Digital Investigation 15 (2015) 119–123

Contents lists available at ScienceDirect

Digital Investigation

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

Letter to the Editor

Comments on the Linux FAT32 allocator and file creation order reconstruction [Digit Investig 11(4), 224–233]

Keywords
Linux file system
FAT32
Recovered file
Creation time

A B S T R A C T

Minnaard proposed a novel method that constructs a creation time bound of files recovered without time information. The method exploits a relationship between the creation order of files and their locations on a storage device managed with the Linux FAT32 file system. This creation order reconstruction method is valid only in non-wraparound situations, where the file creation time in a former position is earlier than that in a latter position. In this article, we show that if the Linux FAT32 file allocator traverses the storage space more than once, the creation time of a recovered file is possibly earlier than that of a former file and possibly later than that of a latter file on the Linux FAT32 file system. Also it is analytically verified that there are at most n candidates for the creation time bound of each recovered file where n is the number of traversals by the file allocator. Our analysis is evaluated by examining file allocation patterns of two commercial in-car dashboard cameras.

Introduction

Minnaard analyzed the file allocation pattern of Linux FAT32 file system and revealed that relative positions of files on a storage device is directly related with their creation order (Minnaard, 2014). The relationship between file locations and their creation order can be utilized to induce the creation time bound of recovered files whose time-related metadata was removed by the file deletion operation. In the relationship addressed by Minnaard, the creation time of a file in a former position is earlier than that of a file in a latter position. Then the lower bound of creation time of a recovered file is the creation time of the neighboring front file with time-related metadata, and its upper bound of creation time is the creation time of the neighboring rear file with time-related metadata. As a case study, the relationship is applied to the creation time bound reconstruction of cfg files, which are deleted but recovered with carving softwares, on the TomTom CPS car navigation device. The relationship is applicable only to non-wraparound situations, in which the file creation time in a former location is earlier than that in a latter location. In other words, the Minnaard's creation order reconstruction method is not applicable to wraparound situations, in which the file creation time in a former location is possibly later than that in a latter location.

In this article, we show that if the Linux FAT32 file allocator traverses the storage space *twice and more*, the file creation time in a former location is possibly later than that in a latter location. Only where the Linux FAT32 file system is consistently used and the file allocator traverses the storage space *once*, the creation time of a file in a former location is earlier than that of a file in a latter location. Also we analyze the accurate creation time bound of files recovered without time-related metadata even where the Linux FAT32 file allocator traverses the storage space *twice*

DOI of original article: http://dx.doi.org/10.1016/j.diin.2014.06.008.

http://dx.doi.org/10.1016/j.diin.2015.09.003
1742-2876/

and more. It is verified analytically that multiple candidates for the creation time bound of each file can be established for the repeated traversals of the file allocator. At most n candidates for creation time bound of each recovered file are established, where n is the number of traversals by the file allocator. Unfortunately, we cannot select one time bound deterministically among the established candidates for all cases because there has been no study helping to differentiate during which traversal the recovered file was allocated. The practical correctness of our analysis is evaluated by examining file allocation patterns of two commercial in-car dashboard cameras: Thinkware Inavi Black and Finedigital FineVu Pro.

Outline of Minnaard's creation order reconstruction method

The storage device managed with the FAT32 file system consists of three parts: reserved area, FAT area, and data area (Carrier, 2005). Reserved area contains file system information such as boot code, partition information, sector size, number of sectors per cluster, etc. The minimum unit of data area logically addressable is called *cluster*. FAT area contains a linked list of cluster offsets allocated to each file. Data area contains *clusters*, where a cluster is a group of consecutive disk sectors with a fixed size. Content of each file is recorded in one or more clusters and a cluster is assigned to recording of at most one file. Through source code analysis of Linux kernel, it is verified that the file allocator on Linux FAT32 file system finds available clusters based on the *next available* algorithm, whenever needing space to be allocated to a new file. In the next available algorithm, available clusters are linearly scanned from the cluster lastly allocated. The reference to the cluster lastly allocated is stored in the FSI_Nxt_Free field of the FSINFO structure, which exists in the reserved area of a FAT32 file system. The file deletion operation of the Linux FAT32 file system does not change the value of the FSI_Nxt_Free field. If the search of the file allocator reaches the end of the data area, the search is wrapped around and restarted at the beginning of the data area. In summary, the file allocator performs only the forward search with wraparound operations when finding available clusters for a new file.

With existing forensic tools (van Eijk and Roeloffs, 2010; Nutter, 2008), the file with a 'cfg' format can be retrieved from TomTom GPS car navigation devices and binary content of the file is interpreted to trace route plans such as a list of entered addresses, a list of recently used addresses, a route destination, a route origin and the last recorded position. If creation time of the cfg files as well as their location-related information is interpreted, we can obtain valuable information on when the device visited a particular place. Forensic tools carve the cfg file in an automatic way and retrieve many incarnations of the cfg file commonly named with 'Mapsetting.cfg'. The latest instance of the file has time-related metadata in a directory entry, but old instances do not because their time-related metadata is removed by overwriting their corresponding directory entries. Creation time bound of carved cfg files is established using a correlation between storage locations of files and their creation order. The method is designed based on a premise that the creation time of a carved cfg file is later than those of former files and earlier than those of latter files on storage device on Linux FAT32 file system. Suppose that a carved cfg file identified with #1475 is located between two undeleted adjacent files with their respective timestamps: 'triplog-2014-02-14.dat' with a creation time of 2014-02-14 17:22:11 and 'triplog-2014-02-15.dat' with a creation time of 2014-02-15 13:14:15 (which are the examples mentioned in Table 1 and Table 2 on page 230 of the Minnaard's article). Then it is asserted that the creation time of the cfg file #1475 is later than 2014-02-14 17:22:11 and earlier than 2014-02-15 13:14:15. That is, the lower bound of file creation time is 2014-02-14 17:22:11 and the upper bound is 2014-02-15 13:14:15. This creation order reconstruction method is applicable only to non-wraparound situations.

Creation time bound analysis for multiple traversals of file allocator

The Minnaard's creation order reconstruction method is not applicable to wraparound situations, in which the file creation time in a former position is possibly later than that in a latter position. Only where the Linux FAT32 file system is consistently used without detachment and reattachment of the storage device and the file allocator traverses the storage space *once*, the file creation time in a former position is earlier than that in a latter position. As addressed in the Discussion section of Minnaard's article, if a removable storage device is exchanged between two computer systems, the file creation time in a former position of the removable storage device is possibly later than that in a latter position. In this section, we also show that if the Linux FAT32 file allocator traverses the storage space *more than once*, the file creation time in a former position is possibly later than that in a latter position.

Fig. 1 shows an example, where data area of a storage device is composed of 20 clusters.[1] Gray boxes denote used clusters unavailable for allocation of new files, while white boxes denote unused clusters available for allocation of new files. In Fig. 1(a), files A, B, C, D and E are sequentially written with respective cluster size of 3, 4, 3, 5 and 3 upon the initial state with all clusters empty. Then file A occupies three clusters from index 1 to index 3. File B occupies four clusters after index 3 (i.e., from index 4 to index 7). File C occupies three clusters after index 7, file D occupies five clusters after index 10, and file E occupies three clusters after index 15. At this moment, the pointer value of FSI_Nxt_Free is 18 which is the cluster index lastly allocated. The second step is to delete files B and D. Then clusters allocated for files B and D (clusters 4, 5, 6, 7, 11, 12, 13, 14 and 15) become available for allocation of other new files, which is depicted in Fig. 1(a). After file deletion operations, the pointer value of FSI_Nxt_Free is not changed. The third step is to write files F, G and H sequentially with

[1] For the sake of simplicity, a storage device with a few clusters is given although the number of clusters in practical storage devices is very large. Also clusters used for directory entries recording file metadata are excluded.

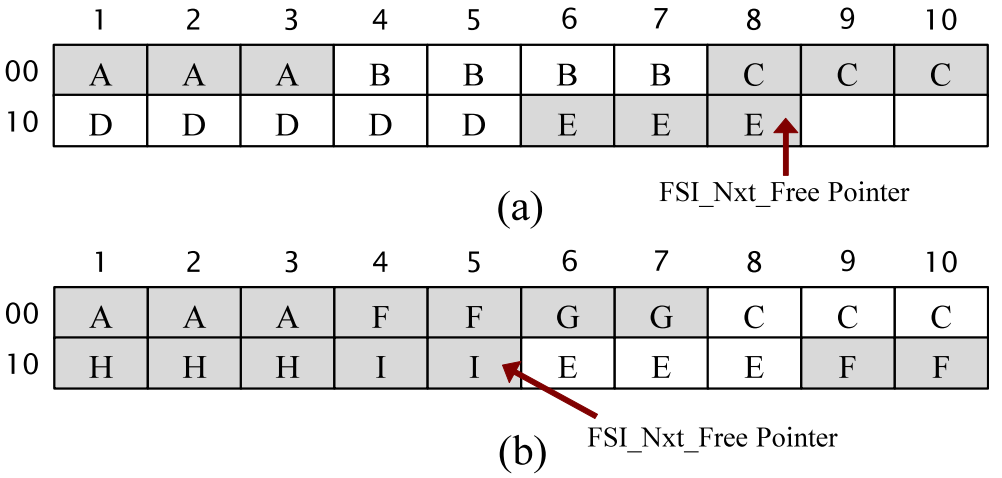

Fig. 1. Working example 1 of Linux FAT32 file system.

respective cluster size of 4, 2 and 3. From index 18 in FSI_Nxt_Free, the file allocator searches four available clusters for allocation of file F but reaches the end of data area. To obtain more available clusters, the file allocator searches available clusters from the beginning of data area. Then file F occupies four clusters 19, 20, 4 and 5 as shown in Fig. 1(b). File G occupies two available clusters behind index 5 lastly allocated. File H occupies three available clusters behind index 7 lastly allocated. Because clusters 8, 9 and 10 are already used for file C, clusters 11, 12 and 13 are allocated to file H. The fourth step is to delete files C and E. Then clusters allocated for file C and E (clusters 8, 9, 10, 16, 17 and 18) become available. The final fifth step is to write file I with cluster size of 2 after index 13 lastly allocated, which is depicted in Fig. 1(b). At this moment, the creation time of the deleted file C is earlier than those of former files such as files F and G. Contents of deleted files can be recovered with carving tools (van Eijk and Roeloffs, 2010; Nutter, 2008) if they remain in unused clusters, but time-related metadata of deleted files stored in directory entries is removed by overwriting their corresponding directory entries for recording metadata of other files.

Fig. 2 shows another example, where file sizes are equal to those in the example of Fig. 1. Like as the example in Fig. 1, the first step is to write files A, B, C, D and E upon the initial state. The second step is to delete files B, C and D. Then clusters allocated for files B, C and D (from cluster 4 to cluster 15) become available, which is depicted in Fig. 2(a). At this moment, the pointer value of FSI_Nxt_Free is 18 which is the cluster index lastly allocated. The third step is to write files F, G and H sequentially. Similarly to the example in Fig. 1, the file allocator reaches the end of data area when searching available clusters for file F and restarts the search operation from the beginning of data area. The fourth step is to delete files H and E. Then clusters allocated for file H and E (clusters 8, 9, 10, 16, 17 and 18) become available. The final fifth step is to write file I and file J sequentially after index 13 lastly allocated, where the cluster size of file J is 3. At this moment depicted in Fig. 2(b), the creation time of the deleted file H is later than that of a latter file F.

Next, we analyze the creation time bound of recovered files for multiple traversals of the Linux FAT32 file allocator. We verify that there are at most n candidates for the creation time bound of each deleted file, where n is the number of traversals by the Linux FAT32 file allocator. In Fig. 3, file X_i denotes some file allocated during the i-th traversal. In the case that file X_1 is deleted *before* the second traversal passes the clusters used for file X_1, file X_2 occupies these clusters during the second traversal. When file X_2 is deleted and recovered, its creation time is bounded only by the files allocated during the second traversal but not by the files allocated during the first traversal. On the contrary, in the case that file X_1 is deleted *after* the second traversal passes the clusters used for file X_1, file X_1 still remains in these clusters during the second traversal because the Linux FAT32 file allocator performs only the forward search with wraparound operation. When file X_1 is recovered, its creation time is bounded only by the files allocated during the first traversal. Unfortunately, there is no information remaining about deleted files and thus we cannot differentiate during which traversal the deleted file was allocated.

Minnaard addressed that if there exist data behind the end of the lastly allocated file, the file allocator traverses the storage space more than once (Minnaard, 2014). Based on this criterion, it can be determined whether the file allocator traverses the storage space more than once or not. However, it is unknown how to determine at which traversal a deleted file was allocated for multiple-traversals cases of the file allocator. There is no reported information to differentiate the two cases of Figs. 1(b) and 2(b). In the example of Fig. 1(b), the deleted file C is allocated during the first traversal, whereas the deleted file H in the example of Fig. 2(b) is allocated during the second traversal. Hence, the creation time of the recovered file C in Fig. 1(b) is bounded by two files A and F neighboring with timestamps among the files allocated during the first traversal, or

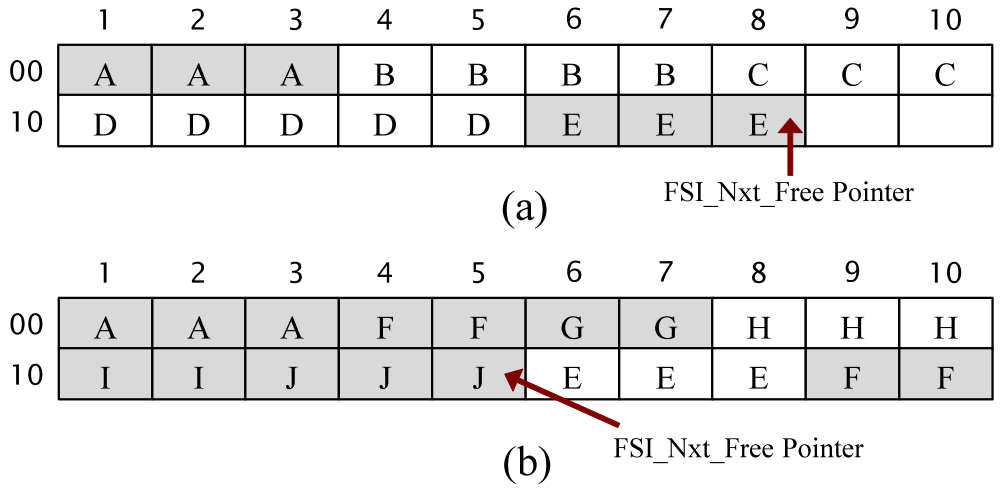

Fig. 2. Working example 2 of Linux FAT32 file system.

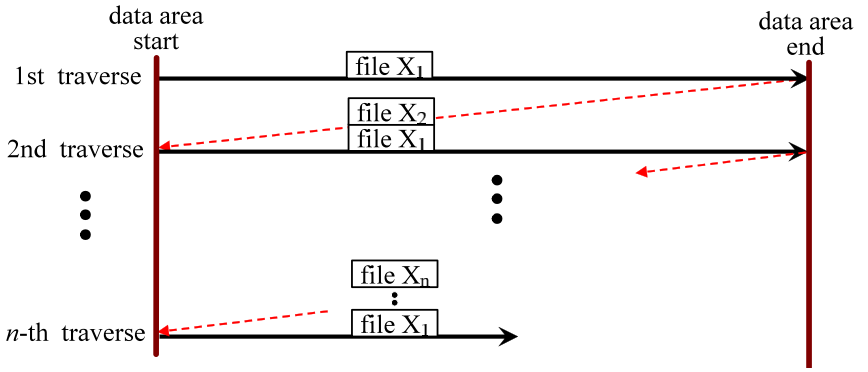

Fig. 3. Illustration for creation time bound of a deleted file.

bounded by two files G and H neighboring with time-stamps among the files allocated during the second traversal. When $T(x)$ denotes the creation time of file x, the creation time bound of the recovered file C in Fig. 1(b) is $T(A) < T(C) < T(F)$ or $T(G) < T(C) < T(H)$. The creation time bound of the recovered file H in Fig. 2(b) is $T(A) < T(H) < T(F)$ or $T(G) < T(H) < T(I)$.

By a similar reason, if a deleted file is recovered when the file allocator performs the n-th traversal, there are n allocation cases for the deleted file, each case representing the allocation during the i-th traversal for $1 \le i \le n$ as shown in Fig. 3. To the best of our knowledge, there has been no study that finds a hint to differentiate at which traversal the deleted file was allocated for multiple traversals of the file allocator. Consequently, the creation time of each deleted file must be bounded by at most n sets of available files separately, each set representing the available files allocated during the i-th traversal for $1 \le i \le n$. In the example of Fig. 1(b), a set $\{A, F\}$ represents the available files allocated during the first traversal and a set $\{F, G, H, I\}$ represents the available files allocated during the second traversal.

Experiments of commercial multimedia digital devices

For practical correctness of our analysis, we examine the file allocation pattern of two commercial in-car dashboard cameras: Inavi Black FXD700 Mach of Thinkware Corporation and FineVu Pro II of Finedigital Corporation. The two devices employ Linux operating system and store recorded video contents in SD flash memory cards. *FTK Imager* tool is used to dump the binary image of a memory card, and *WinHex* tool is used to interpret all binary codes of the dumped image.

In experiments of the Inavi Black device, the default setting of 1920 × 1080 resolution is applied and a FAT32 formatted memory card with 7.47 Gbytes size is used. Its kernel version is 2.6.37. Video contents recorded for a long time are divided into multiple video files with the maximum recording time of 1 min. All video files are constructed to have the AVI structure and the size of 1-min recording video file is about 80 Mbytes. Video files can be classified into multiple groups: driving-time recording video, swaying-moments recording video, manual

recording video, etc. The first wraparound event of the file allocator occurs at about 80 min running. Before the first wraparound event, the file allocator stores the video files sequentially from the beginning of the data area. A video file recorded earlier is located in a former position of data area and that recorded later is located in a latter position, which follows the pattern addressed in the Minnaard's article.

When the file allocator approaches the end of data area, the device automatically deletes some files in order to prepare enough available space. When selecting the files being deleted, driving-time recording video files are preferentially selected rather than the other class video files such as swaying-moments video files that may include critical scenes caused by shocking of external contacts. Among driving-time video files, those with earlier creation time are preferentially deleted. As a result, the files being deleted are selected non-linearly from the beginning of data area and thus available spaces generated from the deleted files are not always neighboring. After the first wraparound event, the file allocator searches for available space from the beginning of data area. The file allocator assigns a small available space to a large video file according to the *next available* algorithm. We examined the file allocation pattern until the fifth wraparound event occurs, and confirmed that the file allocator follows the pattern described in Section (Creation time bound analysis for multiple traversals of file allocator).

In experiments of the FineVu device with kernel version of 2.6.18, 1920 × 1080 resolution is applied and a FAT32 formatted memory card with 14.95 Gbytes size is used. Video contents recorded for a long time are divided into multiple video files with the maximum recording time of 3 min. All video files have the AVI structure and the size of 3-min recording video file is about 180 Mbytes. The first wraparound event occurs at about 150 min running. The rest file allocation behavior of this device is identical to that of the Inavi Black device. We confirmed that the file allocator of this device follows the pattern given by Minnaard before the first wraparound event, and follows our analysis given in Section (Creation time bound analysis for multiple traversals of file allocator) between the first wraparound event and the fifth wraparound event.

Directory entries store metadata of each file, including the file creation time and the offset of clusters assigned to file storing. If the metadata of a deleted file remains in its corresponding directory entry, the exact creation time of a deleted but recovered file can be retrieved by matching the cluster offset of the recovered file with metadata of all directory entries. However, the file allocators of these devices overwrite the deleted directory entries with metadata of new files according to the *first available* algorithm that scans available entry linearly from the beginning whenever needing a new entry. In this case, the directory entries of deleted files are overwritten in most cases.

Additionally, we examine the file allocation pattern of a FAT32 formatted memory card upon PCs with Windows XP and Windows 7. The Windows FAT32 file allocator searches for available space linearly from the beginning of data area after each wraparound event. As addressed in the Minnaard's article, the Windows FAT32 file allocator searches according to the *next fit* algorithm that skips available spaces smaller than the size of a file being stored. The following behavior is not mentioned in the Minnaard's article. The Windows FAT32 file system searches for available *directory entry* according to a variant of the next available algorithm, whereas the Linux FAT32 file system does according to the *pure* first available algorithm. The Windows FAT32 file system searches according to the next available algorithm until the last directory entry in a cluster is assigned to metadata recording of a new file. After the last directory entry is assigned, it searches for available entry from the first directory entry of the cluster according to the first available algorithm. Assigned entries are released and become available if their corresponding files are deleted. When all entries of the cluster are unavailable, another available cluster is allotted to further recording of directory entries. Similarly, available entries of the newly allotted cluster are assigned according to the next available algorithm until the last entry of the cluster is assigned. In the case of Windows FAT32 file systems, the metadata of recently deleted files remains in their corresponding directory entries in most cases.

Acknowledgments

This research was supported by the Public Welfare & Safety Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2012M3A2A1051118).

References

Carrier B. File system forensic analysis. Addison-Wesley; 2005.
Minnaard W. The Linux FAT32 allocator and file creation order reconstruction. Digit Investig 2014;11(4):224–33.
Nutter B. Pinpointing TomTom location records: a forensic analysis. Digit Investig September 2008;5(1–2):10–8.
van Eijk O, Roeloffs M. Forensic acquisition and analysis of the random access memory of TomTom GPS navigation systems. Digit Investig May 2010;6(3–4):178–88.

Wan Yeon Lee
Dept. of Computer Science,
Dongduk Women's University,
Seoul 136-714,
South Korea

Hyuckmin Kwon, Heejo Lee*
Dept. of Computer Science and Engineering,
Korea University,
Seoul 136-713,
South Korea

* Corresponding author.
E-mail address: heejo@korea.ac.kr (H. Lee)

16 April 2015
Available online 21 October 2015