Digital Investigation 28 (2019) S60–S67

Contents lists available at ScienceDirect

Digital Investigation

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

Deleted file fragment dating by analysis of allocated neighbors

Ahmed A. Bahjat*, Jim Jones

George Mason University, Fairfax, VA, United States

ARTICLE INFO

Article history:

Keywords:
Digital stratigraphy
Computer forensics
Digital evidence
File slack
Deleted file
Event reconstruction

ABSTRACT

Timestamps play a substantial role during digital forensic investigations and address two main objectives. First, they serve as a primary culling criterion to reduce the amount of digital evidence subject to analysis. Second, timestamps are the sole feature that allows reliable reconstruction of time-lines and they assist in locating temporal anomalies. File fragments, typically from previously deleted or relocated content, are often useful, especially when intact files are unavailable. Such fragments rarely contain embedded timestamps or have file-system timestamp information, which renders them less useful. In this work, we investigate and propose a framework for determining a time-window for deleted file fragments that are typically found in un-allocated space and file slack. We hypothesize that using the known temporal state of neighboring clusters allows us to derive a date-and-time range for when the file fragment was first written to media until it was subsequently deleted.

Introduction

In order to reconstruct events, digital forensic examiners seek every piece of evidence available, including full or partial files which have been deleted or deallocated. File carving — extracting full or partial files from un-allocated space — has been studied extensively in the past ten years (Silberschatz et al., 2012). In many cases, the successful recovery of a file is infeasible (if fragmented) or impossible (if partially overwritten). However, remnants of these files — as small as a single identifiable sector — may still allow for a reconstruction of a time-line and lead to a probable piece of evidence. Buchholz and Spafford (2004), suggest that timestamps are invaluable meta-data to record and hunt for in order to answer the essential "when did what happen?" question. In addition, they allow an ordering of the file operations into a timeline. As a simple scenario, consider a case where a specific website was claimed to have been visited. An indication of this visit often appears in Cookies, history files, and the system registry which are stored in clear text under default settings. Remnants of these files' contents may still exist in the un-allocated space of the hard drive — even if the original file has been deleted. Similarly, remnants of these files may exist in a file slack (Carrier, 2005), which refers to the unused remaining sectors of an allocated file's storage area. Depending on the file system format, a file slack can be as big as 60 Kilo-Bytes or larger. After finding a file fragment, we need to contextually place the evidence in a timeline to draw a complete picture of the case. In the scenario above, searching for text — e.g. a URL — solves half of the story but without a plausible timestamp the story is incomplete. With a valid timestamp the question of when the file was created or deleted can be answered. File systems typically log at least a few useful dates about each file, including: created date, last modified date, last accessed date, and the file record changed date. This information is lost when the file is deleted and its record is then allocated for another file. As a first step, if the cluster where the fragment was found is a file slack of an existing file, we can consider the new file created time or its allocation time to be an upper-bound of the evidence lifespan under certain conditions. This means that the URL was surely visited before it was later overwritten. This is not always a straightforward reading, as we will see in this work, due to the way these dates are updated. Furthermore, by looking into timestamps of adjacent allocated files, we can infer a lower-bound date for when the evidence fragment was originally created. The two bounds defines the time-window of a given file. The upper-bound date approximates when the file was last seen before it was deleted. Similarly the lower-bound date approximates the last allocation date for the file.

Digital stratigraphy is a new sub-field emerging in digital forensics that studies file system traces and writing patterns to discover anomalies and behaviors (Casey, 2018). Stratigraphic dating is a well known method in archaeology that studies geographical strata and predicts dates for natural artifacts based on

* Corresponding author.
E-mail addresses: abahjat@gmu.edu (A.A. Bahjat), jjonesu@gmu.edu (J. Jones).

https://doi.org/10.1016/j.diin.2019.01.015

known dates and the principle that upper layers are younger than the layers underneath. Prior research has addressed related problems, such as data fragments classification (the file type where these fragments originally existed). Poisel et al. (Poisel et al., Tjoa) summarized and grouped this body of work into five approaches to data fragment classification. Three of them stand out, namely, statistical-based approaches that infer a class from statistical features of the data universe; signature-based approaches that identify a fragment based on prior knowledge of fragment content; and machine-learning based approaches that use supervised or unsupervised learning algorithms to classify data fragments into categories of files. However, the usefulness of classifying data fragments is limited if the investigator fails to transform these fragments into a logical and often a temporal scenario to support or refute a hypothesis.

Our initial investigation as well as studies by Casey (2018) show that a file allocation scheme translates into a disk writing pattern for each file system. Understanding these patterns through temporal logic allows us to place fragments into a probable timeframe. To further narrow down this timeframe window, we are working to model the file system's writing pattern that allows for the prediction of a discrete timestamp for each cluster in the file system.

Related works

Our work complements the work in file fragment identification. For example, identifying uninstalled applications is a very useful tool in the investigation arsenal. Jones and Khan defined a practical method to discover evidence of deleted applications by performing a sector-by-sector matching of hard drive under investigation to a pre-constructed sectors' catalog of applications (Jones and Khan, 2017). Our work helps to identify when applications were uninstalled. Other relevant work on this problem can be divided into three broad categories: the role of time in digital forensics, event reconstruction and analysis, and work in file fragment identification and recovery.

In the role of time, James Allen can be considered one of the first scholars who wrote about time in computing (Allen, 1983). In his work, Allen emphasized the temporal information logic in various computing problems and paved the road for the event reconstruction work. Allen published a time representation technique as well as a reasoning system about temporal intervals (Allen, 1991). In a more specific research to digital forensics, Boyd and Foster outlined a case study that shows how misinterpretation of timestamps can lead to a wrong accusation (Boyd and Forster, 2004). Great care has to be taken in at least three categories of concerns; the contextual meaning of timestamps, time zone interpretation and conversion, and clock synchronization (Buchholz and Tjaden, 2007; Lamport, 1978). Furthermore, Chow et al. (2007) studied and reported a list of observations about system times in the NTFS file system. Their work is very useful for any investigator dealing with time to understand the relationship between system dates and user actions. One should take caution when dealing with the observations in (Chow et al., 2007) as not all of them were true at least from our experimentation as we will show in this paper.

The work in event reconstruction started more than forty years ago. Event reconstruction is a particularly challenging task due to the lack of reliable tamper-proof source of time in addition to the challenges of automating the reconstruction process. Leslie Lamport, studied events ordering in distributed systems (Lamport, 1978). Her research is a cornerstone in event reconstruction; it defines the "Happens before" relationship between two events based on message correspondence between two processes. Furthermore, Jeyaraman and Atallah (2006) empirically studied the effectiveness of automating event reconstruction and they found that most systems proposed in the literature have a very large false positive rate. Similarly, In (Hargreaves and Patterson, 2012) Hargreaves and Petterson propose a less verbose and aggregated timeline construction approach from a the large set of low-level events generated on a typical hard drive use such as file created, modified, or deleted while still maintaining a trusted chain to the originating event. The aggregated events are high-level events such as Skype call, Google Search, or USB Connected. Low-level events are extracted mostly from the file system i-node store i.e. $MFT record in NTFS file system. Other dates are extracted from the System Registry hives. Events sources can be heterogeneous and expands beyond events logged the system events or transaction journals. Aggregating these events can leads to knew knowledge about high level events (Chabot et al., 2014), but may still be missing important evidence when there is no trace to the low-level events — i.e. when traces to fragments are lost due to the rolling write on system logs. Schatz et al. (2006) proposed a method that correlates system timestamps with other corroborating sources of time connected to the computer in question — i.e. Proxy server — to infer the temporal behavior of a particular computer. This approach assumes there is a connected computer and cannot be applied to an isolated system. However, a finite-state machine approach can be used for isolated system (Gladyshev and Patel, 2004). Visualizing and analyzing automated timelines can be daunting task most of the times, for this Inglot and Liu built an enhanced analysis tool and framework to filter, group, and visualize timelines (Inglot et al., 2012; Inglot and Liu, 2014).

Since our focus addresses fragments in file slacks, file fragment classification work is also of interest. If a fragmented file can be recovered, then dating will be straightforward if the file has internal dates — e.g. a MS Office file or a PDF. Several other research has been done on fragment classification (Poisel et al., Tjoa) including classification using machine learning (Beebe et al., 2013), carving fragmented files (Garfinkel, 2007), and statistical analysis of fragment types (Roussev and Garfinkel, 2009; Veenman).

Background

The primary goal of this publication is to propose a framework for file fragment dating. We define file fragment dating as a process of determining an age for a given file remnant. In archaeology science, there exists two types of dating. Absolute dating is a collective term for techniques that assign specific dates or date ranges in calendar years to artifacts and other archaeological finds. The second type is relative dating which is a technique of assigning an order of events or artifacts relative to other events or artifacts.

File systems

File systems can be seen as levels of abstraction implemented by the operating system. These abstractions mask the complexity of hard drive system calls to open, create, append, or delete files as well as managing the file space efficiently and effectively. For a file system to do its job effectively, it has to keep track of files and their relationship to the hard drive and to each other. In addition, it keeps track of changes that happen to each file in order to provide a reasonable recovery from errors.

An NTFS file system maintains track of all file locations in a system file named $MFT. This file stores a file record of each file created in the hard drive that includes information about its allocated clusters, the parent directory, permissions to the file, attributes, and system dates. Each file record has a set of four date types stored in two attributes to make a total of eight dates: four dates in the $STANDARD_INFO attribute of the file and four in the $FILE_NAME attribute. The four date types are the Date Modified (M),

Date Accessed (A), Date Created (C), and Date MFT Entry Changed (E) — collectively referred to as MACE. We refer the four set of dates as MACE for brevity. In general, the set of four dates in the $FILE_NAME attribute are often overlooked since they are rarely modified are most likely found out of sync with the dates in the $STANDARD_INFO. However for dating purposes they are more a appealing, and in fact reliable, source for dating especially when the Created (C) date is under suspicion. The dates in the $FILE_NAME are less frequently touched and, therefore, they are more reliable in providing a true creation time for a file. Another and more important aspect of dates in the $FILE_NAME attribute is that they are only modifiable by the system kernel. There is no known anti-forensics utility that can modify these dates. This is not the case for dates in $STANDARD_INFO, as there are tools i.e. Timestomp that can modify them (Cho, 2013; Casey, 2011).

In addition to the $MFT table, NTFS file system maintain a record of each changed file in a file named $USNJrnl and a more detailed transaction oriented log in $LogFile. It is worth noting that system date updates are counter-intuitive in many cases. The NTFS file system for example, can modify the file without updating the file access date. Furthermore, the file access date does not always reflect user access; an anti-virus scan updates the access date when it scans it. Moreover, the defragmentation process does update the access date when it moves a file from one cluster to another according the caching and last access updated policy. Furthermore, many file systems maintain a tunnel cache that is used to store file records to be used if a deletion and a subsequent creation occurs within a short period of time defined by the operating system configuration (Casey, 2018). If file tunneling is enabled, the file record in the $MFT is reused for the new file without changing any of the dates except the C date. In Table 1 we report our own investigation results in what user operations affect the file system dates in the $MFT.

Digital fragments

A digital fragment is a remnant of a deleted file that resides in one or more contiguous sectors of a hard drive. It is worth noting that a file may leave multiple remnants if the file was fragmented or occupied more than one cluster. Each contiguous remnant is considered an independent fragment for our dating purposes. Finding digital fragments is a common practice in digital forensics and can be achieved in several ways. The most common way is through text searching as most forensics tools allows searching the entire hard drive for specific word, byte, or string. The second way of finding fragments is through utilizing sector hashing. An example of fragment hash approach is implemented by Jones and Khan to find evidence of uninstalled applications (Jones and Khan, 2017).

Slack space

Slack spaces exists in various forms. The two most common forms are the volume/partition slack and file slack. Volume slack is the un-allocated space left after creating a hard drive partition. File slack occurs in files that do not fully align with a multiple of a cluster size. The default cluster size for NTFS file system is 4 Kilo-Bytes (KB) for up to 16 Tera-Bytes hard drive. The size of the cluster needs to be larger for lager hard drives (e.g for 130 TB drive, the default cluster size is 128 KB). As an example, if a file system with a 4 KB cluster size needs to allocate a space for 50 KB file, then 13 clusters — total size 52 KB — will be allocated leaving 2 KB of unused space. Out of the available space, 100 sectors will be occupied by the file and 4 sectors of size 512 bytes are left as file slack. Some operating systems use padding to zero out file slacks. Windows uses padding only for the last sector used for a file. Furthermore, modern hard drives are moving to a 4 KB sector size - a.k.a Advanced Format-in order to reach higher capacities.

Dating model

Operating systems deal with hard drives at a file system level. File allocation algorithms are of special interest to our research since they determine where files are placed relative to the available sectors in a hard drive. There are three major allocation methods described in the literature: contiguous allocation where no fragmentation is allowed, linked allocation in which space can be scattered and pointers are maintained to allocated blocks, and indexed allocation in which pointers to allocated blocks are maintained in a central repository. Fig. 1 shows a snapshot of the allocation after running an automated creation of text files on a heavily used FAT32 drive. The data show how the file system allocated the clusters on the end of the hard drive sectors [1,026,422−1,049,990] before it allocated the next available sectors − 408,326 − seeking from the beginning of the drive (see Fig. 2).

File system allocations are ignorant of the actual physical allocation managed by the hard drive firmware. Physical allocation is often times a proprietary algorithm owned by hard drive manufacturers. Furthermore, the physical allocation might change after the file has been allocated at any time for performance reasons depending on the type of media (e.g. mechanical disk or flash-based storage), but this detail is still hidden from the file system. From our experimentation, both NTFS and FAT32 file systems uses first available cluster algorithm on a brand new hard drive. However, after the file system is almost full, the NTFS system starts to allocate files differently than the FAT file system. This is because the allocation algorithms in NTFS system is a best fit allocation which behaves differently than the next available allocation that the FAT file system uses. This means there is a strong positive correlation between the cluster number and the file allocation time. This correlation goes astray after heavy use of the hard drive and after system defragmentation operations. In the case of hard drive defragmentation, the mapping between the file record number in the index table and the cluster number changes. This is because the primary system call used during a defragmentation process is the Move operation (FSCTL_MOVE_FILE).

Table 1
System dates.

Location	Date	Operations
$STD_INFO	M	modify, create
	A	Volume move, copy, open, create, defragmentation, anti-virus scan
	C	copy, create
	E	rename, volume or local move, copy, modify, create
$FILE_NAME	M	Volume move, modify, create,
	A	Volume move, copy, open, create
	C	copy, create
	E	Volume move, rename, volume or local move, copy, modify, create

Drive E:

\Test

Name	Size	Created	Modified	Attr.	1st sector
.. = (Root directory)	4.0 KB				3,838
. = Test	8.0 KB	03/21/2017 15:51:23.7	03/21/2017 15:51:24		712,790
file_161	1.9 MB	03/21/2017 17:49:41.9	03/21/2017 17:49:48	A	1,026,422
file_162	1.7 MB	03/21/2017 17:49:46.0	03/21/2017 17:49:50	A	1,030,278
file_163	1.6 MB	03/21/2017 17:49:49.5	03/21/2017 17:49:54	A	1,033,678
file_164	1.2 MB	03/21/2017 17:49:53.1	03/21/2017 17:49:56	A	1,036,870
file_165	576 KB	03/21/2017 17:49:55.6	03/21/2017 17:49:58	A	1,039,246
file_166	55.8 KB	03/21/2017 17:49:57.0	03/21/2017 17:49:58	A	1,040,406
file_167	572 KB	03/21/2017 17:49:57.2	03/21/2017 17:50:00	A	1,040,518
file_168	1.7 MB	03/21/2017 17:49:58.2	03/21/2017 17:50:04	A	1,041,670
file_169	548 KB	03/21/2017 17:50:02.0	03/21/2017 17:50:04	A	1,045,062
file_170	1.9 MB	03/21/2017 17:50:03.1	03/21/2017 17:50:08	A	1,046,158
file_171	0.8 KB	03/21/2017 17:50:07.5	03/21/2017 17:50:08	A	1,049,982
file_172	1.8 MB	03/21/2017 17:50:07.7	03/21/2017 17:50:12	A	1,049,990
file_173	1.2 MB	03/21/2017 17:50:12.2	03/21/2017 17:50:16	A	408,326
file_174	1.3 MB	03/21/2017 17:50:15.5	03/21/2017 17:50:20	A	410,862
file_175	1.8 MB	03/21/2017 17:50:18.7	03/21/2017 17:50:24	A	413,590
file_176	1.9 MB	03/21/2017 17:50:23.4	03/21/2017 17:50:30	A	430,630
file_177	1.0 MB	03/21/2017 17:50:28.6	03/21/2017 17:50:32	A	434,422
file_178	709 KB	03/21/2017 17:50:30.5	03/21/2017 17:50:34	A	436,470
file_179	1.5 MB	03/21/2017 17:50:32.5	03/21/2017 17:50:38	A	437,894
file_180	444 KB	03/21/2017 17:50:36.1	03/21/2017 17:50:38	A	441,062
file_181	1.9 MB	03/21/2017 17:50:37.2	03/21/2017 17:50:42	A	441,950
file_182	1.0 MB	03/21/2017 17:50:41.7	03/21/2017 17:50:44	A	445,838
file_183	1.4 MB	03/21/2017 17:50:43.9	03/21/2017 17:50:50	A	447,894

Fig. 1. Next available allocation.

\Program Files\AVG\AVG9 — 8 min. ago

Name	Ext.	Size	Created	Modified	Record changed	Attr.	1st sector
.. = AVG		144 B	11/08/2009 19:57:24.8	11/08/2009 19:57:24.8	11/08/2009 20:19:51.0	I	6,314,904
. = AVG9		32.1 KB	11/08/2009 19:57:24.8	11/12/2009 12:19:06.5	11/12/2009 12:19:06.5	I	14,334,760
setup.dat	dat	2.2 MB	11/09/2009 11:32:49.5	11/09/2009 11:32:49.7	11/09/2009 19:38:40.0	A	1,930,432
avgfws9.exe	exe	2.2 MB	11/09/2009 11:31:57.6	11/09/2009 11:31:57.8	11/11/2009 15:00:22.8	IA	14,705,464
avgresf.dll	dll	2.2 MB	11/08/2009 19:58:16.8	11/08/2009 19:58:17.0	11/08/2009 19:58:17.0	A	1,786,216
avguiadv.dll	dll	2.3 MB	11/09/2009 11:33:02.9	11/09/2009 11:33:04.1	11/09/2009 19:38:53.1	A	14,906,088

Drive S: Drive Z:

\Program Files\AVG\AVG9 — 9 min. ago

Name	Ext.	Size	Created	Modified	Record changed	Attr.	1st sector
.. = AVG		144 B	11/08/2009 19:57:24.6	11/08/2009 19:57:24.6	11/08/2009 19:57:24.6	I	14,673,992
. = AVG9		16.4 KB	11/08/2009 19:57:24.7	11/08/2009 19:57:24.7	11/08/2009 19:57:24.7	I	14,628,768
avgtray.exe	exe	1.9 MB	11/08/2009 10:36:17.2	11/08/2009 10:36:17.2	11/08/2009 10:37:53.0	IA	10,695,952
setup.dat	dat	2.2 MB	11/08/2009 20:30:07.7	02/24/2005 22:35:05.0	11/11/2009 15:04:46.8	A	10,092,584
avgresf.dll	dll	2.2 MB	11/08/2009 10:36:12.6	11/08/2009 10:36:12.6	11/08/2009 10:37:53.0	A	10,733,616
avguiadv.dll	dll	2.3 MB	11/08/2009 10:38:04.0	11/08/2009 10:38:04.0	11/08/2009 10:38:04.0	A	1,719,168

Fig. 2. File re-allocation example after a software update operation.

When a file in NTFS file system is deleted, its file record in the $MFT table is marked deleted and the corresponding clusters are marked available in the system $Bitmap. The deletion event is recorded in the transaction journals but none of the dates change in the $MFT drive (see Fig. 5). Therefore, at this point, no dating is required and the file can be fully restored. There is no guarantee on how long the deleted file can stay intact. Two types of overwriting can happen: first, the file record in the $MFT is allocated to a different file in which the pointer to the deleted file is lost; second, the available clusters are later allocated for a different file which results in partial or complete overwriting of the file content. However, in the first type, the data is still recoverable by creating a new pointer to the data. The new pointer will allow a recovery of the file content but will create new system dates as a new file record has to be created. Our dating scheme can help in recovering a date for this type of scenario in addition to the fragment dating. It is worth noting, that some files, i.e PDF, Office Files, or email types, have internal dates that in many cases are sufficient for dating and a recovery of the system dates may not add much value. In the second type of overwriting, the system dates are recoverable from the file record although the data might be lost partially or completely. Note that, the first and the second types can both happen in the worst case scenario where the data and its pointer are both lost, which is not unusual. Our recovery scheme is also very handy in these situations.

Contrary to Observation 2 in (Chow et al., 2007), we found that

moving a file from one partition to another — even within the same physical drive — will not change the creation time in $Std_Info but will change all dates in $FILE_NAME to the new creation time. We tried this with a move command and with cut and paste through the file browser in both a Windows 7 and a Windows 10 systems.

Fig. 3 shows how defragmentation changes the allocated sector as well as the Last Date Accessed. It is important to note that the Date accessed is not updated right away, and can take up to 24 h to be written to the file record. Windows caches the accessed date for performance reasons; and this feature can be disabled and the date accessed is then left without being updated.

Fig. 4 shows how a file can be modified and have a modified date updated but the accessed date is not.

For the dates anomalies described above, it is important that we look on all the dates on a file to determine last activity. Furthermore, to start dating a file fragment in a given system, we should start by determining the logical dating boundaries. A logical upper-bound for our ranges is the collection date. Although this is intuitive but it is an important step to consider and use for verification. Similarly, the logical lower-bound for all files must be the first record creation date. In NTFS system the first record is the $MFT file itself. This lower-bound can further be verified by looking into the operating system files create dates. This lower-bound is the latest date the drive was formatted. A drive could have been formatted multiple times; to reach a more robust lower-bound in case of multiple formatting can be extended to the drive manufacturing date.

Although $FILE_NAME dates are rarely updated which makes them less useful than the dates on $STANDARD_INFORMATION dates but we can use this to our advantage. Fig. 5 shows three snapshots: the first is taken before the file is deleted; the second is after the file is deleted and, therefore, moved to Recycled bin; and the third is after the Recycled bin is emptied. The only date that indicates the deletion is the E date as shown in the figure only if the file is moved to the recycled bin and does not get changed with a hard deleted or when the file is no longer in the recycle bin.

Experiment design

We hypothesize that adjacent files have correlated system dates. In the following details, we propose a set of conjunctures with evidence based on our empirical study. In order to test our hypothesis, we applied the dating logic on a subset of the M57 Patent digital corpora built by the Naval Postgraduate School (Garfinkel et al., 2009). The M57 data set includes hard drives snapshots over a 17-day period between Nov, 11 2009 and Dec 11, 2009

Name	index.dat	index.dat
File Class	Regular File	Regular File
File Size	311,296	376,832
Physical Size	311,296	376,832
Start Cluster	2,283,451	2,283,451
Date Accessed	11/12/2009 8:14:05 PM	11/12/2009 8:14:05 PM
Date Created	11/12/2009 8:14:05 PM	11/12/2009 8:14:05 PM
Date Modified	11/19/2009 5:58:02 PM	11/20/2009 11:25:01 PM

Fig. 4. Modified file but accessed not updated.

collected from a fictitious patent research institute (the actual duration is 33 days according to the dates found in the dataset [Table 4]). We chose this data set because we can extract the ground truth of when files were deleted, by looking into missing files in the following snapshot to any given one. For our experiment, we only used the institute CEO's drives (Pat's drives). Starting from the second snapshot taken on Nov 16, 2009, we parsed several system properties, including the set of eight dates of each deleted or relocated file. The full list of data collected is listed in Table 2. In this step we check each file in the current snapshot for its existence in the same location in the prior snapshot. If the file does not exist we add it to our evidence list as a fragment artifact and then parse its meta-data. Lastly, we collect the meta-data of all files in the last snapshot, we declare the last snapshot as the forensic snapshot. The forensic snapshot is what will be available in a typical investigation, also known as the duplication drive. Table 3 list the properties of the forensic snapshot. There was 34813 files in the hard drive, 1483 of the files were non-resident deleted files and their sector were allocated for other files. Other properties are not used but give a context on the distribution of the files based on their allocation type.

Our experiment has two phases, the data collection phase and the analysis phase. For our data collection, we developed a C# application with native code using a wrapper of DLLImport < Kernel32.dll > to be able to read the $MFT file as well as reading file data from a specific sector. Next, we store the results in two SQL server tables, the first table holds the meta-data of the evidence list and the second table holds the forensic snapshot meta-data. Using the two tables, we built an application to read and populate lower and upper dates for each evidence file. Before we can start predicting dates let us denote a TrueDate as a date approximating the actual create date for a file. This date is the file create date (C) taken from the $FILE_NAME property only if this date is greater than the $MFT (E) date. Otherwise, the TrueDate is the M date from the

Name	avgupd.dll	avgupd.dll
File Class	Regular File	Regular File
File Size	1,657,112	1,657,112
Physical Size	1,658,880	1,658,880
Start Cluster	1,627,222	317,645
Date Accessed	11/9/2009 4:31:29 PM	11/20/2009 5:07:02 PM
Date Created	11/9/2009 4:31:29 PM	11/9/2009 4:31:29 PM
Date Modified	11/9/2009 12:58:02 AM	11/9/2009 4:31:28 PM
Encrypted	False	False
Compressed	False	False
Actual File	True	True
Start Sector	13,017,776	2,541,160

Fig. 3. File De-fragmentation.

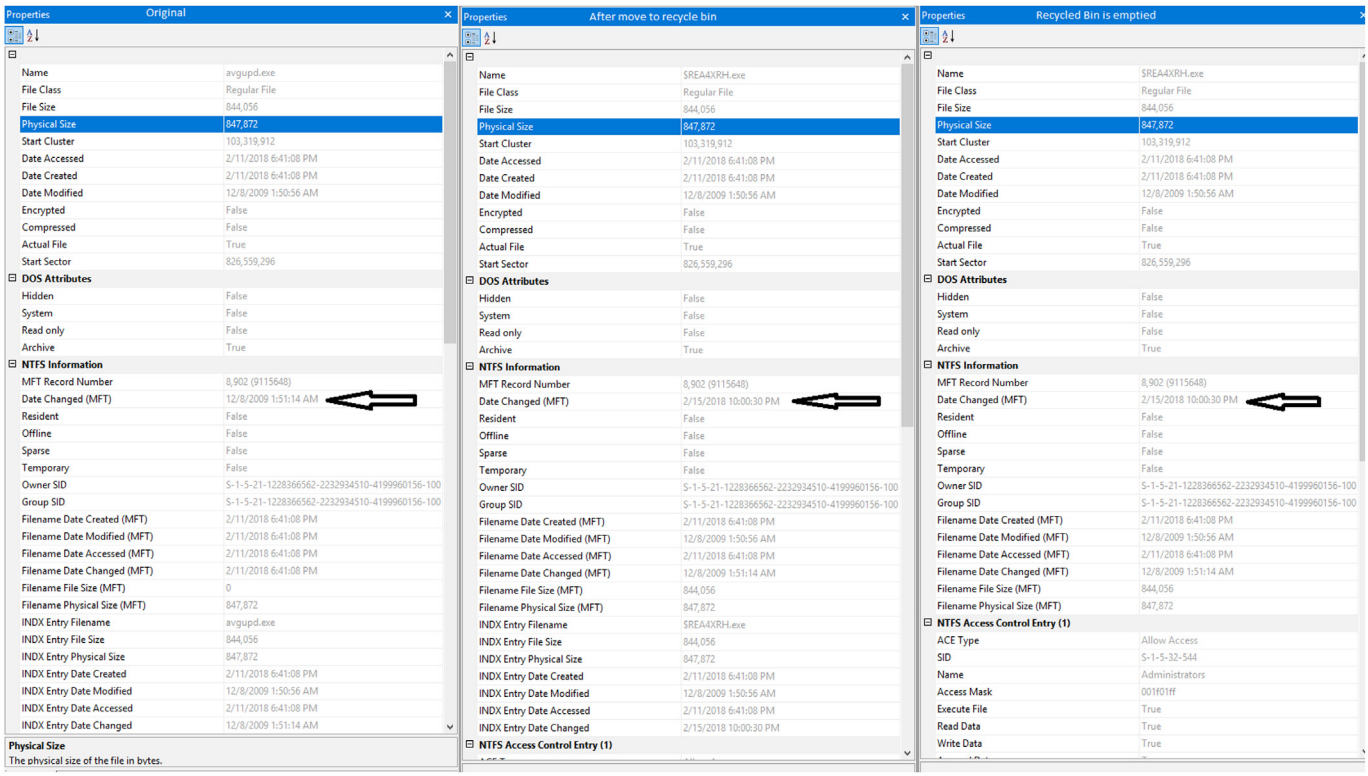

Fig. 5. File after it has been deleted and the Recycle Bin emptied.

Table 2
Dataset attributes.

Artifact	Attributes
Deleted File	Name,Size, SectorNumber, Fragments Count, MFT Index, MFT SeqNumber, Std_MACE, FileName_MACE
Slack-Owner File	Name,Size, MFT Index, MFT Sequence Number, Std_MACE, FileName_MACE
Previous File	Name,Size, MFT Index, MFT Sequence Number, Std_MACE, FileName_MACE
Post File	Name,Size, MFT Index, MFT Sequence Number, Std_MACE, FileName_MACE

Table 3
Deleted files data set.

Property	File Count
All Files	34813
Resident files	5369
Non-resident files	29444
Fragmented files	7019
Total deleted or relocated files	4961
Total resident	1287
Total non-resident	3674
Total non-resident allocated	1483
Total non-resident non-allocated	1424
Total non-resident error-ed	767

$FILE_NAME property. As explained in the previous section this is done to account for files brought in from an external source or another partition.

The upper-bound date for a file fragment found in a file slack is the maximum of the eight dates found in the Slack-Owner (the file allocated to that space). We take the file occupying the first sector of the evidence file to be the Slack-Owner file and, therefore, its maximum date is the upper date.

The lower-bound date is taken from the K-nearest neighbors by sector similarity; after experimenting with several number of K for neighbors, we found that 10 neighbours are sufficient to calculate the average and the average does not drop significantly by adding more nodes. This can change and has to be evaluated on a larger set. Specifically, we take the sector number of each record in the evidence table and query the forensic drive table to find the neighbors. We ignore neighbors with TrueCreate > the upper-bound date. We then calculate the average TrueCreate date as the predicted lower-bound date for the evidence file.

Results and discussion

Table 4 shows the range of dates found in the last snapshot in the following format: 'MM/DD/YY HH:MM'. One can see from the dates shown in the table that the last accessed date can sometimes be corrupt. In our data set, we found 8 files, 3 of which are resident files, from the Java Run-time Environment (JRE) that have a last accessed date of '11/10/2073'. We re-installed the same installer found on that hard drive but the last accessed date were the date of the installation; which indicate a date forged or corrupted.

Slack-Owners and fragments found within its slack have an interesting relationship. Intuition may lead one to think that a Slack-Owner's last accessed date is always greater than the fragment's last accessed date. This assumption turns out to not always be the case in NTFS systems for the following reasons: file tunneling, downloading a file over the web without accessing the

Table 4
System dates.

Location	Date	Minimum	Maximum
$STD_INFO	M	2/8/99 14:08	12/11/09 17:11
	A	7/29/08 0:53	10/11/73 12:21
	C	2/8/99 14:08	12/11/09 17:11
	E	11/8/09 15:36	12/11/09 17:11
$FILE_NAME	M	4/14/08 12:00	12/11/09 17:11
	A	8/25/09 9:17	10/11/73 12:21
	C	1/22/06 0:01	12/11/09 17:11
	E	8/11/99 9:00 0:14	12/11/09 17:11

file subsequently, defragmenting the file system where the system is set to disable last accessed updates, or acquiring – performing the forensics collection – the hard drive before the operating system updates the file records (as this can take up to 24 h).

In Fig. 6 we show the accuracy of the upper-bound prediction under various confidence levels in days. The accuracy reaches above 90% when the threshold is set to five days or more — most of the data were in the ±5 days from the actual deletion date. Similarly, Fig. 7 shows the lower-bound accuracy given a confidence level in days. The lower-bound prediction is more difficult due to the lack of a robust date that we can pivot on. The lower-bound date is calculated as the average create date of the neighbouring files preceding the Slack-Owner in sector allocation as explained in the previous section. These accuracy graphs show how a flexible confidence level would result in a more accurate prediction until we have a full coverage of all deleted files with a range close to the logical boundaries.

We know from our experimentation which is supported by Chow's findings (observation 1,3 and 7) in (Chow et al., 2007), that if the file create date equals the modified date then the file is intact and has not been updated. These observations do not suggest the contextual meaning of the time. In fact, the creation time and modified time do not always refer to the actual creation date on the file system. Extracted files will always keep their original create and modified date. In our data-set for example, Python26 was downloaded to Pat's drive and it appears in the snapshot taken on 11/16/2009. The create/modified dates for the original compressed installer at Download folder was dated 11/13/2009 at 2:37AM, which was then expanded after installation to the root folder of the Python26. By investigating the Python26 dates we can see the folder was created during 1 min after the file was downloaded, but the dates on the decompressed files reflect a 11/26/2009 date.

Our analysis shows an accuracy of at least 94.26% in bounding the non-resident allocated files. The average lower-bound is (−12),

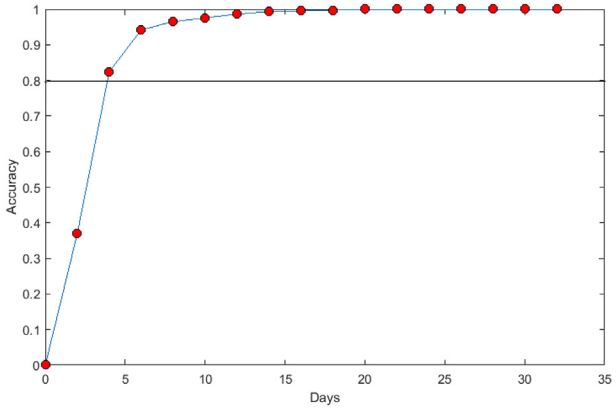

Fig. 6. Upper-bound Accuracy with various threshold.

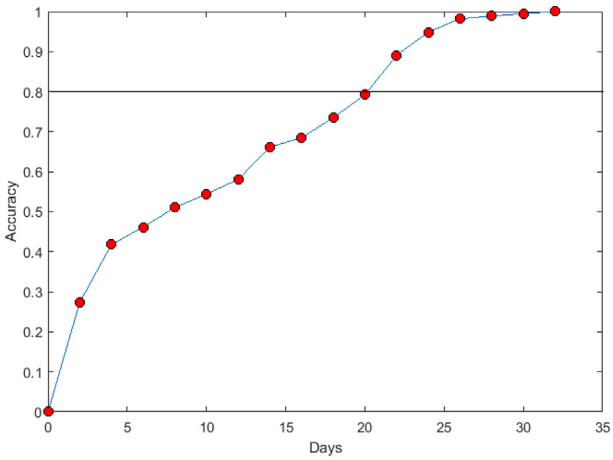

Fig. 7. Lower-bound Accuracy with various threshold.

that is 12 days before the last evidence of existence or its deletion. The average upper-bound is 3 days after the last evidence of existence. Our analysis was performed on a system where the last access date update is enabled. This is the default on Windows but it can be changed using the registry file or FSUTIL command ("fsutil behavior set disablelastaccess 1″). The forensic examiner can check the registry hive on the hard drive under investigation looking for the value 0 for the key "HKLM/SYSTEM/CurrentControlSet/Control/FileSystem/NtfsDisableLastAccessUpdate".

Conclusion

This research is a foundation for building a dating framework for file fragments. Dating file fragments is an important step for event reconstruction when deleted files are found to be part of the evidence corpus. In this work, we showed how dates of neighboring files can be used to infer a minimum boundary for when a deleted file — found by its fragment(s) — was created. Similarly, we showed how the maximum date from the currently allocated file — Slack-Owner — can be used to define the upper-bound period for when the file was deleted. Together these two boundaries create a time window for a deleted file for which a fragment was found in a file slack. Our dating method accuracy is affected by heavy usage of the hard drive, the frequency of hard drive defragmentation, and the file system type being used.

Our future work will include applying the relative dating scheme on FAT32 systems and comparing the results to the NTFS findings. Other work we are currently investigating is the applicability of our bounding scheme on fragments found on the unallocated space or the volume slack of a partition. Furthermore, we intend to expand our analysis to a more realistic data set where the hard drive is used for a longer period of time than the activity period found in the M57 data set.

References

Allen, J.F., 1983. Maintaining knowledge about temporal intervals. Commun. ACM 26, 832–843.
Allen, J.F., 1991. Time and time again: the many ways to represent time. Int. J. Intell. Syst. 6, 341–355.
Beebe, N.L., Maddox, L.A., Liu, L., Sun, M., 2013. Sceadan: using concatenated n-gram vectors for improved file and data type classification. IEEE Trans. Inf. Forensics Secur. 8, 1519–1530.
Boyd, C., Forster, P., 2004. Time and date issues in forensic computinga case study. Digit. Invest. 1, 18–23.
Buchholz, F., Spafford, E., 2004. On the role of file system metadata in digital forensics. Digit. Invest. 1, 298–309.
Buchholz, F., Tjaden, B., 2007. A brief study of time. Digit. Invest. 4, 31–42.

Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley Professional. Part 3 File System Analysis.
Casey, E., 2011. Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet. Elsevier Science. CH-17.
Casey, E., 2018. Digital stratigraphy: contextual analysis of file system traces in forensic science. J. Forensic Sci. 63, 1383–1391.
Chabot, Y., Bertaux, A., Nicolle, C., Kechadi, T., 2014. Automatic timeline construction and analysis for computer forensics purposes. In: IEEE Joint Intelligence and Security Informatics Conference, pp. 276–279.
Cho, G.-S., 2013. A computer forensic method for detecting timestamp forgery in ntfs. Comput. Secur. 34, 36–46.
Chow, K.-P., Law, F.Y., Kwan, M.Y., Lai, P.K., 2007. The rules of time on ntfs file system. In: Systematic Approaches to Digital Forensic Engineering, pp. 71–85. SADFE 2007. Second International Workshop on, IEEE.
Garfinkel, S.L., 2007. Carving contiguous and fragmented files with fast object validation. Digit. Invest. 4 (Suppl. ment), 2–12.
Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora, Digital Investigation 6. In: The Proceedings of the Ninth Annual DFRWS Conference, pp. S2–S11.
Gladyshev, P., Patel, A., 2004. Finite state machine approach to digital event reconstruction. Digit. Invest. 1, 130–149.
Hargreaves, C., Patterson, J., 2012. An automated timeline reconstruction approach for digital forensic investigations, Digital Investigation 9. In: The Proceedings of the Twelfth Annual DFRWS Conference, pp. S69–S79.
Inglot, B., Liu, L., 2014. Enhanced timeline analysis for digital forensic investigations. Inf. Secur. J. A Glob. Perspect. 23, 32–44.
Inglot, B., Liu, L., Antonopoulos, N., 2012. A framework for enhanced timeline analysis in digital forensics. In: IEEE International Conference on Green Computing and Communications, pp. 253–256.
Jeyaraman, S., Atallah, M.J., 2006. An empirical study of automatic event reconstruction systems. Digit. Invest. 3, 108–115.
Jones, J.H., Khan, T.M., 2017. A method and implementation for the empirical study of deleted file persistence in digital devices and media. In: IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), pp. 1–7.
Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 558–565.
R. Poisel, M. Rybnicek, S. Tjoa, Taxonomy of Data Fragment Classification Techniques, Springer International Publishing, Cham, pp. 67–85.
Roussev, V., Garfinkel, S.L., 2009. File fragment classification-the case for specialized approaches. In: Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, pp. 3–14.
Schatz, B., Mohay, G., Clark, A., 2006. A correlation method for establishing provenance of timestamps in digital evidence. Digit. Invest. 3, 98–107.
Silberschatz, A., Galvin, P., Gagne, G., 2012. Operating System Concepts, ninth ed. Wiley Global Education. CH-11, File System Implementation.
C. J. Veenman, Statistical disk cluster classification for file carving, in: Third International Symposium on Information Assurance and Security, pp. 393–398.