Research Project 1
APFS checkpoint management behaviour in macOS

Maarten van der Slik
maarten.vanderslik@os3.nl

Security and Network Engineering
University of Amsterdam
Amsterdam, The Netherlands

Supervisor: Ruud Schramp
Nederlands Forensisch Instituut (NFI)
The Hague, The Netherlands
20-01-2020

***Abstract*—The Apple File System (APFS) is the default file system for Apple devices since macOS High Sierra and iOS 13 [1]. APFS has the ability to create and restore snapshots, in addition to other new features. Snapshots create and restore checkpoints, which contain the actual data and metadata [2]. While other researches have focused on decoding the file system [3] [4], no research has been done on when macOS creates and removes checkpoints. This paper shows the correlation between different file operations and the amount of data traces macOS leaves behind. Therefore, 12 experiments with different file operations are performed and analyzed.**

From the results, we conclude that macOS does not always use the Copy-on-write principle, which, in some cases, leaves only one file version behind. In other cases, macOS does use this principle, leaving many or all the file versions behind. Restarting macOS can cause some file versions and checkpoints to be overwritten.

While there might be some gaps between series of checkpoints, macOS creates them a lot and they contains a vast amount of metadata, which can be retrieved by parsing a tree with inode information like MAC times and owner.

I. Introduction

Since 2017, Apple uses his new file system technology, Apple File System (APFS), by default on his macOS and iOS devices. APFS is the successor of HFS+, that served the Apple platforms for almost two decades. When the new system was introduced during WWDC on June 14, 2016, Apple stated that it is crash protected, supports "Space Sharing" between volumes, cloning, and snapshots. It should also feature enhanced encryption options, compared to its predecessor. [5]

The new snapshots feature holds the state of an APFS system at a past point in time [1]. Snapshots refer to a certain checkpoint, which are copies of important file system structure metadata [3]. Because these checkpoints refer to actual files, they could contain forensically interesting data.

Nowadays, there has been done a couple of researches on the low-level technology, on both the general structure of the system, as well as the forensic possibilities the platform offers. For example, Hansen and Toolan describe the tables and artifacts of APFS [3] in their paper, Dewald and Plum found an effective way to recover removed file content from it [2] and Song et al. analyzed timestamps of directories and files [6].

However, to the best of our knowledge, the behavior of handling APFS checkpoints in macOS has not been analyzed yet and the frequency of creating and removing them is still unknown. In this research, we try to specify when checkpoints are created and removed and which data could be extracted from it.

II. Research Question

The research question we defined is: **When does macOS Catalina create APFS checkpoints and which data could be retrieved from them?**

This paper researches the correlation between certain file system operations and the amount of data traces in an APFS partition that are left behind by macOS. We look at the results of a set of experiments with certain disk operations in a testing environment. We create, change, clone, move and delete volume items. We also change file content using different file access modes (low-level method) and using the Foundation API (higher-level method). Lastly, we use the TextEdit application to show, which is the default graphical text editor of macOS [7].

The experiments show which versions of a file or a volume remain after changing it for a certain amount of time and how much meta-data is available. Lastly, we explain what could cause these differences in results, based on the differences between the experiments and the data we retrieve during them.

To be able to interpret the results, this paper also gives the necessary background information about APFS checkpoints, based on former research. We look into the properties of an APFS checkpoint (what are the technical differences and interdependencies between them and a regular checkpoint).

III. Related Work

In the first year of the introduction of APFS, no technical information was available about its structure [3], which was the motivation for Hansen and Toolan to decode APFS. They described the major components and structure of APFS, using their own naming schema [4]. A very important component of APFS is the Container Superblock (CSB), which is the table that defines the overall structure of an APFS container. Therefore, it describes the total number of blocks, block size limitations and the locations of earlier versions of the CSB (checkpoints). The Master Superblock (MSB) is a subset of the latest CSB. [3] While containers describe a whole APFS system, they are subdivided by volumes. The locations of the volumes are specified in the CSB. Many important information about a volume is saved in

the Volume Superblock (or as Hansen and Toolan name it: the Volume Checkpoint Superblock) [1]. One example of important information in the Volume Superblock is the block number to the catalog B-tree. The OS uses this B-tree to find out where a file is located and if it has Extents (file content) [3]. Snapshots point to a previous CSB (sblock_oid) and a backup of the extent-reference tree using a block address (extentref_tree_oid) [1]. Hansen and Toolan also described the Bitmap Structures (BMS). This is a collection of tables, that records used and unused blocks. Using this mechanism, blocks can be allocated to a specific volume [3]. The Bitmap tables should be maintained, because when data is written to a block without knowledge of this system, it may be overwritten [8].

After the research paper of Hansen and Toolan, a couple of new papers where published. Plum and Dewald described in their paper how files could be carved from APFS. The tool they created searches for references in the container- and volume superblock to find pointers to file extends. Because APFS is a copy-on-write system, there is a great chance that removed files still exist. Plums and Dewald showed that using their file system parsing method, a significantly greater percentage of content could be recovered [2].

To the best of our knowledge, the macOS behavior of creating and removing checkpoints has not been analyzed jet, and as Plum and Dewald pointed out: it still has to be researched [2]. CSB's could also be generated without a snapshot reference (e.g. the first checkpoint of the initial state of the container), which should also contain metadata of the volumes [3], which makes it forensically interesting. As far as we know, the frequency of generating checkpoints is still unknown.

IV. Method

To find out how many versions of a file or a volume are available after performing certain disk operations, we use a test setup, consisting of a computer (Core i7-6700HQ, 16GB RAM, 256GB SSD) running VMware software, with a macOS Catalina (10.15.2) VM. We use APFS version 1412.61.1. We disable automatic snapshot creation in macOS, to prevent the system to create read-only checkpoints during the experiments, which would influence the checkpoint results.

We perform 12 experiments, 4 of them change the content of a file and the remaining 8 the structure of a volume. All experiments perform a disk operation 65 times, the file content experiments wait for 60 seconds between each operation, the volume structure experiments 30 seconds. All experiments are automated using a script, except for the TextEdit file content experiment. We perform the experiments at least two times. Before each experiment, we create a raw disk of 256MB which does not contain any data. We used diskutil to format the disk with a GPT partition table and an unencrypted, case-insensitive APFS partition. The block size is 4096, which is the default configuration [4].

A. File content experiments

For each file content experiment, a UTF-8 file of 400.0kB is created. The file contains 9999 lines, every line contains the ctime (time changed) and a line number. We perform four different experiments which execute one of the following file operations:

- Seek & write file editing (system calls: open, seek through the file, rewrite only 40 bytes, close)
- Normal file editing (system calls: open, rewrite all the blocks of the file, close)
- Appending to file (system calls: open, seek to the end of the file, write 41 bytes, close)
- Edit using a file handle with Foundation API (system calls: open, seek through the file, rewrite only 40 bytes, close)
- GUI file editing by TextEdit application (system calls: open file and temporary, write all blocks to temporary, close both, change inode entry of original file, delete the inode of the temporary file)

While the scripts we use perform simple changes to a text file, we aim to cover the common file access write modes that are used by macOS applications.

According to Apple's documentation, most applications that need to read or write a disk will use the Foundation API [1]. We use an Objective C script to see if macOS checkpoint behavior differs when using this high-level API's. We also use TextEdit, which is a regular GUI application that works fundamentally different than the other file content operations.

Between each file operation, the file is closed and opened again and we check the system calls using the dtruss and fs_usage log tools. After the experiment, we analyze the low-level data on the disk, by carving for the magic UTF-8 bytes "APSB" (a method which is described by Plum & Dewald [2]) and parsing the Volume Superblocks using The Sleuth Kit, to find out how many versions are available from the checkpoint mechanism. Also, we verify this method by also manually search the low-level data of the seek & write and Foundation-API disks for file content blocks.

After this first collection of experiments, we execute the file content operations again but restart the system two times during the experiment. The first time after 23 disk operations, the second time after 44 operations. We analyze the low-level data and see if there is a difference in the number of checkpoints.

To verify that macOS does handle APFS Checkpoints in the same way on smaller disks as it does with bigger disks, we also perform the seek & write and Foundation API experiments on a 48GB raw disk image.

B. Volume structure experiments

We also research the checkpoint behavior of macOS when changing the file system structure. We perform the following file system operations:

- Create a file (touch)
- Create a folder
- Move a file
- Move a folder
- Clone a file (using the clonefile() function)
- Clone a folder (using the clonefile() function)
- Remove a file
- Remove a folder

We do not create file content during the file creation experiment because the recovery possibilities of removed file content are already researched by Plum & Dewald [2].

After we performed these operations, we execute the experiments again, but restart the system two times during the experiment, at the same times we did during the file content experiments.

We interpret the results using the documentation of Apple [1] and the whitepaper of Dewald and Plum [4].

V. Ethical issues

This research does not have any negative impact on an individual's security or privacy nor does it expose security weaknesses of APFS that could be exploited by attackers. Experiments took place in an environment specifically set up for the tests themselves (see paragraph IV) and the environment did not contain any sensitive information.

VI. Results

We divide this section into two parts: we look at the behavior of macOS when executing various file operations and in the other part, we look at the behavior of macOS when changing the file system structure (i.e. creating and removing items like folders and files).

A. File operations

In Table I, we see the results of the file operation experiments. In this paragraph, we look into the number of file versions that are available after performing a disk operation 65 times, without rebooting the system during the experiment.

In the seek & write file experiment, we observe the script opening a file in read-write mode (open: RW_____X), seeking to a specific location and changing 41 bytes. After every operation, we see the close syscall to close the file. When performing a low-level data search, we find only file content blocks of the latest version of this file.

The second file operation opens the file in write mode (open: _WC_T_____X) and writes all blocks again. Afterward, we see the close syscall to close the file. After parsing all the carved Volume Superblocks, we see that changes are always written to a new data block, which results in the availability of all versions of the file.

The third operation opens the file in append mode (open: _WCA_____X), seeks to the end of the file and writes only 41 new bytes to it. Afterward, we see the close syscall to close the file. While there are many newer versions available of this file, not every file version is available after parsing all the carved Volume Superblocks.

The last file operation in this category uses the Foundation framework, which opens the file in read-write mode and seeks to a specific location, just like the seek write experiment. Afterward, 41 bytes are changed and the file is closed. This operation writes to the same set of blocks every time and all the checkpoints which contain this file point to this original block.

Performing the seek & write experiment using a 48GB disk gives 141 checkpoints and 1 file version, performing the Foundation API experiment on it gives 196 checkpoints and 1 file version. Both experiments are performed without restarts between the operations.

After these experiments, we execute them again. This time, we restart macOS two times during the experiment. We see a significant difference in available file versions: the file versions that are created before the first and second restart are overwritten (FIGURE), only one corrupted file version could be restored. However, all the file versions after the last (second) restart are still available.

B. File operations

During the TextEdit experiments, we observe results that are similar to the second operation (rewrite all file content blocks). dtruss output shows that this application opens a new temporary file (.sb) to save the new file version and removes it after changing the pointer of the original file to the changed content. Without reboots during the experiment, this method creates many checkpoints (223) and contains all file versions. With reboots during the experiment, this method creates 108 checkpoints and contains 32 file versions. But, the write procedure (which is described in section IV) is very different from the file operations in Table I, which do not create temporary files.

C. Volume structure operations

In Table II we find the number of available volume versions and created checkpoints when performing the folder experiments, in Table III we find the results when performing file operations. We see no significant change in the average amount of checkpoints and volume versions when comparing the experiments with each other and comparing the experiments with restarts against the experiments without restarts.

D. Available metadata

```
01 20 2020 21:27:21   409959 m..b f 0   0   0-149-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-150-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-151-133 /root/Test1A/Test1AA/Low-level/1
01 20 2020 21:28:21   409959 m..b f 0   0   0-152-133 /root/Test1A/Test1AA/Low-level/1
01 20 2020 21:29:21   409959 m..b f 0   0   0-153-133 /root/Test1A/Test1AA/Low-level/1
01 20 2020 21:30:21   409959 m..b f 0   0   0-154-133 /root/Test1A/Test1AA/Low-level/1
01 20 2020 21:31:22   409959 m..b f 0   0   0-155-133 /root/Test1A/Test1AA/Low-level/1
01 20 2020 21:32:21   409959 m..b f 0   0   0-156-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-157-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-158-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-159-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-160-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-161-133 /root/Test1A/Test1AA/Low-level/1
                      409959 m..b f 0   0   0-162-133 /root/Test1A/Test1AA/Low-level/1
```

Fig. 1. A part of the seek & write timeline, shown in mactime

Every Volume Superblock contains a reference to the volume Object Map (OMAP), where a pointer to the root tree is located [1]. The root tree contains inode entries, from which the MAC times, owner, and parent inode of files and folders could be extracted [4]. By extracting all inode entries from the existing Volume Superblocks, we are able to create a timeline of volume modifications.

The tool Plum & Dewald created for removed file content recovery is also able to parse the metadata and creates a body file for The Sleuth Kit [9].

Figure 1 shows a part of a timeline, which is created using the body file of a seek & write experiment and the mactime tool from The Sleuth Kit. Even though this experiment has only one file version, the timeline shows significantly more information about when the file is created and changed.

VII. Discussion

While Plum & Dewald already created a tool that is able to parse older versions of an APFS Volume with their metadata, they did not look into how macOS threats file access modes different from each other; when it is possible to recover older versions of a file successfully. If macOS had always used the Copy-on-write mechanism, the chance of file version recovery would have been significantly greater, as the results of the second file experiment (rewrite of all blocks) shows us.

To verify the results of the seek & write experiment and the Foundation API experiment (that show that no more than one version of the file is available from an APFS partition image) we used low-level data searches to find all blocks which contain

	Operation	Checkpoints w/o restart			File versions available w/o restart			Checkpoints w/ restart		File versions available w/ restart	
		Result 1	Result 2	Result 3	Result 1	Result 2	Result 3	Result 1	Result 2	Result 1	Result 2
1	Seek & write	65	127		1	1		67	163	1	1
2	File rewritten	84	285	91	65	65	65	108	67	24 (1 corrupted)	23
3	File appended	80	30		21	18		91	116	22	31
4	Foundation API	218	278		1	1		111	175	1	1

TABLE I

AMOUNT OF CHECKPOINTS AND AVAILABLE FILE VERSIONS AFTER PERFORMING CERTAIN FILE OPERATIONS 65 TIMES. **W/O** MEANS "WITHOUT", **W/** MEANS "WITH". ONLY CHECKPOINTS THAT INCLUDE THE FILE ARE INCLUDED IN THE NUMBER OF CHECKPOINTS.

	Operation	Checkpoints w/o restart		Versions available w/o restart		Checkpoints w/ restart		Versions available w/ restart	
		Result 1	Result 2	Result 1	Result 2	Result 1	Result 2	Result 1	Result 2
1	Folders created	35	38	19	21	85	54	37	22
2	Folders cloned	49	49	29	33	48	70	31	34
3	Folders moved	38	55	20	17	32	63	8	30
4	Folders removed	44	24	27	19	32	56	13	9

TABLE II

AMOUNT OF CHECKPOINTS AND AVAILABLE VOLUME VERSIONS AFTER PERFORMING CERTAIN FOLDER OPERATIONS 65 TIMES. **W/O** MEANS "WITHOUT", **W/** MEANS "WITH". ONLY CHECKPOINTS THAT INCLUDE ONE OR MORE OF THE CREATED FOLDERS ARE INCLUDED IN THE NUMBER OF CHECKPOINTS.

	Operation	Checkpoints w/o restart		Versions available w/o restart		Checkpoints w/ restart		Versions available w/ restart	
		Result 1	Result 2	Result 1	Result 2	Result 1	Result 2	Result 1	Result 2
1	Files created	39	37	19	19	20 (1 overwritten root tree)	60	10	28
2	Files cloned	37	39	17	19	38	16	11	10
3	Files moved	38	56	19	20	86	31	35	12
4	Files removed	42	57	25	16	62	57	15	15

TABLE III

AMOUNT OF CHECKPOINTS AND AVAILABLE VOLUME VERSIONS AFTER PERFORMING CERTAIN FILE OPERATIONS 65 TIMES. **W/O** MEANS "WITHOUT", **W/** MEANS "WITH". ONLY CHECKPOINTS THAT INCLUDE ONE OR MORE OF THE CREATED FILES ARE INCLUDED IN THE NUMBER OF CHECKPOINTS.

parts of the file. We did not use this time-consuming method on every experiment (only on the seek write and Foundation API experiments), but used the Volume Superblock carving method to find all available file versions. In theory, this could mean that there is still an older version of a file block present and the associated Volume Superblock is overwritten by other file content. But, our method should be sufficient due to the vast amount of available checkpoints.

In this research, we perform the experiment on a specific disk, which is not the system partition where macOS is installed. If macOS handles system partitions in a different way, the results do not apply to them. Also, we did not research if the amount of checkpoints macOS creates depends on the APFS partition size. It could be that the checkpoint numbers are higher when using bigger APFS partitions than the 256MB partitions we used in the experiments.

Due to the long list of experiments and the time they take to complete, there are not many samples per experiment. But, because the key findings are based on both the with reboot and without results, we consider the number of samples to be enough for our conclusions. This does not amount to the amount of file versions in the write mode situation, where the "with reboot" results differ from the "without reboot" results. For this reason, we performed that experiment one more time.

VIII. CONCLUSION

We revisit our research question to give a final verdict on our research: **When does macOS Catalina create APFS checkpoints and which data could be retrieved from them?**

macOS leaves many data traces behind when using an APFS partition. Many checkpoints are created during our experiments, and a vast amount of metadata could be retrieved from parsing the root tree with the inode entries that checkpoints contain. The inode entries contain information such as MAC times and owner. The structure of an earlier version of a volume could be reconstructed or a timeline could be created by parsing all the root trees. Rebooting does not have a significant impact on the number of checkpoints, but loss of several file versions can occur.

While some file operations leave many file version traces behind, other file operations leave only one version behind. Our results show that macOS writes changes to new blocks when using the write-only access mode, while the other file access modes do not (always) write their changes to new blocks. While some papers might claim in their introduction sections that macOS uses a Copy-on-write mechanism, our results show that macOS does not always use this mechanism, which makes it not comparable to the Copy-on-write implementation in Btrfs or the "always_cow" configuration in XFS, which never overwrite existing blocks [10] [11].

IX. FUTURE WORK

During our research, we researched the correlation between different file system operations and the data traces macOS leaves behind. For future work, we would:

- Perform the experiments a couple of times more.
- Reverse engineering the macOS APFS driver or performing more experiments to be able to describe in more detail when macOS creates checkpoints.
- Invent a method to find root trees and their nodes without carving for the Volume Superblock magic bytes, because the Volume Superblock can be overwritten.
- Research if there is a correlation between the number of checkpoints or file versions and the size of a disk.
- Research if macOS threats a system partition differently from other partitions.
- Find out what the impact is on checkpoints when improperly unmounting a disk device since Apple mentioned that

some data is saved in-memory which should be saved to a checkpoint [1].

REFERENCES

[1] Apple Inc. Apple file system reference, Feb 2019.

[2] Jonas Plum and Andreas Dewald. Forensic apfs file recovery. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, page 47. ACM, 2018.

[3] Kurt H Hansen and Fergus Toolan. Decoding the apfs file system. *Digital Investigation*, 22:107–132, 2017.

[4] Andreas Dewald and Jonas Plum. Apfs internals for forensic analysis, Apr 2018.

[5] Apple Inc. Introducing apple file system - wwdc 2016 - videos, Jul 2016.

[6] Jong-Hwa Song, Se Ho Kim, Song Yi Hwang, Seung Gyu Kim, and Sung-Jin Lee. A study on the apfs timestamps in macos. *International Journal of Engineering & Technology*, 7(2.33):133–138, 2018.

[7] Apple Inc. Open documents in textedit on mac.

[8] Marcel den Reijer and Henk van Doorn. Artifacts in the apple file system. OS3 student, 2018.

[9] Jonas Plum. afro (apfs file recovery), Jan 2020.

[10] Jonathan Corbet. The btrfs filesystem: An introduction, Dec 2013.

[11] Michael Larabel. Xfs copy-on-write support being improved, always cow option, Feb 2019.