Generic Metadata Time Carving

By: Rune Nordvik (Norwegian University of Science and Technology (NTNU); Norwegian Police University College), **Kyle Porter** (NTNU), Fergus Toolan (Norwegian Police University College), Stefan Axelsson (NTNU and Halmstad University), and Katrin Franke (NTNU)

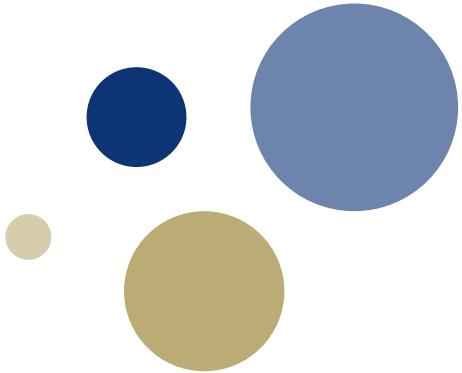
Winner of the Best Paper Award for USA 2020

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2020 USA

Virtual -- July 20-24



Generic Metadata Time Carving

Rune Nordvik, Kyle Porter, Fergus Toolan, Stefan Axelsson, Katrin Franke

Breaking Down the Title

- **Carving** for filesystem **metadata** entries
- Metadata is carved by first looking for evidence of **timestamps**
  - We use timestamps as a *dynamic signature* for metadata entries
- The approach should be as **generic** as possible

Why carve for FS metadata?

Situations where a file system may be damaged beyond use.

- Recover files and metadata without using filesystem critical datastructures
  - No $MFT file, no Super Blocks, no Group Descriptor tables.
- Possibility of full file recovery (FS metadata and file)
  - Traditional file carving misses the filesystem metadata and has difficulty with fragmentation.

Why Timestamps as a Dynamic Signature

- Filesystems typically record multiple timestamps per metadata entry, typically near each other.

File System	Co-located timestamps	Granularity
NTFS [3]	4	64 bit - ns intervals since 1.1.1601
ReFS [16]	4	64 bit - ns intervals since 1.1.1601
APFS [13]	4 (5)	64 bits - ns since 1970
HFS+ [1]	4	32 bits - s since 1904
BTRFS [2]	3 (4)	64 bits - s since 1970 + 32 bits (ns)
ExFAT [12]	3	32 bits + UTC offset
FAT [3]	3	16 bits for time (except accessed), 16 bits for day
UFS1 [3]	3	32 bits - s since 1970 + 32 bits (ns)
UFS2 [3]	4	64 bits (ns) since 1970
Ext2/3 [3]	4	32 bits - s since 1970
Ext4 [7]	4	34 bits - s since 1970 + 30 bits (ns) [11]

Table 1: File Systems with timestamps co-located within metadata structures

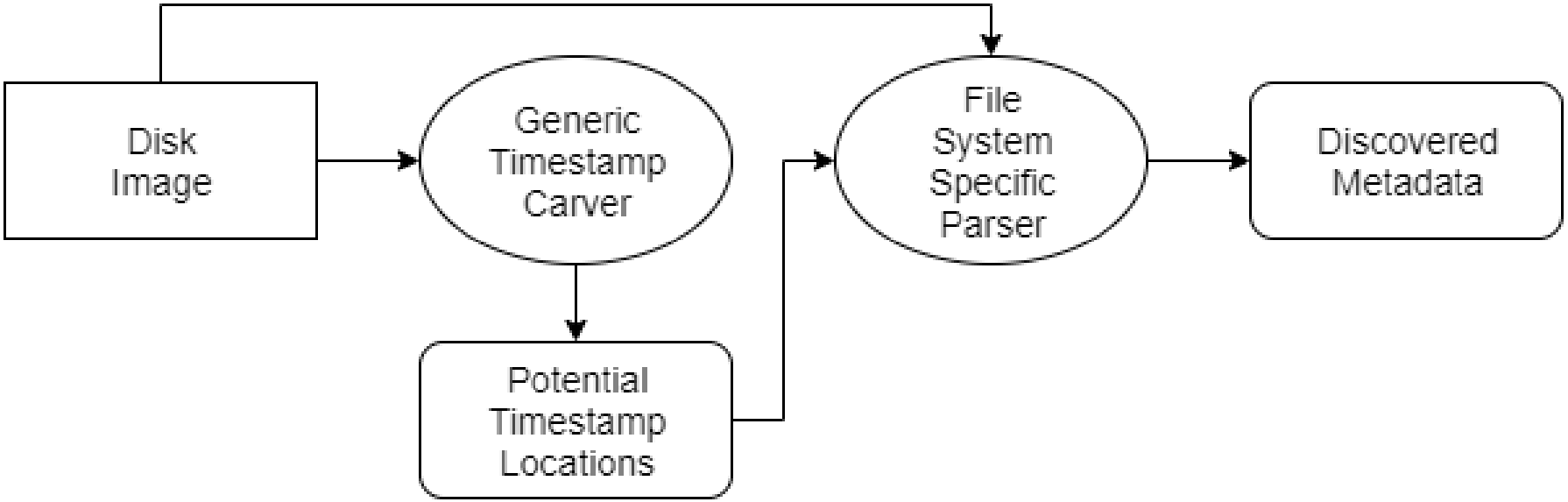
Why Timestamps as a Dynamic Signature

- Multiple timestamps within a metadata entry often appear to be equal.
- If these features are present in most filesystems, then we have found a generic feature of FS metadata.

Research Objectives

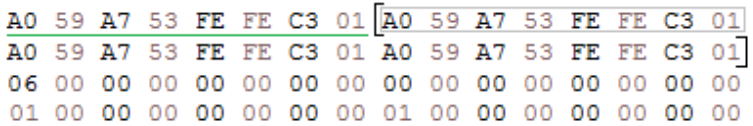
- Can we reliably use time as a generic identifier to carve for file and metadata structures in different file systems?
- What is the reliability of recovery of files using the discovered metadata in Ext4 and NTFS?

Generic Metadata Time Carving Flowchart

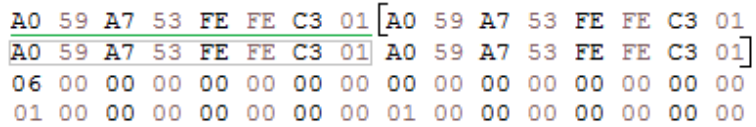


Generic TS Carving Algorithm

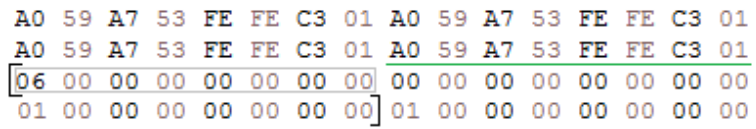
An example:



Match count = 1.



Match count = 2. Potential timestamp found.



Search repeats after shift.

Simple string matching algorithm that runs in $O(\lfloor\frac{|T|}{m}\rfloor * \frac{k}{m})$

- T is the image, m is the length of the timestamp, and k is the length of search threshold.

Filesystem Specific Parsers

- Potential timestamp carving alone will produce many false positives.
- We need a hand tailored parser for the assumed file system, which act as an validator for the found locations of potential timestamps
  - Checks offsets from timestamps for expected features.
  - Checks features within the metadata is consistent with itself.

Experiment

- Question: can our method extract metadata entries from damaged file systems?
- To test, we damaged synthetic disk images we created running NTFS and Ext4 by:
  - Reformatting NTFS with exFAT
  - Reformatting Ext4 with NTFS (with the inode number as an attribute inside the inode itself).

Experiment

- NTFS image:
  - 2 GB
  - 10 bmp, 10 jpg, 10 png, 10 tiff, 10 txt
- Ext4 image:
  - 2 GB
  - 25000 txt files, 50 directories

Recorded the precision of our method's ability to identify inodes/MFT records.

Record the recall of the known metadata entries that we extracted.

Assumptions for Experiments

- NTFS:
  - Timestamp Length
- Ext4:
  - Timestamp Length
  - Blocksize
  - Inodes are 256 bytes in size
  - Offset to beginning of Ext4 partition
- Additional assumptions for Ext4 necessary to make guesses of the inode number and filename pair for each recovered inode.

Results: NTFS reformatted with exFAT

	TP	FP	FN	Precision	Recall
SIA matches	162	1	0	0.9939	1

Table 2: Precision and Recall for finding MFT records in ntfsexfat.dd

Results: Ext4 reformatted with NTFS

TP	FP	Precision
57427	0	1

Table 6: Precision of inode classification for reformatted image.

	TP	FP	Precision	Files Found
Recorded inode matches True inode	15544	41692	0.2716	4848
Estimated inode matches True inode	7091	50145	0.1239	5755
Est or Rec inode Matches True inode	16553	40683	0.2892	5755

Table 5: Precision and Files Found for finding and attributing iNode numbers for known files in Ext4AttrNowNTFS.dd

Comparing Commercial Tools

- Could these tools retrieve any metadata entries from our reformatted disk images?

	EnCase	X-Ways	EaseUS	Bulk_Extr	cPTS
NTFS metadata	N	N	Y?	Y	Y
Ext4 inode	N	N	N	N	Y

Summary of Analysis

- NTFS overwritten with ExFat:
  - Large benefit is we can find resident files in MFT entries, which most tools missed (Bulk_Extractor could read it).
  - Most tools had trouble with the fragmented images
- Ext4 overwritten with NTFS:
  - Overwriting Ext4 wiped approximately 20257 inodes from the inode table (Ext4 partition used flex groups)
  - We recovered 5755 inodes, which means that we recovered some 963 overwritten inodes.
  - The other tools could not recover any of the text files.

Runtime

- NTFS (2 GB):
  - Potential timestamp carving: ~13 seconds
  - NTFS Parser: < 1 second
- Ext4 (2 GB):
  - Potential timestamp carving: ~20 seconds
  - Ext4 Parser: ~8 seconds

Conclusion

- Can use timestamps as a reliable identifier of filesystem metadata entries, but need to filter results with parsers.
- Can reliably use our method to fully recover files, if the metadata entry contains multiple equivalent timestamps.
- Can work on images with damaged filesystems.

Questions?

- Rune.Nordvik@phs.no
- kyle.porter@ntnu.no

- Link to github (code and data):
  - https://github.com/reviewscientific2020/cPTS

Extra Slides

GMTC Method

Assuming a timestamp of length m, and let the disk image be represented by a byte array of length T:

- For *most* non-overlapping byte sequences of length m, we test if it is equal to a number of m length byte sequences within a k byte threshold.
- If the test is passed, we record the byte position in an output file.
- The output file is used by our semantic parsers in order to validate if this is a specific type of file system metadata
- The output of the semantic parser can then be used for file recovery

Generic Timestamp Carving Details

- Timestamp search is filesystem agnostic:
  - Only looks at re-occurring byte sequences
- Don’t need to assume a datetime range
- Must assume length of timestamp

Metadata Hits and False Positives

Example for a positive hit

SIA Location	Filename	File Type	Index Attribute Datarun	Extra Filenames	SIA created	SIA Created (Decimal)
664922184	File9.txt	File		[]	9/26/2019 12:04	1.3214E+17
664836472	$Quota	File		[]	9/26/2019 12:04	1.3214E+17

Elements of a false positive

```
Timestamp: Mon Jan  1 00:00:00 1601 Microseconds 405 (UTC), FNA hit
Byte Location (dec): 665646104
Start of File Name Attribute (FNA)
Filename: b'\x00\x00\x05\xd3Q'
Created: Mon Jan  1 00:00:00 1601 Microseconds 405 (UTC)
Modified: Mon Jan  1 00:00:00 1601 Microseconds 220001 (UTC)
MFT Modified: Wed Jan  3 07:57:13 1601 Microseconds 967296 (UTC)
Accessed: Fri Jun 17 21:44:48 1605 Microseconds 355329 (UTC)
Allocated Size of File (dec): 0
Logical Size of File (dec): 0
Parent_ID: 0100000001000100
End of File Name Attribute (FNA)
```

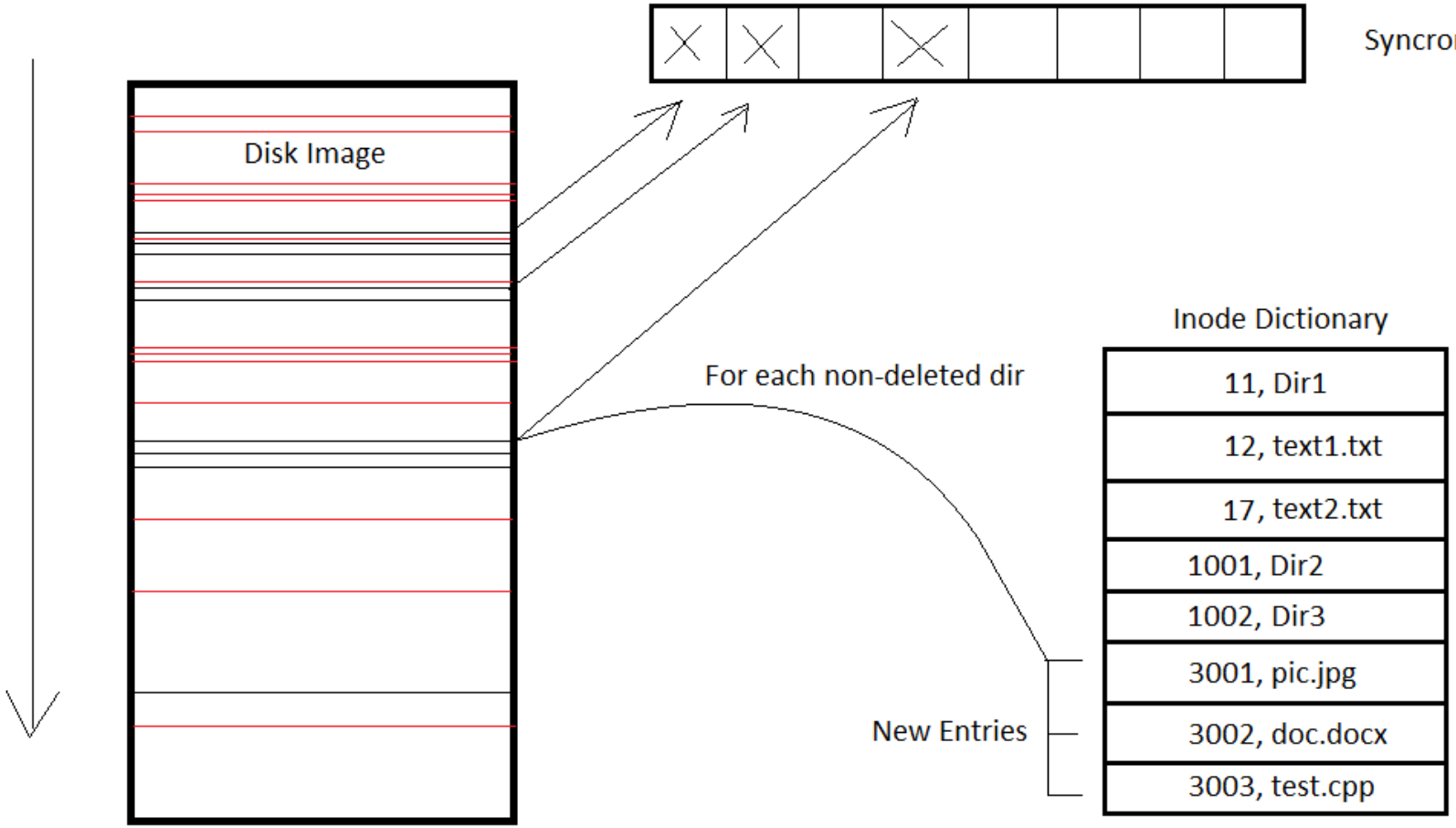
Matching filenames and inode numbers to inodes

First pass over the image (set up):

Using the list of byte offsets of the potential timestamps, run some basic validity tests to determine if this is really an inode. Add valid inodes to a **list**.

1. For inodes that are added to the list, are a directory, and have not been deleted, go to the directory via its direct block/extent and add all filename and inode number combinations to a **dictionary**.
2. If a inode of a valid directory is the first of its kind in a blockgroup, we add its inode number and location to a **synchronization list** (one entry per blockgroup).

First Pass set up



Matching filenames and inode numbers to inodes

Second pass over the image:

Using the list of valid timestamps, guess their inode numbers and filenames (2 methods).

1. For valid inodes that are in the same block group as an entry in the synchronization list we calculate its inode based on its physical position:

Est inum = directory inum + (valid inode loc – dir inode loc)/256

We can try to look up its name in the dictionary based on the estimated inode number.

Entry per block group of the synchronization list updated when coming across a new directory that hasn't been deleted. Allows for local synchronization.

Second Pass Estimated Inode Num

Syncronization List

inum: 11, position 4096

Disk Image

position: 5632 (dec)

estimated inum = 11 + (5632 - 4096)/256 = 17

estimated name = text2.txt

Inode Dictionary

11, Dir1
12, text1.txt
17, text2.txt
1001, Dir2
1002, Dir3
3001, pic.jpg
3002, doc.docx
3003, test.cpp

Matching filenames and inode numbers to inodes

Second pass over the image:

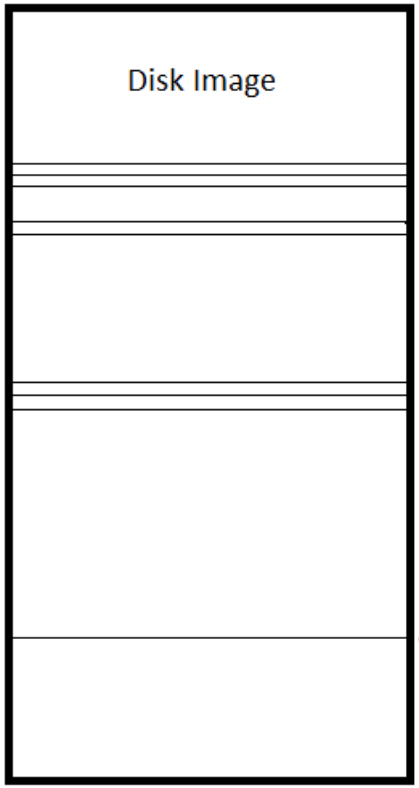
Method 2:

When the first inum estimate is made, we add relatively unique features from the inode to its entry in the inode dictionary (file version and created time).

When encountering future inodes, we can try to look up its inode number and name in the dictionary, but keyed with the file version and created time.

Second Pass Recorded Inode Num

Syncronization List



position: 29184 (dec)

estimated inum = ?

file version = 712

Inode Dictionary	file version/creation time
11, Dir1	123/Jan-01-2014
12, text1.txt	124/Nov-24-2016
17, text2.txt	125/Dec-25-2014
1001, Dir2	222/Jan-01-2014
1002, Dir3	223/Jan-01-2014
3001, pic.jpg	711/Sep-05-2016
3002, doc.docx	712/Sep-05-2016
3003, test.cpp	720/Oct-31-2016

Recorded inum = 712
Recorded name = doc.docx

Why not just….

- Search for FILE signature in NTFS?
  - Will not identify records for BAAD
  - Will not identify records where part of the header is overwritten
- Search for the extent header magic number to find Ext4 inodes?
  - Not all inodes in Ext4 will have extents, thus misses inodes using block pointers.
- Reminder: We are trying to find a general signature that can be applied to most file systems.

Limitations

- Timestamps in filesystem metadata entries were guaranteed to have at least 2 or 3 matching timestamps.
  - It does not indicate our tool can recover all metadata entries, but it will work well *given* there are equivalent timestamps in the metadata entries.
- We do not differentiate between filesystem metadata entries found in the MFT/inode table and those found in journals etc.
- We have not even considered the case involving virtual machines.

Further work

- Support for additional file system validators, more than NTFS and Ext4
- Improve the accuracy connecting inodes to their filename-inode number pairs.
- Support for approximate timestamp carving