Digital Investigation 20 (2017) S92–S98

Contents lists available at ScienceDirect

Digital Investigation

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

DFRWS 2017 Europe — Proceedings of the Fourth Annual DFRWS Europe

Selective deletion of non-relevant data

Christian Zoubek [a, *], Konstantin Sack [b, **]

[a] Department of Computer Science, Technische Hochschule Nürnberg Georg Simon Ohm, Nuremberg, Germany
[b] Department of Computer Science, Friedrich Alexander Universität Erlangen-Nürnberg (FAU), Erlangen, Germany

A R T I C L E I N F O

Article history:
Received 26 January 2017
Accepted 26 January 2017

Keywords:
Digital forensic
Image
Wiping
Private personal data
Computer forensic
Selective deletion

A B S T R A C T

In crime investigation, especially in computer crime investigations, seizure and analysis of digital evidence is a de facto standard procedure. To prevent alterations on the original digital evidence a so called (bitwise) image is created. In this image all data contained on the digital evidence is stored, even non-relevant content or content with the risk of associated privacy violations. In countries with an elaborate protection of private personal or confidential data, this data has to be securely deleted from the image. Facing the rising request for a selective deleting functionality, common problems, limitations and requirements for a tool selectively deleting non-relevant data are outlined in this paper. For demonstration purposes, a prototype as a plugin for the Digital Forensics Framework (DFF) was implemented. The design of the implementation, some considerations as well as a comparison between a commercial tool and the evaluation of the implemented wiping strategy are presented.

Introduction

Motivation

In a 2.0 environment with an increasing IT-knowledge of users and the trend of digitalization, electronic data has become a significant role in today's life. From the point of view of crime investigators, evidence in different forms can be found on digital media like computers, mobiles or USB-sticks. In addition to the rising challenge of mass data, law enforcement agencies have to face more specialized defense lawyers and prosecutors. During a criminal investigation process, even in investigations that are of non-electronic nature, the search and seizure of digital devices is a standard procedure. In regard to the forensic soundness of the digital evidence, best practice guides like "The Good Practice Guide for Computer-Based Electronic Evidence" were introduced to forensic practitioners. The creation of a so called image (bitwise copy) of the original digital evidence has become common.

Legal considerations

The idea behind the tool, which is proposed in Section Implementation, is based on German laws limiting the access and usage of information, for example the data privacy law (corresponding to the data protection act 1998 [UK], different information privacy and data protection laws in the US or Article 8 of the European Convention on Human Rights (ECHR) providing the right to respect "private and family life, his home and his correspondence"). Especially German law enforcement has strict legal specifications about the seizure of data and its utilization for criminal investigations. Content of data that leads to knowledge about one's way of life are prohibited to be utilized. §100a of the German code of criminal procedure (*German: Strafprozessordnung*) states in subparagraph 4, that recordings of such data are mandatory to be deleted immediately and that both, the obtainment and its deletion, are to be documented.

This is derived by the so called "Elfes-Urteil", a decision made by the German Federal Constitutional Court (*German: Bundesverfassungsgericht*) in 1957. It strictly states that one's data is part of a human being's inviolable dignity and freedom (Moser-Knierim, 2014). In consequence, law enforcement is forced to spare non-relevant data blocks to a case when imaging a suspect's volumes. As this guideline can hardly be executed in reality, the selective deletion approach provides a possibility to fulfill the legal requirement afterwards.

Even though this legal requirement is expected to be fulfilled, its implementation is not yet actively pursued. If asked, the typical

* Corresponding author.
** Corresponding author.
E-mail addresses: christian.zoubek@th-nuernberg.de (C. Zoubek), konstantin.sack@fau.de (K. Sack).

http://dx.doi.org/10.1016/j.diin.2017.01.006

answer of local German criminal investigation offices is, that there exists no established tool that is yet able to selectively delete data, neither during acquisition nor afterwards (Lg frankfurt, 2004; für den Datenschutz, 2012). The main argument to not implement this feature is that the deletion of content always modifies images. As the modification of evidence damages the integrity of the very same and thus applicability in court may be endangered.

Expectations to a selective deletion tool

For a secure deletion of objects, a lot of specialized wiping tools exist. Carrier describes their function as: ‘Most wiping, or “secure delete” tools operate in the content category and write zeros or random data to the data units that a file allocated or to all unused data units’ (Carrier, 2005). Garfinkel stated, that tools which overwrite information are the ‘oldest and most common forms of anti-forensic tools’ (Garfinkel, 2007). Besides the functionality to overwrite entire media or individual files, the possibility to delete metadata as a possible source for creating timelines about the user's activity should be given (Garfinkel, 2007). In the context of personally identifiable information, Pan et al. claimed, that ‘Metadata can provide a place for sensitive data to reside, but it is not particularly easy to remove’ (Pan et al., 2010). Following these opinions, we believe that forensically sound deletion tools should include the possibility to securely delete all remaining associated information, for instance meta information given by file tables corresponding to the deleted files. On top of that, a detailed report or log file helps to a) document the deletion of selected files and b) to authenticate the integrity and the provenance of the image and the remaining files. Following the NIST Special Publication 800-88 (NIST, 2014) the presented deletion tool operates with a single overwrite mechanism.

Related work

Castiglione et al. (2011) drafted an approach for the ‘Automatic, Selective and Secure Deletion of Digital Evidence’. They proposed different methods for the wiping implementation in regard of leaving possible traces for digital investigations by anomalies or suspicious patterns on the disk. By defining the wiping of the physical locations of the data, the removal of associated metadata and ‘unwanted’ or ‘suspicious’ traces as theoretical necessity, their case study only faced the deletion of the physical locations.

In contrast to a selective deletion, Stüttgen et al. (2013) designed a DFF module for the creation of selective images. Contrary to full bitwise copies, the selective imaging allows the creation of partial images with only selected data. The main difference between this approach and ours is that selective imaging is done during the acquisition process, while the selective deletion is done after the imaging process.

Short introduction of tool

The tool for a selective deletion is realized as a plugin for the Digital Forensics Framework.[1] As the tool was intended to investigate whether technically a secure selective deletion is possible, this plugin is bounded to NTFS as the only file system. Currently only raw images are supported.

This tool only provides a technical solution to the deletion process and therefore is limited in its support for the investigator in some ways. First, the search of sensitive data is done manually by the investigator. The authors don't know of any possible way to automatically detect such data. Yet this question is out of the scope of this paper.

Second, the tool works at file system level. If routines of an operating system shadows certain data or meta data, this tool cannot directly find corresponding entries. Using the *Matcher*-module, which is described later in greater detail, 1:1-copies of files are able to be detected in an automatic fashion. Hence, slightly changed data still needs to be detected by the user.

Outline

The focus of this paper lays on the implementation of a forensically sound selective deletion tool for images of NTFS[2]-partitioned disks. Precise criteria for forensically sound deletion and provenance verification techniques will be defined in Section Expectations to a selective deletion tool and further described in Section Selective deletion. Section Implementation points out specific functions of the implementation. In Section Evaluation the functions are tested and compared to these of an existing forensic tool (X-Ways Forensics). Section Conclusion and future work finally concludes the paper.

Selective deletion

For forensic purposes it is essential to define a proper understanding of the irrecoverable deletion of data objects. By using the deletion method of common operating systems (Windows or OS X), only the entry in the file allocation table (Master File Table for NTFS — Inode-table for Ext) is deleted or marked with an unused flag. The data itself still resides on disk until the space is reused by the system. This type of deletion is not proper for forensic use, because data content could be recovered easily using open source tools like the forensic suite Autopsy.

Additionally, reformatting hard disks usually just rewrites partition tables. In consequence, data of former formats is still in place and will remain until newer formats overwrite these certain blocks. So called file-carvers yield the possibility to detect and carve remnants. As those file-carvers miss the opportunity to recover meta data corresponding to the installed file system, a file carver cannot find out whether carved data belongs to the latest format or an older, unknown format.

To counter the latter case, some tools exist that clear whole formats or disks by overwriting all data with random numbers or zeros. There are also numerous tools to erase individual files more or less securely. These tools commonly operate in the content category (Carrier, 2005). All blocks that are allocated as content of data are located and wiped by overwriting them with nonsense data.

Still, this method is not forensically save enough. Depending on the used file system most of a file's metadata still remains on disk. Even if all data content is securely erased, metadata still yields information about the user. For instance, a suspect went on a vacation some time before seizure, took some pictures of visited places and saved these on their hard disk. Structurally saved directories and unique names like “Vacation 2016 — Lake Constance” give enough information about one's private life. Correctly interpreting the already discussed “Elfes”-decision of the German Federal Constitutional Court examining, saving or in any other way utilizing such data is prohibited, exceptionally said data is case relevant.

In file systems like *NTFS* the file's metadata is saved in special data structures with some of it located directly in the *$MFT*[3] the heart of the file system. Selective deletion in *NTFS* proves difficult as

[1] Tool and a brief manual can be found here: https://www1.informatik.uni-erlangen.de/content/selective-deletion.

[2] Microsoft's New Technology File System.

[3] For this paper the main focus was Microsoft's *NTFS*.

wiping the data may leave the file system in a corrupt state. Hence the modification of affected data structures is the better choice. To enhance the access time to files, modern file systems usually sort file names in a B-tree.

In *NTFS*, the B-trees of all the directories are saved in data structures that are separated from the file's content. Each node of the B-tree is represented by a so called cluster of a specific size (see Fig. 1 as a simple example of a B-tree). Within such a cluster, the filenames indicate the position of each entry within the tree. A flag within an entry declares whether a subtree is attached to the entry. The B-tree has to be balanced and sorted, hence the entries in a subtree are always lexicographically smaller than the parent entry. For example, in Fig. 1 the 2nd leaf holds entries lower than the corresponding head entry, in this case 11.

Just clearing a filename may impact the tree in such a way that a whole subtree may be irrecoverable. For instance, wiping entry 11, the 2nd leaf would not be recoverable anymore, as no entry in the head points to this leaf. The algorithm used find entries in a B-tree is now forced to descend at entry 16 for every entry with a number between 5 and 16. As entry 16 only points to the third leaf, only these entries can be found. Hence entries 6,7,8,9 and 10 cannot be found by this routine anymore.

In Section Implementation a possible solution for a more secure deletion is shown. Here the deletion-module finds the greatest value in the left subtree to overwrite entry 11.

Another major aspect of a valid selective deletion is integrity. Just wiping and clearing case irrelevant data is not enough. As evidence may be needed in court, the investigator has to guarantee the integrity of the remaining data. A change could lead to a false interpretation of data, either positively or negatively for the suspect.

Two approaches that complement each other are presented in greater detail in Section Implementation. First, all changes in data – both deletion and modification – are documented. The resulting log file keeps every alteration of hardware blocks with the exact address. The alteration is linked to the deleted file.

The other approach uses Merkle trees to verify the integrity of remaining data. Two trees are to be created, one before the selected deletion process and the other right afterwards. By comparing both trees, differences indicate modifications of hardware blocks. When matching the difference of both trees with the log file received in the first step, alterations of the same hardware blocks are to be received. If the results differ, an undocumented alteration can be assumed. Hence the integrity of this image cannot be guaranteed anymore.

To summarize aforementioned problems, the forensically sound implementation of a selective deletion tool should at least be able to

- detect all blocks of content of data and completely erase them
- find corresponding metadata and delete it
- leave the file system in a working state
- meticulously document every modification and deletion
- ensure integrity of remaining data

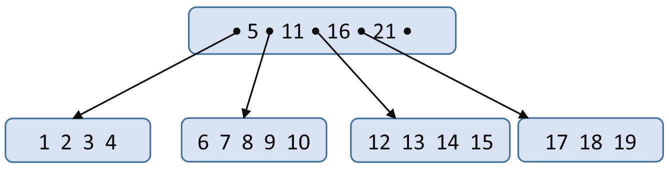


Fig. 1. Exemplary B-tree with overall 19 entries.

Implementation

The prototypical implementation is realized as a plugin to the *Digital Forensics Framework (DFF)*, which is an open source forensics toolkit.[4] For popularity reasons *NTFS* is one of the most analyzed file systems in digital forensic examinations, therefore the decision to implement a selective deletion to handle that file system was made. In this paper, *NTFS* is only explained rudimentary. Special data structures (e.g. fixup values), run lists, attributes, structure of *MFT*-entries, virtual and logical clusters and many more to consider when modifying a *NTFS* would push the boundaries of this paper. Those are explained in great detail in literature like Carrier (2005). The plugin for a selective deletion tool is divided into five modules, whereas some modules are not necessary for a selective deletion but are an addition for an investigator to help find all files. The modules are named as follows

- Selector
- Matcher
- Carver-Cleaner
- Hashcalculator
- Deletion module

The *Selector* module interprets existing partition tables, *NTFS*-partitions and spans the corresponding directory structures. In addition, this module is able to carve the *MFT* for files which were only deleted by unsetting the *in use* flag. The Matcher supports the user to find duplicates and files linked to the same data content. The third module is used after the execution of a carving tool. Files, that do not belong to installed partitions, are flagged accordingly. The purpose of the Hashcalculator is to verify the integrity of remaining data after a selective deletion. The main functionality of the prototype is realized in the deletion module. Each directory and file is checked whether it is tagged for deletion. If so, data and corresponding metadata is deleted in a way, that the rest of the partition is still consistent and usable.

To ensure a forensically sound process, the main modules produce logfiles including integrity information like the path of the deleted data objects and the memory area according to the containing partition.

Hashcalculator

The function of the *Hashcalculator Module* is to ensure the integrity of the court exhibit (digital evidence). To obtain the forensically soundness, a Merkle tree (Merkle, 1982) with user defined options for the block size and the number of children per node in the tree structure is used in combination with the calculation of MD5 hash values. A Merkle tree, also called hash tree, calculates hash values out of blocks. Then, a predefined number of hash sums is grouped together to calculate a new hash value out of these hash values. This is done subsequently until only one hash value, the so called top hash, is reached.

For the correctness of the hash calculation the module spans an entire balanced tree. The minimal height of the tree is calculated using Eq. (1). The result of the equation is used in Eq. (2) to determine the amount of zero blocks needed to be added to finally balance the tree

$$\text{tree height} = \left\lceil \frac{\log_{10}\left(\frac{\text{image size}}{\text{block size}}\right)}{\log_{10}(\text{children per node})} \right\rceil, \tag{1}$$

[4] The source can be found here: *blind review.*

$$\#leaves = (\text{children per node})^{\text{tree height}} \quad (2)$$

The tree is saved in a tuple of XML-files.[5] With the calculation of hash trees before and after the deletion process, a user is able to easily identify modified blocks. The user compares hash values of a certain level across both trees. Different values indicate that data in the range of the affected block is modified. The descension to the following subtree, holding the modified block, further narrows the range to search.

Matcher

The *matcher* module carries out two functions. One is an expansion of the forensics application of a secure deletion in a manner to leave a coherent file system after deletion. In *NTFS*, like in most modern file systems, it is possible to use more than one hard link to specific data content. A hard link connects *MFT* entries with corresponding data blocks. Using several hard links to the same data blocks counters copying the very same content to different locations. Hence it increases performance and HDD lifetime. Hard links don't necessarily use the same file name to content, as data content and metadata are strictly separated.

In the context of a selective deletion this yields an issue. If an investigator wants to delete a specific file while he missed another hard link that points to the very same file content, zeroing the content corrupts the file system in some way. Drivers are still able to see the hard link (including all of the metadata like file name and ownership) but are unable to open the specific file. In tests, the operating system popped up an error message notifying the user about a corrupt I/O-device. The *matcher* module can counter this by detecting these cases and flagging them appropriately so that a user is able to find such double pointers.

Another function of this module is to find real duplicates of files. This is important, because it is more and more common to save data, especially if sensitive, in more than just one location. With the sheer amount of data a volume is able to hold today, there is a chance that the user of this plugin may miss all duplicates of a file. To aid the user, a block hasher is integrated to this module. It calculates hash values of a certain number of blocks of files that are flagged to be deleted. These blocks are then compared to blocks of all files with the very same file size. In cases, in which both files are identical the duplicate is flagged accordingly.

Additionally to flagging all duplicates and hard link copies, this module saves detailed information in a log file. The log file allows a user to reconstruct as to why the algorithm flagged individual files as duplicate.

Carver-Cleaner

The Carver-Cleaner-module is not mandatory if only a selective deletion is wanted. It implements the requirement made by the Elfes-decision (see Section Introduction) to spare private data. With carvers, the possibility to find data of older formats arises. Unfortunately, also many so called false positives may be found. After the application of such a carver, this module probes data sections of carved files against information found in existing *MFT*s.

The algorithm checks whether data blocks of carved files overlap with blocks marked by entries of the *MFT*. If only one block overlaps, the deletion of it could corrupt the file system, hence only files without overlapping content is marked accordingly. This way, false positives and data belonging to file systems are spared. A possible damage to the file system is avoided.

Deletion

The main functionality of the whole plugin is implemented in the Deletion-module. As metadata and content of data is separated, this module is divided into two parts. The first part just wipes content by zeroing corresponding blocks. The second one decides whether data in metadata is wiped or modified in a way that a file system stays intact. For example, a modification is applied when changing B-tree structures.

As the *DFF* is a forensics toolkit, it is prohibited to modify images directly. To enforce this rule, volumes in *DFF* are always mounted read-only. The first step to allow modification is to obtain the origin path by using *DFF*-call-routines. Following this, the correct path to the image is handed to C++ routines to open the path with write access.

A synchronization of pointers to data is additionally needed to prevent alteration of wrong sections in the volume. The deletion is done recursively. The root directory is the starting point of this algorithm. Here, the algorithm descends recursively to subfolders. In each directory, entries marked as *delete* are further inspected.

Every block of data or metadata corresponding to a file/directory, that is declared to be deleted, is located and saved in an internal data structure. This data structure saves, which block is to be deleted and which block needs modification. After inspecting a whole directory, it is checked whether any file or sub directory was changed in the process. The latter case implies an update of the current directory's B-tree.

Updating the B-tree proves partially difficult (see Section Selective deletion). If done improper, a whole subtree in *NTFS*'s internal representation of a B-tree could be lost. In consequence, some or all files cannot be found by interpreting the B-trees as designated. Therefore, this algorithm first checks, whether corresponding entries in the B-tree are at leaf level or node level.

The first case is fairly easy to solve:

- remember the size of entry
- temporarily save everything after the entry to the end of the leaf
- overwrite the deleted entry with saved entries
- after the last entry, overwrite data of the size of the deleted entry with zeros

The other case implicates some specialties. At first, the greatest value in the left sub tree is identified by recursively descending the tree. This entry is locally saved and temporarily deleted by calling the aforementioned routine. Depending on the size difference between the saved entry and the one to be deleted, the marked entry is overwritten by the saved entry. If the subtree is obsolete after deletion, the child-flag in the corresponding entry is deleted.

Some more specialties are to be considered regarding *NTFS*. Each cluster is only of specific size, therefore under- or overflows are possible. Updating data fields about the sizes used within a cluster and updating fixup values is also mandatory. How to possibly implement solutions to these problems would push the boundaries of this paper. These can be directly found in the source code.

When modifying data on the image, a critical section occurs. In a perfect scenario, changing content, metadata and B-tree updates would be applied in parallel and without interruptions. As computers and programs could crash, this module tries to keep the critical section as small as possible. Hence before changing a file or a directory, all needed alterations are booked in a data structure before. If all modifications within a directory are booked, the application of these are done at once. Even though a critical section still exists, it is far smaller than applying alterations immediately.

[5] Extensible Markup Language.

One thing to mention is that the gapless documentation of every modification and deletion of data still yields information about the user. As already mentioned in Section Introduction, §100a of the German code of criminal procedure forces a documentation, which keeps information about obtainment and deletion of data. For this paper the selective deletion and the integrity of remaining data was the focus. Hence the question, how deep documentation about the deletion process may go, is not answered. Here the deletion of every single file is documented by the whole path and the corresponding data blocks. Considering the Elfes decision such documentation may already hurt one's basic human rights, see Section Introduction.

Evaluation

Two different test scenarios built the basis to evaluate the prototype. The first scenario is used to verify the proper functioning, while the second scenario points out the benefits of the implementation in comparison to an existing deletion function used by the forensic tool X-Ways Forensics.

For the first scenario seven test cases were designed, which cover a small variety of typical use cases:

- standard allocations of data on devices (e.g. saving of picture files)
- non-standard allocations like resident files[6]
- reformatted devices with data transparent to the new format

Two of them are especially eligible to show that due to the reconstruction of B-Trees even deletion in system folders is possible without creating any inconsistent state of the device. In all test cases the proper function of the selector, matcher, Carver-cleaner and deletion module could be verified. Deleting a whole user directory on a boot device forces a windows routine to recreate the user directories temporarily. A popped up message at startup informs the user accordingly. Even though the user was created temporarily, there was no evidence that this windows routine could recover any deleted file. Hence we assume that the selective deletion works proper in this case, but there is additional future work to do underpin this assumption.

The second scenario is used to compare this implementation with a widely used forensic tool, which also allows the selective deletion of files. A USB-stick with a capacity of 3.6 GB and a *NTFS* served as a basis. Some logical folders and files were created for testing purpose. The tasks for the forensic software X-Ways Forensics and our implementation was to delete a folder including all child objects. Furthermore a jpg-file and all of its hash-similar duplicates on the device were to be found and deleted. To point out the enhancement of this prototype, the first step is to show the basic deletion function of X-Ways 18.9 (X.-G. AG, 2016).

For a selective deletion in images in X-Ways the image needs to be added to a case. To delete a file the file is to be "excluded". The same applies to folders with the option of recursive exclusion of all child objects. Finding duplicates is an embedded function in X-Ways which works either with Hash/Photo-DNA-criteria or by name of the object. Unfortunately every duplicate has to be marked as "to be excluded" manually. A deletion of the objects is achieved by the creation of a new image of the device omitting the excluded files.

X-Ways offers an option to mark the first bytes of the data content of excluded files user-defined watermarks. The rest of the file is deleted by zeroing all allocated blocks. However all the data's metadata including names, mac-times and other information remain on the new image. Fig. 2 exemplifies the deletion process in X-Ways (File A and B are to be deleted). With only the file content to be deleted, X-Ways doesn't touch the *$MFT*. Furthermore X-Ways doesn't create a dedicated report including the listing of the excluded data object; a log-report with information concerning the image creation itself is generated.

The comparison between X-Ways and this *DFF* plugin is done by the following setup. Three hash-wise identical files called kipo1.jpg in three different subfolders on the device are to be wiped. Additionally a whole directory is deleted recursively so that all files and subdirectories within are cleared.

The verification of the deletion processes was done by using FTK Imager 3.4.0.5 (Ftk Imager, 2016) and Hex-Editor MX (nexsoft.de, 2016). With both tools the physical sectors of all affected files and folders (including child-objects) were checked to be zero. To cross-check the deletion function of X-Ways a user defined watermark was written to all overwritten lead bytes of the wiped physical sectors in a second test. A post-mortem carving analysis using the forensic tool FTK 5.6.1.55 (Forensic toolkit, 2016) finalized the verification process. As a result of X-Ways' implementation all metainformation and system structures of the deleted files and folder remained on the disk, while all data content itself was irrecoverably removed.

To underline the innovation of the proposed implementation the same test scenario was applied to the proposed prototype. After selecting one occurrence of kipo1.jpg the matcher module marked the duplicates in all other folders with a separate log file highlighting correlating files. By tagging the marked files and the folder as "delete", the deletion module was executed. This module produces a detailed log file about the selective deletion as well. It includes a list of all deleted files/directories, modified physical locations and information about corresponding B-tree updates. In

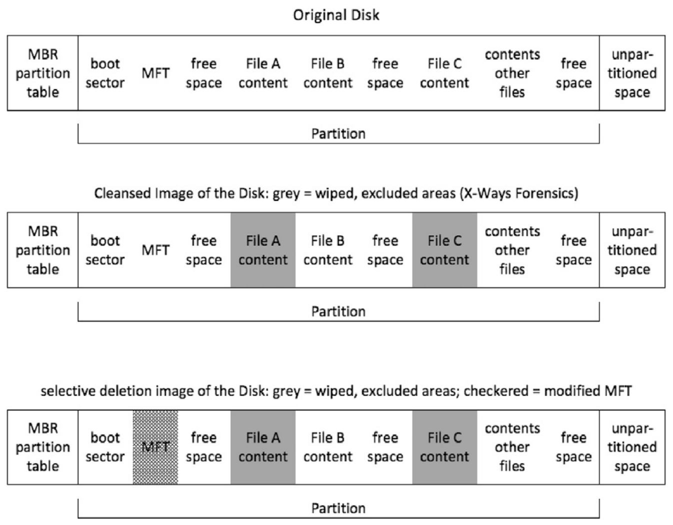
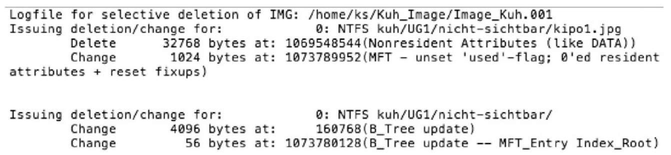

Original Disk
MBR partition table
boot sector
MFT
free space
File A content
File B content
free space
File C content
contents other files
free space
unpar-titioned space
Partition
Cleansed Image of the Disk: grey = wiped, excluded areas (X-Ways Forensics)
selective deletion image of the Disk: grey = wiped, excluded areas; checkered = modified MFT

Fig. 2. Comparison between the original image, the cleansed image (`X-Ways)` and our implementation.



```
Logfile for selective deletion of IMG: /home/ks/Kuh_Image/Image_Kuh.001
Issuing deletion/change for:            0: NTFS kuh/UG1/nicht-sichtbar/kipo1.jpg
        Delete     32768 bytes at: 1069548544(Nonresident Attributes (like DATA))
        Change      1024 bytes at: 1073789952(MFT - unset 'used'-flag; 0'ed resident
attributes + reset fixups)


Issuing deletion/change for:            0: NTFS kuh/UG1/nicht-sichtbar/
        Change      4096 bytes at:      160768(B_Tree update)
        Change        56 bytes at: 1073780128(B_Tree update -- MFT_Entry Index_Root)
```

Fig. 3. Snippet of the log file created by the deletion module.

[6] In *NTFS* small files are directly allocated to the *MFT*.

Fig. 3 a snippet of the log file is shown. The modification of B-tree entries within the *MFT* is shown by the checkered background in Fig. 2.

Fig. 4 illustrates the change of the file's data content. Like the other tool, this prototype wipes content data by overwriting everything with zeros.

The second benefit of our implementation is presented in Fig. 5. In contrast to X-Ways our implementation not only wipes the physical location of the selected file's content, but furthermore deletes the corresponding *MFT*-entry. As a *MFT* would be corrupt if a whole entry is just wiped, the header still remains almost the same. Only data describing the *MFT* entry itself (e.g. used size of entry) are modified accordingly.

Another difference between the commercial tool and the proposed one is the handling of directory entries. As already described B-trees are saved in data structures that are not directly connected to a file's content. To span a directory a driver reads such data structure. In consequence, changes made in this data structure may lead to a damaged file system. In Fig. 6 the update of a B-tree is shown. Just wiping could yield a corrupt file system as mentioned in Section Selective deletion. Hence, the presented algorithm modifies the corresponding cluster in a way that the whole directory is still readable. As presented in Fig. 6, the appropriate data sections are overwritten by moving the rest of the cluster.

While X-Ways creates an additional (skeleton) image as output, our implementation directly deletes the desired parts in the original image. The correctness of our implementation is verified using the same tools, FTK Imager and Hex-Editor MX. The Hex-Editor was used to prove that the physical sectors of the three files and the folder (including all child-objects) were overwritten with zeros. Furthermore using the Hex-Editor it was verified that B-trees and entries in the *MFT* were adapted appropriately. A last verification,

```
3fc00400  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 01 00 60  |......JFIF.....`|
3fc00410  00 60 00 00 ff e1 00 68  45 78 69 66 00 00 4d 4d  |.`.....hExif..MM|
3fc00420  00 2a 00 00 00 08 00 04  01 1a 00 05 00 00 00 01  |.*..............|
3fc00430  00 00 00 3e 01 1b 00 05  00 00 00 01 00 00 00 46  |...>...........F|
3fc00440  01 28 00 03 00 00 00 01  00 02 00 00 01 31 00 02  |.(...........1..|
3fc00450  00 00 00 12 00 00 00 4e  00 00 00 00 00 00 00 60  |.......N.......`|
3fc00460  00 00 00 01 00 00 00 60  00 00 00 01 50 61 69 6e  |.......`....Pain|
3fc00470  74 2e 4e 45 54 20 76 33  2e 35 2e 31 31 00 ff db  |t.NET v3.5.11...|
3fc00480  00 43 00 01 01 01 01 01  01 01 01 01 01 01 01 01  |.C..............|
3fc00490  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
*
3fc004c0  01 01 01 ff db 00 43 01  01 01 01 01 01 01 01 01  |......C.........|
3fc004d0  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
*
3fc00500  01 01 01 01 01 01 01 01  ff c0 00 11 08 01 2a 01  |..............*.|
```

(a) Content of file before deletion

```
3fc00400  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
3fc01400
```

(b) Content of file after deletion

Fig. 4. Deletion of file kipo1.png. Data content is wiped.

```
4000bc00  46 49 4c 45 30 00 03 00  43 4f 80 00 00 00 00 00  |FILE0...CO......|
4000bc10  01 00 01 00 38 00 01 00  58 01 00 00 00 04 00 00  |....8...X.......|
4000bc20  00 00 00 00 00 00 00 00  03 00 00 00 2e 00 00 00  |................|
4000bc30  02 00 00 00 00 00 00 00  10 00 00 00 60 00 00 00  |............`...|
4000bc40  00 00 00 00 00 00 00 00  48 00 00 00 18 00 00 00  |........H.......|
4000bc50  60 a7 a1 08 43 6a d1 01  00 4f 63 10 3f ff cf 01  |`...Cj...Oc.?...|
4000bc60  60 a7 a1 08 43 6a d1 01  60 a7 a1 08 43 6a d1 01  |`...Cj..`...Cj..|
4000bc70  20 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  | ...............|
4000bc80  00 00 00 00 06 01 00 00  00 00 00 00 00 00 00 00  |................|
4000bc90  00 00 00 00 00 00 00 00  30 00 00 00 70 00 00 00  |........0...p...|
4000bca0  00 00 00 00 00 00 02 00  54 00 00 00 18 00 01 00  |........T.......|
4000bcb0  24 00 00 00 00 00 01 00  60 a7 a1 08 43 6a d1 01  |$.......`...Cj..|
4000bcc0  60 a7 a1 08 43 6a d1 01  60 a7 a1 08 43 6a d1 01  |`...Cj..`...Cj..|
4000bcd0  60 a7 a1 08 43 6a d1 01  00 80 00 00 00 00 00 00  |`...Cj..........|
4000bce0  00 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00  |........ .......|
4000bcf0  09 03 6b 00 69 00 70 00  6f 00 31 00 2e 00 6a 00  |..k.i.p.o.1...j.|
4000bd00  70 00 67 00 00 00 00 00  80 00 00 00 48 00 00 00  |p.g.........H...|
4000bd10  01 00 00 00 00 00 01 00  00 00 00 00 00 00 00 00  |................|
```

(a) Hexdump of original *MFT*

```
4000bc00  46 49 4c 45 30 00 03 00  43 4f 80 00 00 00 00 00  |FILE0...CO......|
4000bc10  01 00 01 00 38 00 00 00  58 01 00 00 00 04 00 00  |....8...X.......|
4000bc20  00 00 00 00 00 00 00 00  03 00 00 00 2e 00 00 00  |................|
4000bc30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
4000c000  46 49 4c 45 30 00 03 00  c5 50 80 00 00 00 00 00  |FILE0....P......|
4000c010  01 00 01 00 38 00 01 00  58 01 00 00 00 04 00 00  |....8...X.......|
4000c020  00 00 00 00 00 00 00 00  03 00 00 00 2f 00 00 00  |............/...|
4000c030  02 00 00 00 00 00 00 00  10 00 00 00 60 00 00 00  |............`...|
4000c040  00 00 00 00 00 00 00 00  48 00 00 00 18 00 00 00  |........H.......|
4000c050  20 6a a6 08 43 6a d1 01  00 49 eb 38 3f ff cf 01  | j..Cj...I.8?...|
4000c060  20 6a a6 08 43 6a d1 01  20 6a a6 08 43 6a d1 01  | j..Cj.. j..Cj..|
4000c070  20 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  | ...............|
4000c080  00 00 00 00 06 01 00 00  00 00 00 00 00 00 00 00  |................|
4000c090  00 00 00 00 00 00 00 00  30 00 00 00 70 00 00 00  |........0...p...|
4000c0a0  00 00 00 00 00 00 02 00  54 00 00 00 18 00 01 00  |........T.......|
4000c0b0  24 00 00 00 00 00 01 00  20 6a a6 08 43 6a d1 01  |$....... j..Cj..|
4000c0c0  20 6a a6 08 43 6a d1 01  20 6a a6 08 43 6a d1 01  | j..Cj.. j..Cj..|
4000c0d0  20 6a a6 08 43 6a d1 01  00 a0 00 00 00 00 00 00  | j..Cj..........|
4000c0e0  00 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00  |........ .......|
4000c0f0  09 03 6b 00 69 00 70 00  6f 00 32 00 2e 00 6a 00  |..k.i.p.o.2...j.|
4000c100  70 00 67 00 00 00 00 00  80 00 00 00 48 00 00 00  |p.g.........H...|
```

(b) Entry is zeroed partially. *MFT*-entry-header remains intact!

Fig. 5. Change in *MFT*. The entry corresponding to the file kipo1.png is zeroed. The corresponding *MFT* entry starts at 0x4000bc00 and is followed by the next entry at 0x4000c000. The deleted *MFT* entry remains intact.

```
000277f0  09 03 46 00 69 00 6a 00  69 00 33 00 2e 00 02 00  |..F.i.j.i.3.....|
00027800  50 00 47 00 4a 00 50 00  2e 00 00 00 00 00 01 00  |P.G.J.P.........|
00027810  68 00 54 00 00 00 00 00  24 00 00 00 00 00 01 00  |h.T.....$.......|
00027820  60 a7 a1 08 43 6a d1 01  00 4f 63 10 3f ff cf 01  |`...Cj...Oc.?...|
00027830  60 a7 a1 08 43 6a d1 01  60 a7 a1 08 43 6a d1 01  |`...Cj..`...Cj..|
00027840  00 80 00 00 00 00 00 00  cf 7f 00 00 00 00 00 00  |................|
00027850  20 00 00 00 00 00 00 00  09 03 6b 00 69 00 70 00  | .........k.i.p.|
00027860  6f 00 31 00 2e 00 6a 00  70 00 67 00 4a 00 50 00  |o.1...j.p.g.J.P.|
00027870  2f 00 00 00 00 00 01 00  68 00 54 00 00 00 00 00  |/.......h.T.....|
00027880  24 00 00 00 00 00 01 00  20 6a a6 08 43 6a d1 01  |$....... j..Cj..|
00027890  00 49 eb 38 3f ff cf 01  20 6a a6 08 43 6a d1 01  |.I.8?... j..Cj..|
000278a0  20 6a a6 08 43 6a d1 01  00 a0 00 00 00 00 00 00  | j..Cj..........|
000278b0  2f 92 00 00 00 00 00 00  20 00 00 00 00 00 00 00  |/....... .......|
000278c0  09 03 6b 00 69 00 70 00  6f 00 32 00 2e 00 6a 00  |..k.i.p.o.2...j.|
000278d0  70 00 67 00 4a 00 50 00  30 00 00 00 00 00 01 00  |p.g.J.P.0.......|
```

(a) Hexdump of original B-tree where kipo1.png resided

```
000277f0  09 03 46 00 69 00 6a 00  69 00 33 00 2e 00 02 00  |..F.i.j.i.3.....|
00027800  50 00 47 00 4a 00 50 00  2f 00 00 00 00 00 01 00  |P.G.J.P./.......|
00027810  68 00 54 00 00 00 00 00  24 00 00 00 00 00 01 00  |h.T.....$.......|
00027820  20 6a a6 08 43 6a d1 01  00 49 eb 38 3f ff cf 01  | j..Cj...I.8?...|
00027830  20 6a a6 08 43 6a d1 01  20 6a a6 08 43 6a d1 01  | j..Cj.. j..Cj..|
00027840  00 a0 00 00 00 00 00 00  2f 92 00 00 00 00 00 00  |......../.......|
00027850  20 00 00 00 00 00 00 00  09 03 6b 00 69 00 70 00  | .........k.i.p.|
00027860  6f 00 32 00 2e 00 6a 00  70 00 67 00 4a 00 50 00  |o.2...j.p.g.J.P.|
00027870  30 00 00 00 00 00 01 00  70 00 60 00 00 00 00 00  |0.......p.`.....|
00027880  24 00 00 00 00 00 01 00  40 8e ad 08 43 6a d1 01  |$.......@...Cj..|
00027890  00 a7 42 2a a7 0f cf 01  40 8e ad 08 43 6a d1 01  |..B*....@...Cj..|
000278a0  40 8e ad 08 43 6a d1 01  00 70 67 00 00 00 00 00  |@...Cj...pg.....|
000278b0  2f 6a 67 00 00 00 00 00  20 00 00 00 00 00 00 00  |/jg..... .......|
000278c0  0f 01 4e 00 65 00 75 00  73 00 65 00 65 00 6c 00  |..N.e.u.s.e.e.l.|
000278d0  61 00 6e 00 64 00 31 00  2e 00 4a 00 50 00 47 00  |a.n.d.1...J.P.G.|
```

(b) Modified B-tree.

Fig. 6. Change when B-tree is updated. The change is made where the dumps are highlighted. Entries after the deleted entry are moved to its location.

that all metainformation and system structure of the deleted files and folder were irrecoverable deleted from the disk, was made using FTK for a post-mortem carving analysis.

Conclusion and future work

A practical approach to selectively delete data objects in a forensically sound manner was presented. In addition to existing wiping tools, this tool is able to fulfill legal requirements in regard to non-relevant data for an investigative case the way it was requested by Castiglione et al. (2011) and other mentioned privacy laws. Not only the data content itself is securely erased, but also all metadata, which may contain information about the user or the files, is irrecoverably deleted. The gap between the legal request for a selective deletion method and official statements, that no forensically sound tools might exist, can be closed with the proposed approach. The provenance of authenticity and integrity of the original device respectively the remaining files in the image, is kept. By continuous logging of every single step in the deletion process, a perfect proven documentation is guaranteed. While the presented implementation focuses on *NTFS*-structured devices or their corresponding images, support for other file-systems like (ex)FAT, Ext or HFS is imaginable and technically realizable in the future. To face critical appraisals of selective deletion it is clearly stated that the approach presented here provides a forensically sound deletion process for data objects. The correct handling depends on the user, which means: even if a selective deletion method exists, there is no guarantee to catch all non-relevant data because the selection is still made by a human.

Acknowledgments

We'd like to thank Felix Freiling and Tobias Zoubek for reviewing the final draft of this paper. Furthermore we thank Peter Trommler for his support and precious feedback. Additionally we wish to thank the reviewers for their helpful evaluation.

References

AccessData, Forensic toolkit (2016).
AccessData, Ftk Imager (2016).
Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley.
Castiglione, A., Cattaneo, G., De Maio, G., De Santis, A., 2011. Automatic, selective and secure deletion of digital evidence. In: BWCCA 11 Proceedings of the 2011 International Conference on Broadband and Wireless Computing, Communication and Applications, pp. 392–398.
D. B. L. für den Datenschutz, Prüfbericht quellen-tkü, Tech. rep. (2012).
Garfinkel, S., 2007. Anti-forensics: techniques, detection and countermeasures. In: 2nd International Conference on i-Warfare and Security, p. 77.
Lg frankfurt, beschl. v. 20.7.2004 – 5/30 artw 3/03, 5-30 artw 3/03, stv 2005, 79, 80 (2004).
Moser-Knierim, A., 2014. Der schutz der freiheit vor neuen herausforderungen. In: Vorratsdatenspeicherung. Springer, pp. 207–213.
nexsoft.de, Hex-editor mx (2016).
N. I. of Standards, T. (NIST), 2014. Nist Special Publication 800-88: Guidelines for Media Sanitization. Tech. Rep. Revision 1.
Pan, Y., Stackpole, B., Troell, L., 2010. Computer forensics technologies for personally identifiable information detection and audits. ISACA 2.
R. C. Merkle, Method of providing digital signatures, uS Patent 4,309,569 (Jan. 5 1982).
Stüttgen, J., Dewald, A., Freiling, F.C., 2013. Selective imaging revisited. In: IT Security Incident Management and IT Forensics (IMF), 2013 Seventh International Conference on, IEEE, pp. 45–58.
X.-W. AG, X-ways forensics (2016).