFS and Ext4 parsers over every byte of the encrypted image without using potential timestamp information, with the exception of the first and last 1024 bytes.

For searching for NTFS MFT records, we obtained no false positives for the GMTC experiment or the pure parser experiment. Carving for potential timestamps took approximately 17 min, and where 360990 potential timestamps were discovered. Applying the NTFS specific parser on the image with the potential timestamp results took about 7 min to run. Applying the NTFS parser directly on the encrypted image took 5.75 h.

[12] https://www.openpgp.org/software/kleopatra/.

When searching for inodes, we encountered no false positives for the GMTC experiment or the pure parser experiment. The potential timestamp carving took about 17.5 min, and we identified 182405692 potential timestamps. When applying the Ext4 specific parser on the image with the potential timestamp results, the parser ran for about 17.5 min. Applying the Ext4 parser directly on the encrypted image took about 17.5 h.

These results further reveal why it is the GMTC method produces little to no false positives. The tests where the parsers are directly executed on the encrypted image show that it is the file system specific parsers that ultimately control the precision of the GMTC method since no false positives were encountered. This implies that the filesystem specific parsers are extremely strict when verifying filesystem metadata records, and that the records themselves are highly structured. However, running the parsers directly on the encrypted image took much longer to run.

With these results we get a clearer picture on how potential timestamp carving effectively acts as a data reduction technique, where its parameters influence the number of potential timestamps returned, going on to influence recall and parser runtime, and that the filesystem specific parsers ultimately control the precision of the GMTC method. We can also see that the other parameters for the potential timestamp carver such as the user defined threshold h of the required number of matching timestamps per record also controls the number of returned potential timestamps. For example, despite both applying a prefix length $p = 1$, using the NTFS timestamp carving settings (requiring $h = 3$ matching timestamp prefixes) only encountered 360990 potential timestamps, whereas carving with the Ext4 settings (requiring $h = 2$ matching timestamp prefixes) encountered 182405692 potential timestamps. However, more research needs to be done to understand all the implications of applying different potential timestamp carving parameters.

5.5. Revisiting research questions

Below, we answer our research questions based on our results and analysis of the experiments.

How does the value of the prefix parameter effect the precision and recall of the Generic Metadata Time Carving method?

We hypothesized that as the length of the prefix of the most significant bytes, p, of a potential timestamp decreased, that this in turn would increase the recall but reduce the precision of the Generic Metadata Time Carving method. According to our results, the recall for finding metadata records may significantly increase when applying prefix-based timestamp carving, but the precision in our experiments did not decrease when applying prefix-based timestamp carving. In fact, the precision remained at 100% for all possible prefix values. These items require a short discussion.

It seems that the recall reaches a point of diminishing returns once the timestamp prefix length $p \leq 2$, no matter what the filesystem is. We cannot suggest to make the value of p as low as possible either, as reducing $p = 2$ to $p = 1$ increased the number of potential time timestamps locations in the Ext4 experiment by 55.9 million (increasing parser time by over 100%) and in the Large NTFS experiment by 11.1 million. As noted in the Limitations subsection, decreasing p yields more potential timestamp locations, most of which will not be timestamps at all, but also allows for greater filesystem metadata record recall.

Despite producing much greater recall and many more potential timestamps, lowering the prefix-length p did not reduce the precision for carving MFT records or inodes. Our brief further investigations in the Limitations subsection demonstrated that by using the filesystem specific parsers on an encrypted version of the 59.5 Ext4 image produced no false positive record hits. We mention the trade-offs between the potential timestamp carver and parsers when addressing the last research question.

But overall, we can state with confidence, according to our experiments, that decreasing the value of the prefix parameter p can drastically increase the recall of finding metadata records, without much (if any) loss in precision of identifying metadata records.

How does the original Generic Metadata Time Carving method compare with our prefix matching implementation?

For comparing our GMTC method to the original, we simply set the value of the prefix length, p, of a potential timestamp to its maximum (8 for NTFS or 4 for Ext4). In the small NTFS experiment, the exact matching timestamp method only achieved a recall of 8.8% for carving MFT records from the MFT table, while using the prefix-based method achieved 97.6% recall. Then in the Ext4 experiment, the exact matching timestamp method achieved a 91% recall for carving inodes from the inode table, while using the prefix-based method achieved 94.2% recall. In the Large NTFS experiment, the exact matching timestamp method only achieved a recall of 41.6% for carving MFT records from the MFT table, while, using the prefix-based method achieved 97.3% recall. The precision-recall experiments on the $LogFile also showed improvement of recall as p decreased, though not nearly as drastic as the MFT Table experiments. In fact, our results indicate that the exact matching GMTC method performs nearly identically on the $LogFile from NTFS as our prefix-based version, due to the nature of MFT records found within the file. In all experiments, the precision remained a constant 100%.

Our results indicate that the degree of improvement of the recall is dependent upon the temporal variety of the filesystem metadata records. Both the $LogFile and inode table results showed only a minor improvement in recall since both the data sources appeared to have static records. As the records in the $MFT from both NTFS images were more often updated, then the improvement in recall was much more significant.

Overall, we have shown that the prefix-based GMTC method can potentially carve a significantly greater number of filesystem metadata records than the original, while maintaining perfect or near-perfect precision for realistic test datasets.

In terms of time and space complexity, the time complexity of the prefix matching algorithm is the same as the exact matching algorithm. Our results show that the timestamp carving times are close to constant for all values of p, but carving with lower values of p will take slightly less time. A limitation of our work is that we did not perform extensive tests on the original GMTC algorithm, thus making statements on the speed of our algorithm versus the original mostly theoretical. Where the prefix-based GMTC method performs worse than the original method, is the space required for the potential timestamp locations produced by the potential timestamp carver, and consequentially the time required by the filesystem specific parsers. For example, the exact timestamp carving on the Ext4 image identified nearly 11 million potential timestamps and the parser took about 46 s to run, but carving for timestamps with a prefix length of 1 byte on the same image identified nearly 98 million potential timestamps and the parser took about 7.5 min to run. As there were only 7436 inodes in the Ext4 partition's inode table, the grand majority of the potential timestamps are false positive timestamps.

The rather unexpected results of the timing of the NTFS parser for the Lone Wolf image, where the time to parse the image decreased when applying $p = 1$, is likely the result of the implementation of the NTFS parser.

Do the experimental results indicate that Generic Metadata Time Carving, prefix matching or otherwise, may be used in realistic digital forensic scenarios?

In terms of functionality, the prefix-based GMTC method appears practical for carving filesystem metadata records as our experiments acheived recall of approximately 90% or greater. The exact matching GMTC can be practical if time is of primary concern, or one wishes to carve for records in specific files such as $LogFile (where the existence of dataruns is rare), but for many files or regions of disk this will risk missing many filesystem metadata records.

As filesystem metadata records are highly structured (and often sparse) data, and our filesystem specific parsers run many verification tests, we can understand why our parsers filtered out all of the tested false positive potential timestamps. Further investigations in our Limitations subsection showed that even when running our tools on the encrypted Ext4 image, that we encountered no false positives. The implication is that while potential timestamp carving will allow for greater recall, what ultimately controls the precision are the filesystem specific parsers, and that the precision measured in all cases was 100%.

However, we also showed in the Limitations subsection that by running the parser without potential timestamp information on the disk images took a significantly longer time than the GMTC method. For example, the prefix-based GMTC method took about 35 min to fully run on the encrypted Ext4 image when searching for inodes, whereas running the Ext4 parser alone took about 17.5 h. Applying the prefix-based GMTC method to the encrypted image took 24 min to search for MFT records, while running the NTFS parser directly on the image took 5.75 h.

In terms of time and space both the exact matching and prefix matching GMTC methods are practical. The time taken to carve out potential timestamps on the 476 GB NTFS image was on average just over 2 h. Furthermore, the carving time is not much affected by the change in the prefix length p, as was predicted by the fact that the time complexity of the prefix-based timestamp carving method is the same as the exact timestamp carving method. In general, it appears that as we allow for smaller prefixes in timestamp carving, the time it takes for the filesystem specific parsers to complete increases. The exception to this rule is the NTFS parser for large files, but we believe this to be more of an implementation issue than indicative of a general trend. Even so, the longest time it took for the NTFS parser to scan the Lone Wolf image was about 46 min. Thus, the longest time for total analysis of the 476 GB NTFS image was about 2 h and 48 min (as seen in Table 11).

In summary, there is a performance trade-off that exists for prefix-based Generic Metadata Time Carving, where lowering the prefix parameter p may significantly increase recall, slightly reduces timestamp carving time, but can also significantly increase the filesystem specific parser time due to having the need to validate more potential timestamps.

6. Conclusion and Further Work

In this work, we created and applied a timestamp prefix matching version of the Generic Metadata Time Carving (GMTC) method (Nordvik et al., 2020). The GMTC method can be used to carve for filesystem metadata records from a forensic image without the use of the filesystem, and can potentially allow for full file recovery on a damaged or partially overwritten disk. The crux of our contribution was the prefix-based potential timestamp carving algorithm, that only compares the prefixes of length p as opposed to the entire timestamp. This is because stringologically similar timestamps in most cases should be temporally similar as well. We tested the prefix-based method on three realistic forensic images. Two of the images used NTFS, one of the images used Ext4, and they varied in size from 1 GB to 476 GB.

Our location-based data recovery experiments mostly support our hypotheses. First, we have shown that applying timestamp prefix matching to the GMTC method can produce significantly greater recall in carving filesystem metadata records than the exact timestamp matching version. Surprisingly, performing prefix-based timestamp matching did not appear to affect the precision for carving MFT records or inodes from our test data, as we obtained 100% precision for all of our experiments. Further examinations in the Limitations subsection shows that prefix-based potential timestamp carving will increase the number of potentially valid offsets to metadata timestamps, but it is ultimately the filtering done by the filesystem specific parsers that controls the precision. However, running the parsers on an image without prior potential timestamp information will take significantly longer than using a GMTC method. Using the prefix-based Generic Metadata Time Carving method, the potential timestamp carver essentially performs data reduction of possible MFT record or inode locations for the filesystem specific parsers to check. The method appears to be practical, as our longest experiment on a 476 GB image in total clocked in at about 2 h and 48 min.

Interestingly, changing the size of the matching prefix for the timestamps does not affect the time taken to perform timestamp carving by much. This makes sense as the prefix-based potential timestamp carving algorithm only added a constant number of steps to the original algorithm, therefore producing an algorithm with the same time complexity. Our experiments showed that timestamp carving with lower values of p took slightly less time than experiments with larger p values. On the other hand, reducing the size of the timestamp prefix often greatly increased the time taken by the filesystem specific parsers to extract the metadata records. This is due to the fact that matching for timestamps that are approximately similar results in some magnitudes more of potential timestamps to consider, and thus causing some magnitudes more time to run the filesystem specific parsers. We noted an exception to this rule for the large NTFS image, but this may be due to its implementation and the fact it does not use Python memory mapping libraries as the Ext4 parser does.

Future work for the prefix-based timestamp carving algorithm would be to try to improve its efficiency. Since the algorithm ingests the disk image in a linear fashion, perhaps the efficiency could be improved by using parallel processing to analyze different parts of the disk simultaneously, much like Garfinkel's Bulk Extractor (Garfinkel, 2013). The filesystem specific parsers can also be further optimized.

Our work has shown there needs to be improvements made to the filesystem specific parsers as well, so that they can handle more possible variations to filesystem metadata records. For instance, the NTFS parser needs to be able to handle MFT records with non-resident attributes. For Ext4, there needs to be hard link support, and better support for symbolic links. Then in general, there is also a need for development of parsers of filesystems other than NTFS and Ext4.

Acknowledgement

The research leading to these results has received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project “Ars Forensica - Computational Forensics for Large-scale Fraud Detection, Crime Investigation & Prevention”, grant agreement 248094/O70.

Appendix A. Prefix-Based potential timestamp carving algorithm

Note, we do not include the memory mapping aspects to handle large files. For the full potential timestamp carving program, see:

https://github.com/TimestampPrefixCarving/Peer-Review/blob/main/main.cpp.

```
Input: Raw disk image T as a byte array. Parameters:
m # Length of timestamp;
k # Length of search threshold;
h # Threshold for number of required matching timestamps
  per record;
p # Prefix Length;
Output: Potential timestamp offsets from beginning of
        image (in bytes) in txt file.
i = 0 # Current byte offset from start of image;
while (i < |T| − k) do
    candidateTS = T[i : (i + m)];
    # Boolean holding status of repeated byte sequence
      check;
    repeat = True;
    for b ← 1 to m − 1 by 1 do
        repeat = repeat & (candidateTS[0] ==
          candidateTS[b]);
    end
    # Variable for holding value of a string of characters read
      little-endian and transformed into a numerical value;
    littleEndian = 0;
    if m == 8 then
        littleEndian = (candidateTS[0] <<
          (8 ∗ 0))|...|(candidateTS[7] << (8 ∗ 7));
    else if m == 4 then
        littleEndian = (candidateTS[0] <<
          (8 ∗ 0))|...|(candidateTS[3] << (8 ∗ 3));
    #If candidate timestamp is not a repeated sequence of
      bytes, and the prefix value of littleEndian is not 0;
    if (!repeat & ((littleEndian >> 8 ∗ (m − p)) ! = 0)) then
        matchCount = 0;
        j = i + m;
        while (j < i + m + k) do
            testSequence = 0;
            if m == 8 then
                testSequence = (T[j] <<
                  (8 ∗ 0))|...|(T[j + 7] << (8 ∗ 7));
            else if m == 4 then
                testSequence = (T[j] <<
                  (8 ∗ 0))|...|(T[j + 3] << (8 ∗ 3));
            #Our timestamp prefix matching equivalency
              check;
            xorResult = littleEndian ⊕ testSequence;
            if ((xorResult >> (8 ∗ (m − p))) == 0) then
                matchCount + = 1;
            end
            j + = m;
            if (matchCount >= (h − 1)) then
                #Print Byte Location i;
                j = i + m + k;
                i += (k − m);
            end
        end
    end
    i += m ;
end
```

Algorithm 2: Prefix-based potential timestamp carving algorithm, using timestamp prefix matching for the timestamp equivalency test.

Algorithm 2. Prefix-based potential timestamp carving algorithm, using timestamp prefix matching for the timestamp equivalency test.

References

Aho, A.V., Corasick, M.J., 1975. Efficient string matching: an aid to bibliographic search. Commun. ACM 18 (6), 333–340.

Bayne, E., Ferguson, R.I., Sampson, A., 2018. Openforensics: a digital forensics gpu pattern matching approach for the 21st century. Digit. Invest. vol. 24, S29–S37. The Proceedings of the Fifth Annual DFRWS Europe. https://www.sciencedirect.com/science/article/pii/S1742287618300379.

Boyer, R.S., Moore, J.S., 1977. A fast string searching algorithm. Commun. ACM 20 (10), 762–772.

Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley Professional.

Casey, E., Nelson, A., Hyde, J., 2019. Standardization of file recovery classification and authentication. Digit. Invest. 31, 100873. URL. http://www.sciencedirect.com/science/article/pii/S1742287618304602.

Cowen, D., Seyer, M., 2013. File System Journal Analysis. SANS DFIR. https://digital-forensics.sans.org/community/summits.

Dewald, A., Seufert, S., 2017. Afeic: advanced forensic Ext4 inode carving. Digital investigation 20. In: The Proceedings of the Fourth Annual DFRWS Europe, pp. S83–S91. URL. http://www.sciencedirect.com/science/article/pii/S1742287617300270.

Ext4 (and Ext2/Ext3) Wiki, aug 2019. Ext4 disk layout. URL. https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout.

forensicswiki.xyz, sep 2012. File Carving. https://forensicswiki.xyz/wiki/index.php?title=File_Carving.

Garfinkel, S.L., 2007. Carving contiguous and fragmented files with fast object validation. Digit. Invest. 4, 2–12. URL. http://www.sciencedirect.com/science/article/pii/S1742287607000369.

Garfinkel, S.L., 2013. Digital media triage with bulk data analysis and bulk_extractor. Comput. Secur. 32, 56–72. URL. http://www.sciencedirect.com/science/article/pii/S0167404812001472.

Hamming, R.W., 1950. Error detecting and error correcting codes. The Bell Syst. Tech. Journal. 29 (2), 147–160.

Jan KillDisk, Active, 2021. How To…for Killdisk Software. URL. https://www.killdisk.com/load-bootdisk-win10.htm.

Knutson, T., Carbone, R., 2016. Filesystem Timestamps: what makes them Tick?, 11. GIAC GCFA Gold Certification.

Levenshtein, V.I., 1966. Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet Physics Doklady, vol. 10, pp. 707–710.

Maar, R., 2014. Ext4magic. URL. https://github.com/gktrk/ext4magic.

McCash, J., 5, 2010. Timestamped Registry & NTFS Artifacts from Unallocated Space. URL. https://digital-forensics.sans.org/blog/2010/05/04/timestamped-registry-ntfs-artifacts-unallocated-space.

Moore, Garfinkel, Farrell, Roussev, Dinolt, 2018. Lone Wolf Scenario. Last visited: 2020-04-16. URL. https://digitalcorpora.org/corpora/scenarios/2018-lone-wolf-scenario.

Mueller, L., 1 2008. Search for Windows 64 Bit Timestamps. URL. http://www.forensickb.com/2008/01/search-for-windows-64-bit-timestamps.html.

NIST, mar 2017. Computer Forensic Reference Data Sets: Deleted File Recovery. URL. https://www.cfreds.nist.gov/dfr-test-images.html.

Nordvik, R., Georges, H., Toolan, F., Axelsson, S., 2019. Reverse engineering of ReFS. Digit. Invest. 30, 127–147. URL. http://www.sciencedirect.com/science/article/pii/S1742287619301252.

Nordvik, R., Porter, K., Toolan, F., Axelsson, S., Franke, K., 2020. Generic metadata time carving. Digital Investigation. In: The Proceedings of the Twentieth Annual DFRWS USA, vol. 33, p. 301005. URL. https://www.sciencedirect.com/science/article/pii/S2666281720302547.

Plum, J., Dewald, A., 2018. Forensic APFS file recovery. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. ARES 2018. ACM, New York, NY, USA. https://doi.org/10.1145/3230833.3232808, 47:1–47:10. URL.

Richard III, G.G., Roussev, V., 2005. Scalpel: a frugal, high performance file carver. In: Proceedings of the Digital Forensic Research Conference, 2005.

Schicht, J., 2018. Logfileparser Readme. Github. URL. https://github.com/jschicht/LogFileParser.

Son, N., Lee, Y., Kim, D., James, J.I., Lee, S., Lee, K., 2013. A study of user data integrity during acquisition of android devices. Digital Investigation 10. S3 − S11. In: The Proceedings of the Thirteenth Annual DFRWS Conference. URL. http://www.sciencedirect.com/science/article/pii/S1742287613000479.

TWRP, May 2019. Download twrp-3.3.1-2-dreamlte img.tar. Https://eu.dl.twrp.me/dreamlte/twrp-3.3.1-2-dreamlte.img.tar.html.

Vidas, T., Zhang, C., Christin, N., 2011. Toward a general collection methodology for android devices. Digit. Invest. vol. 8, S14–S24. The Proceedings of the Eleventh Annual DFRWS Conference. http://www.sciencedirect.com/science/article/pii/S1742287611000272.

Zha, X., Sahni, S., 2010. Fast in-place file carving for digital forensics. In: International Conference on Forensics in Telecommunications, Information, and Multimedia. Springer, pp. 141–158.