Digital Investigation 29 (2019) S51–S60

Contents lists available at ScienceDirect

Digital Investigation

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

DFRWS 2019 USA — Proceedings of the Nineteenth Annual DFRWS USA

Using NTFS Cluster Allocation Behavior to Find the Location of User Data

Martin Karresand [a, b, *], Stefan Axelsson [a, c], Geir Olav Dyrkolbotn [a]

[a] Department of Information Security and Communication Technology (IIK), Norwegian University of Science and Technology (NTNU), Norway
[b] Division of Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), Swedish Defence Research Agency (FOI), Sweden
[c] Halmstad University, Sweden

ARTICLE INFO

Article history:

Keywords:
Digital forensics
File carving
Partition content map
Allocation algorithm
NTFS

ABSTRACT

Digital forensics is heavily affected by the large and increasing amount of data to be processed. To solve the problem there is ongoing research to find more efficient carving algorithms, use parallel processing in the cloud, and reduce the amount of data by filtering uninteresting files.

Our approach builds on the principle of searching where it is more probable to find what you are looking for. We therefore have empirically studied the behavior of the cluster allocation algorithm(s) in the New Technology File System (NTFS) to see where new data is actually placed on disk. The experiment consisted of randomly writing, increasing, reducing and deleting files in 32 newly installed Windows 7, 8, 8.1 and 10 virtual computers using VirtualBox. The result show that data are (as expected) more frequently allocated closer to the middle of the disk. Hence that area should be getting higher attention during a digital forensic investigation of a NTFS formatted hard disk.

Knowledge of the probable position of user data can be used by a forensic investigator to prioritize relevant areas in storage media, without the need for a working file system. It can also be used to increase the efficiency of hash-based carving by dynamically changing the sampling frequency. Our findings also contributes to the digital forensics processes in general, which can now be focused on the interesting regions on storage devices, increasing the probability of getting relevant results faster.

1. Introduction

The amount of data to be handled during digital forensic case work is rapidly increasing and is a major challenge. The problem has been of concern to the digital forensic field for many years (Gladyshev and James, 2017; European Police Office (Europol), 2016; Quick and Choo, 2014a; Breitinger et al., 2013; Roussev, 2012), but the problem has not yet been solved. We therefore propose to use the principle of searching where it is more probable to find what you are looking for, instead of regarding every new storage media as a black box filled with randomly distributed data.

The principle is especially valid for the digital forensic sub-fields of *file carving* and *hash-based carving*, which are performed when there is no file system available. Instead the processes are based on using only the properties of the stored data itself (Poisel and Tjoa, 2013; Pal and Memon, 2009). That principle connects this article to our previous work on determining the data type (file type) of fragmented data by using histograms of the frequency of bytes, byte pairs and the difference between consecutive byte values (Karresand, 2008; Karresand and Shahmehri, 2008, 2007, 2006a,b,c). However, this time we determine the most probable position of user data, not the exact type of it.

When performing hash-based carving, hashes of blocks of a suspects hard drive are compared to hashes of blocks of known suspicious material. Since it is unfeasible to compare all hashes of a hard drive with all suspicious material hashes different strategies, techniques and algorithms have been developed (Garfinkel and McCarrin, 2015; Young et al., 2012; Foster, 2012; Garfinkel et al., 2010; Collange et al., 2009).

Different forms of file carving is highly valuable to the digital forensic investigator, but is CPU and I/O intense, hence much effort is put into mitigating the increasing amounts of data by different means. In a survey by Quick and Choo (2014b) the following concepts are listed; data mining, data reduction and subsets, triage, intelligence analysis and digital intelligence, distributed and

* Corresponding author. Department of Information Security and Communication Technology (IIK), Norwegian University of Science and Technology (NTNU), Norway.
E-mail address: martin.karresand@ntnu.no (M. Karresand).

https://doi.org/10.1016/j.diin.2019.04.018
1742-2876/

parallel processing, visualization, digital forensics as a service (DFaaS) and different artificial intelligence techniques.

The main focus of a digital forensic investigation are user activity and commonly the who, what, when, where, why and how (5WH) questions are meant to be answered. The activity comprises anything that has bearing on the user and his or her usage of the computer, i. e. system logs and files created by the user. Often such data are unique, because the probability of two users or processes independently creating exactly the same data is very low. Also shared data (not unique to a specific user) are of course of interest to the digital forensic investigator. The Windows operating system (OS) cluster allocation algorithms together with NTFS cannot differentiate between unique data and shared data, hence the data types will be stored together.

We define *user data* as any data that is created from the user's (daily) activity, regardless of its uniqueness. Data that is created during the installation and usage preparation (configuration) phases, before the user starts using the computer, we call *static data*.

By differentiating between user data and static data and combine that with the cluster allocation pattern in a collection of NTFS partitions we create what can be called a precomputed map of user data, showing the probability of finding user data at different Logical Block Addressing (LBA) position in generic NTFS formatted storage media. The map can then be used in the same way as a geographical map is used for planning, executing and following up activities in the physical world. The precomputed map is therefore meant to be reusable between investigations.

The mapping process is the same regardless of the type of storage media (solid-state drive (SSD) or mechanical drive), because we collect the data at the logical (LBA) level of the hard drive controller. In an SSDs the flash based physical storage is hidden by the Flash Translation Layer (FTL) (Afonin, 2019; Buckel, 2014; Reiter et al., 2014; Barbara, 2014; Chung et al., 2009), which also hides any wear leveling or other low level functions of the controller. If an SSD is accessed using Factory Access Mode (FAM) (Afonin, 2019) during an investigation the FTL is bypassed. However, the map can still be used, but in conjunction with a translation table to restore the logical layout of the disk. The translation table is stored on disk and accessible through the FAM.

To be able to empirically study the behavior of the cluster allocation algorithm used by Windows in NTFS we use virtual machines freshly installed with four different versions of Microsoft Windows (7, 8, 8.1 and 10). Each machine is powered on, a file operation (write, expand, shrink and delete files with a weighted random distribution) is performed on its internal (virtual) hard drive, which is then powered down and finally a copy of the $Bitmap file from the $MFT is extracted externally (via the host). This process is iterated 10000[1] times using 32 virtual machines in 8 nodes in a computer cluster. The $Bitmap file copies were then used to find the difference in cluster allocation status between each iteration by making a bitwise comparison between the files. Finally the usage frequency of each NTFS file system cluster is calculated and used to create a generic map of the allocation activity at different LBA positions in the partitions.

The work presented in this article complements a previous article (Karresand et al., 2019) where we studied the possibility to use real-world drives to create a map of the location of user data. The map was created using unique Secure Hash Algorithm 1 (SHA1) hashes of 512 B sectors. The unique hashes were assumed to represent data created by the user because of the low probability of several users creating data having exactly the same hash values, unless they shared the data.

The rest of this paper is organized as follows: The remaining parts of Section sec:introduction presents related work and our contributions. In Section sec:experiment we describe the experimental platform and how the experiment was implemented. Section sec:result presents the results of the experiment and in Section sec:discussion we discuss the effects and implications of our result to the research field of hash-based carving and also to other areas within and related to digital forensics. Section sec:concl-future-work concludes the work and presents ideas of future work to be done.

1.1. Background

Silberschatz et al. (2012) describe in detail how file systems are constructed. A file system is used to keep track of data stored on secondary storage. It can be organized in different ways, but all share some common properties; the addressing of the physical storage is abstracted by the file system into logical addresses and the position of the stored data is determined by an allocation algorithm. All modern file systems use index allocation, where the addresses of the file data blocks are held in an index separated from a file's data. This allocation strategy does not suffer from external fragmentation, but can waste disk space, especially for small files requiring a full index meta data block to hold just a few index posts.

There are also a number of algorithms used for handling the free space that is to be populated by new files. Silberschatz et al. (2012) list three of them, they are:

First fit where the first available free space large enough to hold the new file is used. The search for free space can either be from the current position or the start of the partition.

Best fit where the free space best fitting the new file is used, i. e. giving the smallest remaining free space. This requires all the free spaces available to be compared before the best can be chosen.

Worst fit where the free space having the worst fit to the new file is chosen. This is the opposite to best fit. The idea is to give the largest possible remaining free spaces, which then can be used to hold future files.

The first fit, best fit and worst fit free space allocation algorithms are not specific to storage of data on disk, they are also used in for example memory allocation in RAM (Silberschatz et al., 2012).

Based on the information given by Microsoft (2018) and Hughes (2009) Windows in combination with NTFS is using a index allocation strategy. The problem of space being wasted when using index allocation is in NTFS solved by storing the data of smaller files (up to approximately 700 B[2]) in the meta data records themselves. Microsoft (2018) states that the meta data in NTFS is held in the Master File Table (MFT), which in turn hold the MFT records associated with the files in the file system. According to Carrier (2005) the best fit algorithm is used by Windows XP on NTFS formatted hard disks. Since the book was written in 2005 it does not cover the allocation algorithms used by Windows 7 and newer. There are indications of the actual behavior of the allocation algorithm in a Superuser Q&A, where groups of free clusters are said to be allocated in descending order of size and ascending order of LBA (Superuser, 2017).

When formatting an NTFS partition 12.5% of the space is reserved for the MFT as default (Microsoft, 2018). The MFT records

[1] We had to break the experiment prematurely after 16 days for a number of machines due to time constraints. These machines had then performed at least 9035 iterations.

[2] The maximum size of an internal $Data attribute varies depending on the size of other attributes stored in the MFT record. Most sources give a maximum internal $Data attribute size of 600–700 bytes. Microsoft reports a 900 byte limit Microsoft (2018).

are 1 KiB in size and usually the size of the smallest allocatable unit (called cluster) in NTFS is 4 KiB. The allocation status of every cluster in the file system is stored in the $Bitmap file, which is record number 6 in the MFT. Each bit in the $Bitmap file represents one cluster in ascending LBA order. If a cluster is allocated the corresponding bit in the $Bitmap file is set to 1, hence 0 represents an unallocated cluster.

1.2. Related work

We have not found any related work directly dealing with the idea of precomputed maps of user data location. Instead we list related work from a number of digital forensic sub-fields that have bearing on our work.

1.3. Hash-based carving

The digital forensics research field of hash-based carving compares hashes of known file blocks to hashes of equally sized blocks from a suspects hard drive. In that way even files that are partially overwritten or damaged can be identified. The roots of the research field can be traced back to the spamsum tool by Tridgell (2002). According to Garfinkel one of the first times hashes are used for file carving is during the Digital Forensic ResearchWorkshop (DFRWS) 2006 Carving Challenge (Garfinkel and McCarrin, 2015). Later the spamsum tool is used as a basis for an article by Kornblum (2006) on piecewise hashing and what is now known as *approximate matching*. The concept of using hashes for file carving is further studied by Dandass et al. (2008) in 2008 in an article presenting an empirical analysis of disk sector hashes. The term hash-based carving is first introduced by Collange et al. (2009) exploring the possibility of using a Graphics Processing Unit (GPU) for comparing hashes of 512 byte sections of known files with hashes of equally sized sectors from disk images.

When Garfinkel and McCarrin use hashes for file carving in the DFRWS 2006 Carving Challenge (Garfinkel and McCarrin, 2015) they use hashes of parts of files found on the internet to find traces of the same files in the challenge image. These experiences lead to the development of the frag_find tool (Garfinkel et al., 2010). In connection with the frag_find article the authors discuss the optimal size of the data blocks to hash. They conclude that the size should be equal to the sector size, without stating if they mean 512 B or 4 KiB sectors. Garfinkel and McCarrin (2015) elaborate further on the size of hashed blocks and state that starting with Windows NT 4.0 the default minimum allocation unit in NTFS is 4 KiB (Microsoft, 2015).

Foster (2012) discusses the problem of data shared across files, stating that “the block of NULs is the most common block in our corpus” (Foster, 2012, p. 15), relating them to the NULL padding of files. The problem of the large amount of data to handle is also discussed. Young et al. (2012) continues the work further developing Foster's ideas. The authors discuss the optimal block size, how to handle a large amount of data, efficient hash algorithms, good data sets to use and common blocks of files.

Random sampling is used to improve the speed of hash-based carving in several articles (Garfinkel and McCarrin, 2015; Foster, 2012; Garfinkel et al., 2010). To find a suitable sampling frequency the problem is regarded as sampling without replacement. Using a higher sampling frequency may increase the detection rate, but has a negative impact on the execution speed. The problem is to find a suitable balance between the two alternatives.

1.4. Data persistence

The concept of data persistence is relevant to our work because the persistence at different areas of storage media indicates that they are not reused. This information is valuable when creating a precomputed map of a generic storage media.

Jones et al. (2016) have created a framework to enable studies of (deleted) file persistence in storage media. They use differential forensic analysis to compare snapshots of file systems in use and follow the decay of deleted files over time.

Fairbanks and Garfinkel (2012) present 12 factors affecting data persistence in storage media. Fairbanks (2015, 2012) also describes the low-level functions of ext4 and their effect on digital forensics.

1.5. Data reduction

Quick and Choo (2016, 2014a) propose methods to reduce the amount of data needed to be analyzed in digital forensic investigations. Their approach builds on extracting specific files using a list of key files and then working on the subset of files. This requires a working file system, limiting the methods applicability. Also the list of key files needs to be constantly updated.

Rowe (2014) has a similar approach as Quick and Choo, although more technical. He compares nine methods for identifying uninteresting files, defined as “those files whose contents do not provide forensically useful information about users of a drive.” (Rowe, 2014, p. 86). The methods studied by Rowe all require a working file system, which is not consistent with the foundation of file carving.

1.6. Data mapping

Key (2012) presents an EnScript module to the EnCase software which creates a map of the recoverable sectors of a file found in a file system. The module can handle situations where other tools does not work, for example when recovering partially damaged files. It is very processor intensive and therefore can only create maps of a few files at a time.

Gladyshev and James (2017) study the problem of file carving from a decision-theoretic point of view. They suggest a model where storage media is sampled with a frequency based on different properties of the hard disk and the file type that is to be found. In some specific situations their carving model outperforms standard linear carving algorithms, but their solution is not yet generally applicable. Gladyshev and James (2017) mention using the distribution of data on disk, but do not seem to relate that to the probability of finding user data at different LBA positions in storage media.

In two articles by van Baar et al. (2014) and van Beek et al. (2015) outlining the DFaaS system Hansken (van Beek et al., 2015) and its predecessor Xiraf (van Baar et al., 2014) the concept of non-linear extraction of data from images is discussed. Both van Baar and van Beek suggest that the MFT records (the file system meta data) of an NTFS partition are extracted first. The MFT records are then used to find other interesting areas of the file system. van Baar and van Beek also suggest that the analysis process is used to influence the imaging process by having specified parts being prioritized.

2. Experiment

Our experiment is based on iterating over the same process a predefined number of times. The process contains the following steps:

1. Boot a virtual machine.
2. Randomly (with bias) either create, delete, expand or shrink a file within the virtual machine's NTFS file system.
3. Shut down the machine.

4. Extract the $Bitmap file from the virtual hard disk (using dd from the host)

The experiment uses freshly installed virtual machines (VirtualBox) running Windows versions 7, 8, 8.1 and 10. The experiment empirically studies the Windows cluster allocation algorithm and its allocation frequency at different LBA positions.

Each virtual machine is installed with its specific Windows version using standard parameters. Then Python 2.7 is added together with the file operation scripts and an auto started `.bat` script, which is small enough to fit into an MFT record and hence does not require any new cluster allocation outside the MFT. The path environmental parameter is then modified to reflect the Python installation. Finally the security level is lowered to allow login without password. The goal is to keep the NTFS file system as pristine as possible to allow us to study the allocation algorithm from the start of the life of the file system. There are however a number of system processes, which is beyond our control, that also modifies the file system in each iteration.

Even though we do not have full control of the cluster allocation and deallocation during an iteration in the experiment we let 16 virtual machines use exactly the same file operation pattern to test if there is any deterministic behavior connected to the allocation, i. e. any similarity of the allocation patterns can be found. Hypothetically it should be, since the virtual machines within each Windows version are exact copies of each other. We do not use the clone function of VirtualBox when distributing the virtual machines to the computer cluster nodes, we use `scp` to copy them to keep them identical. We also verify the copies using SHA1 hash summing, to verify that they are identical.

Unfortunately one virtual machine had to be rebooted when the experiment was started and therefore was disqualified from the similarity test. Due to unforeseen behavior of the virtual machines (machine hung during boot, system messages locking the shut down process etcetera) a total of 5 virtual machines were disqualified from the similarity test at the end of the experiment, giving 11 that were possible to compare.

To be able to generalize the results we also performed a small experiment where we used virtual machines with larger hard drives (256 GiB). Windows 8 was excluded from that experiment due to its slow power cycle. Each machine was setup in the same way as the small hard disk machines and the same file operation pattern as in the similarity test with 16 machines was used.

2.1. Platform

The experiment is run on eight nodes in a large computer cluster. Each node is running Gentoo Linux 10.1 with kernel 4.18.13 and VirtualBox 5.2.20. The nodes are equipped with Intel Xeon E3-1230 v2 3.3 GHz CPUs, 500 GB Samsung 860 EVO SSDs and 32 GiB of RAM. The cluster is managed by another organization and we are not allowed to make any changes to the host and its OS. We therefore cannot install any specialized software, such as the Sleuth Kit by Carrier (2014) on the nodes.

We run four virtual machines in each node, one for each version of Windows in our test (see Table 1). The virtual machines are copied between the nodes and hence identical. The $Bitmap file from the MFT is used to check which clusters are affected by each file operation. Since we shut down the virtual machine as the second last step in every iteration any allocation changes are flushed (written) to the $Bitmap file. Thus the only difference between two consecutive $Bitmap file copies are the allocation changes induced by the latest file operation and any active system processes. The changes can contain both deallocation and allocation at the same time when a file is expanded and therefore moved to a new location. Likewise any changes to the MFT files, for example expansion of the MFT itself, will be manifested in the $Bitmap copies.

Table 1
The four versions of Windows used in our experiment.

Name	Version
Windows 7 Professional SP 1	7601
Windows 8 Enterprise	9200
Windows 8.1 Enterprise	9600
Windows 10 Consumer	1803

Table 2
The four file operations and their numerical representation used in the Python scripts.

Number	File operation
0	create
1	delete
2	increase
3	decrease

To enable us to extract the $Bitmap file after each process iteration the virtual machines are configured to use fixed size virtual disks. This type of disks are given their full size directly when created, which makes them behave as real hard disks, i. e. they are not affected by the virtualization layer of VirtualBox (TerryE and mpack, 2018). The virtual disk files therefore can be handled by standard Linux file carving tools, such as dd.[3]

We have limited the size of the fixed virtual disks to 64 GiB to be able to have four virtual machines in each node and still have space for the $Bitmap file copies, since each $Bitmap copy is 2 MiB large. The size is small compared to the current standard hard disks, but still large enough to be found in cheaper or older computers equipped with SSD hard disks.

Each virtual machine has Python 2.7 installed together with four Python scripts, one for each file operation. There is also an auto started `.bat` script used to send a signal when the virtual machine is completely started. The Python scripts are placed in a directory shared with the host and does not affect the allocation pattern of the virtual disk. For the communication between the host script and the virtual machine scripts we use the VBoxManage interface. Each virtual machine has its own virtual disk shared with the host and hence its own copies of the scripts. This configuration is used to isolate the machines from each other to minimize the risk of unspecified behavior due to several machines reading the same file. The file operations are executed as the local user of the virtual machine to simulate the activity of a real user.

2.2. Implementation

The virtual machines are each controlled by a Python script on the host node. The script is governed by a file containing randomly selected operations (see Table 2).

The selection of file operations is biased (weighted) and the size of the files varied within a size range. This is done by using a Python list (vector) containing different amounts of the numbers 0 to 3 based on the chosen bias. Each number in Table 2 represents a file operation and the amount of a specific number relative to the total

[3] There is a VirtualBox specific header at the beginning of the.vdi file containing the virtual hard disk. This header has to be skipped to reach the actual hard disk part. The header size is measured in one MiB blocks, usually it is two blocks large (TerryE and mpack, 2018).

amount of operations gives its bias. The bias value of an operation is calculated as $bias_{op} = \frac{factor_{op}}{\sum factors}$. The selection of a file operation is done by randomly choosing a value from the file operation vector. The vector [0, 0, 0, 0, 1, 1, 2, 3] will for example give 50% create operations, 25% delete operations and 12.5% increase and decrease operations respectively. The bias of each file operation in the experiment can be seen in Table 3.

There are also limits on the usage of the storage area to simulate a user that fills a hard disk with files over time and then erases a certain amount when the hard disk is believed to be full. The limits are based on our estimation of the behavior of a typical user. The *write start/stop* and *delete start/stop* limits in Table 3 is used to protect the virtual disk from being emptied or completely filled. If the current amount of data (controlled by the main script) in the virtual machine falls outside of the start limit multiplied with the *total size* (see Table 3) it triggers write or delete operations until the stop limit multiplied with the total size is reached. The degree of utilization of the partition for the simulated user behavior of the 16 machine similarity test can be seen in Fig. 1.

We have included the possibility to simulate the behavior of a user with regard to the size of the files operated on. Our assumption is that a user who use the computer for web surfing will create mostly small files (cached data and logs), a file sharing user will create a high amount of large files and a user storing a large amount of images will probably create mostly small to medium sized files. We therefore use size factors which define a file size class, which is measured in 512 B sectors. This function is implemented using a Python list in the same way as the file operation bias. The list is shown as the *size factors* in Table 3. The chosen factor is multiplied with a random number from the *random range* giving the number of sector to be affected (written, increased or decreased).

The script on the host checks if a virtual machine is started before it sends the file operation commands. There is also a check of the exit status of the virtual machine scripts that only logs successful executions. If the exit status indicates an error the iteration counter is decremented and the file operation is repeated. This behavior might induce extra allocations changes due to the extra power cycling of the virtual machine, but we accept them because occasionally a real user might also be forced to reboot a computer.

Every file operation is logged in a file external to the virtual disk. The log contains the sequence number, the action performed, the name of the affected file, the size factors, the current random size number and the current file size. The log file is stored in a VirtualBox share and hence does not interfere with allocation algorithm of the studied file system. We have chosen not to specifically store the file size difference for the increase and decrease operations, because they can be calculated from the stored transactions if needed.

Table 3
The settings used to generate the lists governing the behavior of the file operations on the virtual machines and other relevant settings used in the experiment. The settings are given as sectors of 512 B where applicable. The bias of the write/delete/increase/decrease operations are calculated as $bias_{op} = \frac{factor_{op}}{\sum factors}$.

Setting	Ident. behavior	Uniq. behavior
Size factors	8/2048	8/8/128/2048
Writes	10	10
Deletes	9	9
Increases	11	14
Decreases	10	7
Random range	1024	1024
Write start/stop	0.05/0.3	0.05/0.3
Delete start/stop	0.95/0.7	0.95/0.7
Total size	112000000	112000000

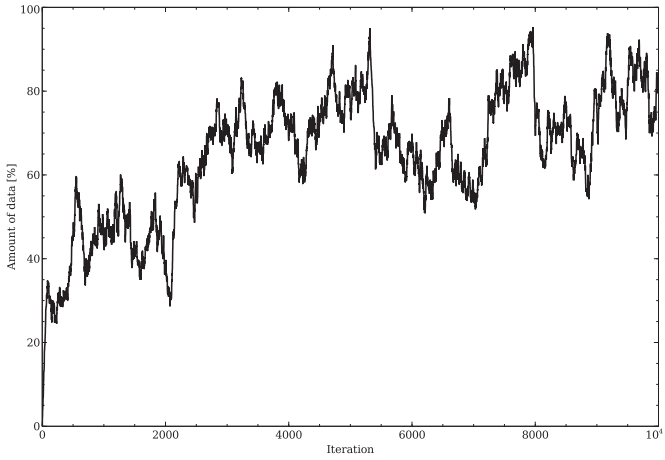

Fig. 1. The simulated user behavior, i. e. the total amount of data stored in the NTFS partition after each iteration, of the 16 virtual machines that were used for the similarity test. The plot shows 10000 iterations. The Windows 8 machines were run at least 9035 iterations before being shut down, the rest of the virtual machines executed all 10000 iterations.

The three Python scripts that execute write operations on the virtual machines are set to write the iteration sequence number into every 512 byte sector of the file. This enables us to see the raw write pattern in the virtual disk file if needed. The three scripts are also given the iteration sequence number as the file name for each file to further increase the traceability. The create and decrease file scripts both write new files (use the `wb` flag in the Python `open` command). This behavior might in the case of a file size decrease lead to the deallocation of the original clusters and the allocation of a smaller amount of new clusters, or even deallocation and allocation of the same clusters depending on the type of allocation algorithm used. The increase script appends new data at the end to an existing file, using the ab flag. Therefore the data written to disk of increased files can contain two or more sequence numbers.

Since the write operations are looped the size factor number of times the OS does not know the final size of the file and therefore can only optimize the allocation strategy for each write. This might induce a more stochastic behavior of the allocation algorithm and possibly hide any deterministic behavioral pattern from us, but that would lead to an underestimation of the experimental results, which is better than an overestimation.

To be able to detect the changes to the allocation status of the NTFS clusters of the virtual hard drive we have chosen to use the NTFS $Bitmap file. The file is extracted after each file operation as the last step in each iteration. Since the $Bitmap give the allocation status of 4 KiB NTFS clusters as the smallest unit we do not see any changes made at the logical 512 B sector level. Instead of using the $Bitmap file we can extract and compare the full virtual hard disk for each file operation to be able to detect any differences at the 512 B level. That would however require us to extract, compare and store 2^{15} times more data (64 GiB instead of 2 MiB) for each iteration. We therefore use 4 KiB blocks as the smallest units for the file operations.

The $Bitmap file copies from each iteration are extracted using the Linux `dd` tool. Using of the `dd` tool requires the position of the $Bitmap file to be known and static. The size of the $Bitmap file should not change since we keep the disk and partition sizes constant and hence it should not have to be extended or moved by NTFS and its position is consequently static. To find the position of the $Bitmap file before the experiment is started we use the `istat`

tool from The Sleuth Kit by Carrier (2014) on the virtual disk images, before they are copied to the hosts. The istat tool is run on an external computer (not a cluster node) because we are not allowed to install new software in the cluster nodes and hence cannot for example use the `icat` tool from the Sleuth Kit (Carrier, 2014) or the `idifference` tool (Garfinkel and Fairbanks, 2012) during the experiment. Using the idifference tool would also force us to perform post-processing since the tool only reports differences at the file level, i. e. we would have to use the istat (Carrier, 2014) to find what clusters each file allocates and then do a difference calculation. Consequently the `idifference` tool is of no use to us.

The virtual hard disks in our experiment are not encrypted because neither *block device encryption* (for example BitLocker) nor *stacked file system encryption* (for example EFS) affect the allocation strategy of the OS or file system, since they act either above or below the file system (Arch Linux, 2018; Gattol, 2015). Adding encryption to the virtual hard disks would therefore put an execution overhead on the experiment without affecting the data allocation.

To be able to estimate the position of the bulk of the OS files in the partitions we extract all existing files containing the string "Windows" somewhere in their paths. We then filter out the files containing "Users/" to avoid contaminating the result with user data. The extraction is done after the experiment is finished to also include any system files written during the file operations. To further strengthen the result we also includ three real-life disks in the OS file extraction. These three disks are taken from home user computers and all have been used for at least a year.

2.3. Map creation

When the chosen number of iterations is reached we do a differential analysis of each consecutive pair of $Bitmap copies extracted as the last step in each iteration. This gives us the LBA position for each change in allocation status for each file operation. Since we cannot control the behavior of the OS any allocation changes induced by the OS are also included. The probability of a new file being larger than the system changes induced by its creation is high, because otherwise the file system would be very inefficient. Hence the allocation changes introduced by the OS at each iteration are negligible compared to the changes occurring because of the file operation.

We then plot the $Bitmap changes from the previous step. All deallocation posts are removed, because we only want to know the allocation frequency of each LBA position and each allocation must be preceded by a deallocation. The remaining posts are then merge sorted in order of LBA position.

To reduce any noise in the plot we group equally sized areas of the storage media together based on the position in the partition. The number of groups is chosen based on the desired precision of the map. Finally the mean of the allocation frequencies of the posts in each group are calculated. These groups then make up the map, which resolution depends on the number of groups.

3. Result

The $Bitmap files extracted during the experiment contain not only traces of the file operations executed by our scripts, but also any operations executed by the OS during each iteration. Especially the start and stop phases of an iteration will induce changes to the MFT of the file system. Hence the data will include clusters allocated by the OS too, but that is a minor problem because the OS activities often affects already allocated clusters. An MFT record is 1 KiB in size and the smallest allocatable unit in a 64 GiB NTFS partition is 4 KiB, hence every fourth file creation will give rise to a new cluster being allocated due to new MFT records being created. Of course there are also for example log files written by the OS, but when such a file grows and requires a new cluster to be allocated the cluster position will most probably be found in the user data allocation area. Consequently it will strengthen the result of the experiment.

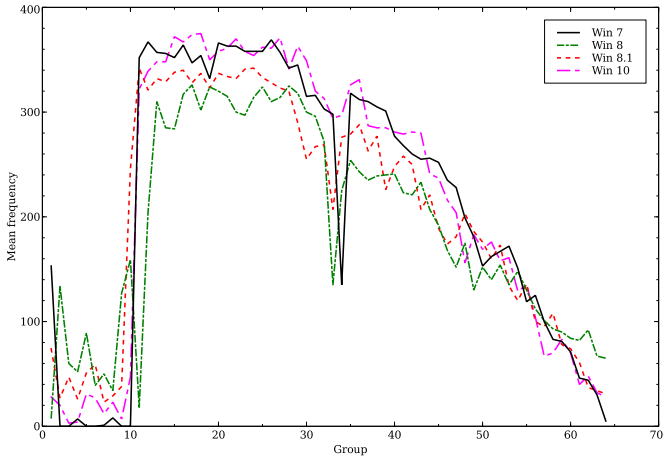

Fig. 2. A map of the mean allocation frequency for all 32 virtual machines having NTFS formatted partitions in 64 GiB disks running Windows 7, 8, 8.1 and 10. The resolution is set to 64 groups.

In Fig. 2 a plot of the allocation frequency at different LBA positions is shown. The plot is divided into 64 equally sized groups based on LBA position. We have separated the plots for each of the OSs in the experiment to enable a comparison of their behavior. As can be seen the behavior of the OSs differ in the first part of the partitions, as well as in the very last part. When formatting a hard disk as NTFS a contiguous area of 12.5% of the partition space is reserved for the MFT (Microsoft, 2018) as default. We have not found any specification of the exact LBA position of this area, but we have noticed that the MFT allocation starts at cluster 786432 in every non-related NTFS formatted partition we have checked in more than 35 computers running Windows 7 and above. Cluster 786432 corresponds to a position exactly 3 GiB into the partition, hence close to the beginning of the partition. The size of the reserved area can be changed by the user when formatting the partition or by the OS if the MFT requires more space. As can be seen in Fig. 2 Windows 7, 8.1 and 10 have a somewhat lower amount of allocation changes at the start of the partition than Windows 8. We can also see that the behavior of the older Windows 7 differ from the other three Windows versions in the very first part of the partitions.

After the area reserved for the MFT the OS, different software from the initial installation and the user data reside. The minimum requirement for free hard disk space for a Windows 7, 8, 8.1 and 10 installation is 20 GiB for 64-bit systems according to Microsoft (2017b, a, c). Hence the 12.5% together with the OS files correspond to approximately 45% of the space in our 64 GiB disks. The distribution and positions of the system files (OS and installed software) is uncertain, we have not been able to check the position of every system file in the partitions due to the large amount of work involved. What we know empirically is that the MFT starts its allocation at exactly 3 GiB into the partitions, which correspond to group 3 on the x-axis in Fig. 2.

Close to the middle of the partitions (see Fig. 2) there are large disturbances in the plots for all versions but Windows 10. According to Carrier (2005) the 4 KiB $MFTMirr file, containing a backup of the first four files in the MFT, is located at the middle of

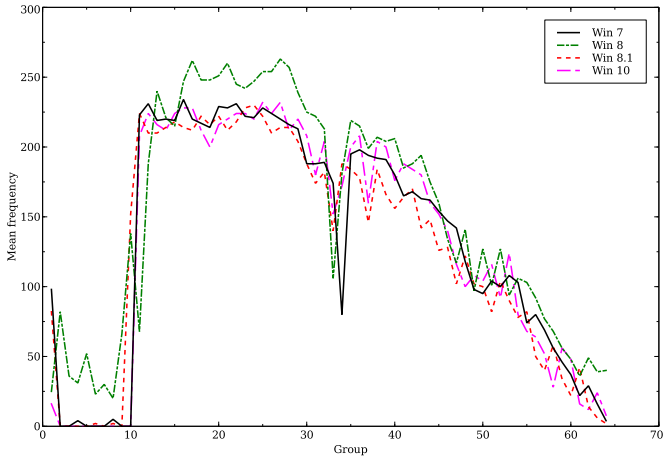

Fig. 3. A map of the mean allocation frequency using the same file operation pattern of the 16 NTFS formatted partitions in 64 GiB disks running Windows 7, 8, 8.1 and 10. The resolution is set to 64 groups.

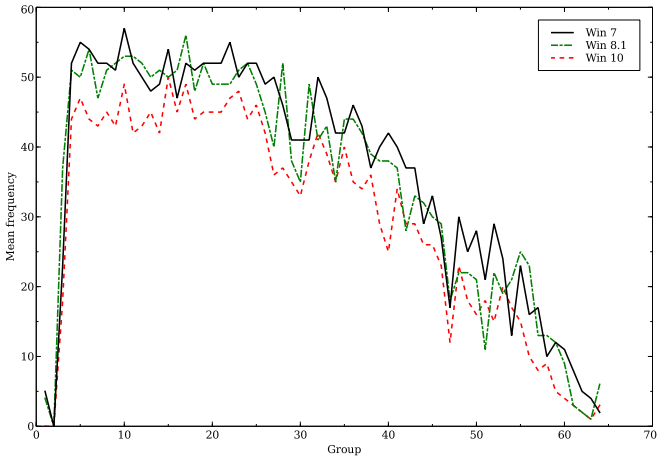

Fig. 4. A map of the mean allocation frequency in NTFS formatted partitions in 256 GiB hard disks running Windows 7, 8.1 and 10 and using the same file operation pattern. The resolution is set to 64 groups.

an NTFS partition. This might be true in Windows XP,[4] but not for the newer versions of Windows in our experiment. In these machines we have found the $MFTMirr file to be located at different LBA positions, none of them at the middle of the partition. Still Microsoft might have reserved the middle of the partition for other special files or functions and therefore the dip in allocation frequency at that position in Fig. 2.

The allocation frequency of the 16 virtual machines that were using the same file operation pattern can be seen in Fig. 3. Since two Windows 8.1 machines and tree Windows 10 machines broke down during the test we have normalized the result by multiplying the numbers for Windows 8.1 with 2 and with 4 for Windows 10. As can be seen the allocation behavior of the different versions of Windows are not similar, although they all performed exactly the same file operations. The deviation might originate from effects in the broken down machines. Still the differences are larger than what can be expected from losing the allocation of a few files. The allocation behaviour is not much different from the result of the full test using all 32 virtual machines. One apparent difference is the dip at the middle of the partition, which is deeper for Windows 10 in the sub-experiment than for the full experiment shown in Fig. 2. Another difference is the relative higher allocation frequency of Windows 8 in groups 15 to 35 compared to the other Windows versions in the sub-experiment (see Fig. 3) than in the full experiment. Hence we can conclude that although similar, the behavior of the allocation algorithm in different versions of Windows are deviating.

The result of the experiment with larger hard disks running the same file operations as 16 of the small hard disk machines is shown in Fig. 4. The experiment only included three different versions of Windows (7, 8.1 and 10) because of time and space constraints. Yet it is possible to see similarities with the results of the other experiments.

The plots in Figs. 2, 3 and 4 all have a common shape with only small variations between them, which indicates that the allocation algorithm is behaving the same way at least in the size span of 64 GiB to 256 GiB. The disturbance at the middle of the partitions is less visible in 256 GiB hard disks than in the 64 GiB versions due to their larger size in combination with the resolution being kept at 64 groups. These similarities between hard disks of different sizes indicates that the result can be generalized to partitions with different sizes, at least up to 256 GiB. The algorithm might still behave differently in partitions larger than 256 GiB, but this is left as future work due to the current limitations in the hardware available to us.

We also inspected the allocation pattern using istat from the Sleuth Kit by Carrier (2014) for some of the largest files present at the end of the experiment in two randomly chosen virtual machines running Windows 7 and 10. The result showed that the files were heavily fragmented and that the allocation algorithm had allocated free areas in order of increasing size. There were noise in the form of some small areas allocated in between the larger, but the trend was clearly visible. We therefore also tested to create eight new files in one of the virtual machines using a modified create script that wrote consecutive numbers into every 512 byte sector. This showed that the allocation pattern from the istat tool was given in consecutive order and that the size of the allocated areas increased towards the end of the partition also for new files.

The result of the experiment where we extract all files having “Windows” in their path indicates that the OS files written during installation are placed at the beginning of a partition, which can be seen in Fig. 5. The result is based on the position of these files in four virtual disks and three real-life disks. The real-life disks are larger than the virtual disks and we have no control of the type of user behavior for these hard disks. Still all hard disks show similar behavior in the first part of the partitions.

The three real-life hard disks in Fig. 5 have their almost vertical bend, which probably represents the end of the OS installation file area, at a higher LBA position than the smaller virtual disks (cluster number 6000000 instead of 2500000). The Windows 7 SP1 partition even has two vertical bends, the second bend probably originates from the Service Pack installation. The reason for the larger installation area is the need for a larger amount of drivers due to the more diverse hardware base in the real-life computers. There is also extra software installed in close connection to the OS installation in the real-life computers, which is not the case for the virtual machines, which were kept as simpel as possible. The more irregular shape of the real-life hard disk curves comes from a more frequent usage and longer life span than for the virtual machines. The two Windows 8.1 machines have not utilized as much of the partitions space for OS files as the rest of the Windows versions, which is manifested by flat and sparse lines in the plot.

[4] We have checked a number of partitions downloaded from the Real Data Corpus (Digital Corpora, 2018) indicating this.

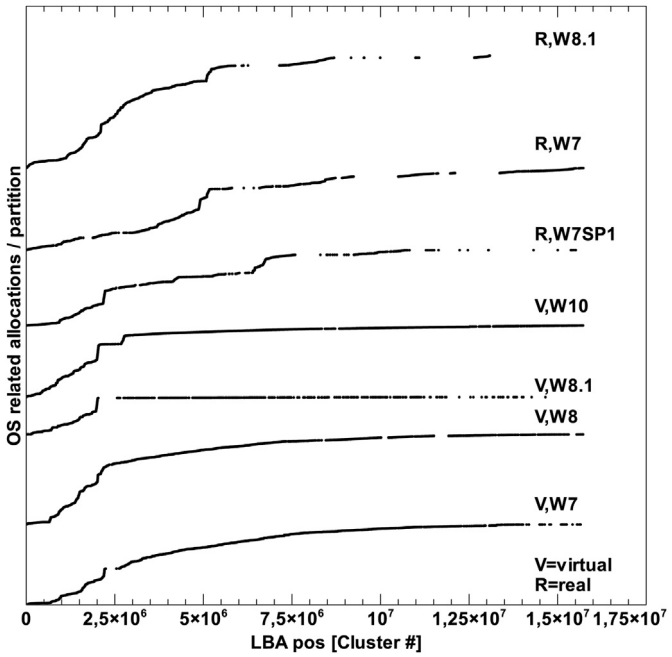

Fig. 5. The density of OS related files at different LBA positions in four virtual and three real-life hard disks. A steeper slope of a curve means a higher density of OS files and a lower slope represents a lower density. As can be seen the starting areas, up to approximately cluster $2.5 \cdot 10^6$, have a steeper slope than the rest of the curves. The curves represent, from bottom to top, four virtual 64 GiB hard disks containing Windows 7, 8, 8.1 and 10 and three real-life hard disks, 500 GiB, 256 GiB and 500 GiB in size, containing Windows 7 SP1, 7 and 8.1.

4. Discussion

Although the result of the experiment might seem limited in its simplicity and already be covered by common knowledge, it still is (as far as we know) the first attempt to empirically prove the common knowledge and make it a scientific fact. An alternative would have been to read the source code of the file system and OS allocation algorithm for each Windows version, but that would not have taken the installation process and any static data into account. By using the empirical approach we have managed to show the impact of the installation, the reserved MFT area and other attributes that might not have been found through a source code inspection.

Our approach of using the $Bitmap file to find the LBA positions allocated to new or modified files is not optimal, but we hade to use it because we were not allowed to install any new software on the computer cluster nodes we had access to and they only contained a basic collection of Linux tools. We would have preferred to use tools (for example *istat*) from the Sleuth Kit toolkit to extract the exact allocation information for each file, which was also suggested by one of the reviewers of the article. Using the istat tool would also have automatically filtered out the allocations of system files from our results. However, we still got results that are similar to our previous work on real-life data presented in an earlier article (Karresand et al., 2019).

The experiment showing the placement of OS files is based on only seven hard disks, including four that are virtual and specifically prepared for the experiment. The validity of the results might therefore be questionable. The behavior at the very beginning of the disks, where the files from the installation process should be placed according to our hypothesis, is valid for all seven disks. However, the result should be seen more as an indicator of approximately where on disk system files reside, not as the truth. The selection of the files to be included is based solely on the existence of the word “Windows” in the path of the files. There certainly are system files that do not fulfill that requirement and likewise there are a large amount of non-system files that have “Windows” in their path, for example user installed software. We will therefore expand the collection of real-life hard disks as much as possible and also create better search terms and filters to increase the amount of system files and exclude non-system files.

The test used to find any deterministic behavior between different versions of Windows did not prove the hypothesis of the allocation pattern being deterministic. There seem to be slight alterations to the allocation algorithms in different versions. However the test was partly flawed due to a number of virtual machines crashing repeatedly and therefore behaving differently. There might still be deviations even in the machines we used for the plot in Fig. 3, which we have not been able to detect. The test do prove that the probability of two different Windows machines having exactly the same allocation pattern is very low.

Our discovery that the allocation algorithm is allocating free areas in order of increasing size is interesting because that contradicts a strict best fit behavior. The behavior gives a high fragmentation of files, but preserves any large unused areas. Hence it seem to adhere to the idea of a worst fit algorithm that is meant to preserve as large areas as possible for future use.

In hash-based carving a collection of hashes of known data is compared to hashes of an unknown source leading to a huge amount of hash comparisons. Different techniques are proposed to mitigate the problem, one being random sampling with a frequency chosen to balance speed and detection rate. The frequency is uniformly distributed and we therefore propose an upgrade by varying the sampling frequency in accordance with the precomputed map of the probability of finding user data at different positions in an NTFS formatted partition. The concept can be compared to the common sense principle of looking for a lost item where the probability of finding it is higher.

Our mapping concept can be of benefit to other areas than hash-based carving too. On a general level it can be used to improve the efficiency of the current digital forensic methods and tools, especially in file carving situations where there is no file system to be used. One example of usage is when searching for hashes of files or parts of files in a hard disk. Then three different scenarios are possible:

Prioritizing speed Instead of using a uniformly distributed sampling our map can be used to lower the total amount of samples without any significant loss in detection ability. This method can for example be used in triage situations when there is a need to get a preliminary answer quickly.

Maintaining speed Using the same total amount of samples as in a uniformly distributed sampling case, a higher detection ability can be achieved at the same execution speed. This method can be used without changing the digital forensic process.

Prioritizing detection rate. By increasing the total amount of samples compared to a uniformly distributed sampling case, the detection rate will increase more than the induced penalty in execution speed. This will be achieved by using the same sampling frequency in areas of low interest as in the uniformly distributed sampling case and increasing the sampling rate for better detection ability in high priority areas of the hard drive.

Our mapping concept is also beneficial to the daily work of the digital forensic investigator by introducing the possibility to plan the forensic process in a better way. Currently hard drives and other storage media are treated as black boxes and scanned from start to end before the analysis. Using our map the forensic investigators can focus on relevant areas of the storage media and postpone, or even skip, less relevant areas.

The map is also applicable when imaging storage media. By starting the imaging process at the most probable area of user data and continue in decreasing order of relevance some of the analysis work can be started immediately rendering results faster. In that way valuable time and effort is saved, although the reliability of the analysis will of course increase as more data are analyzed. This concept is supported by the Hansken project (van Beek et al., 2015; van Baar et al., 2014) and using our mapping concept the speed of the analysis process in Hansken will be even higher, especially when dealing with partly damaged media.

Also when dealing with corrupt or damaged storage media our map can be beneficial. The forensic value of any unreadable areas of a storage media can quickly be found using the map. The information can then be used to establish the forensic value of the lost areas and consequently increase the value of the evidence in court.

Since we have based our map on diving a generic storage media into a reasonable amount of subareas (currently 64) there will not be any real performance penalty due to random seek. Within the areas any seek pattern can be used, it is only when switching between the areas (in order of priority) a random seek situation may occur. The performance penalty of doing a maximum of 64 random seeks can safely be ignored.

5. Conclusion and future work

We have empirically studied the behavior of the cluster allocation algorithms in Windows 7, 8, 8.1 and 10 to see whether it is possible to create a precomputed map of the probability of finding new data at different LBA positions in NTFS formatted partitions. The result show that the OSs in question share a structure at a high level regarding which areas (clusters) that are being allocated more frequently. The highest probability can be found approximately 10% into a (64 GiB) partition. The probability is then slowly decreasing down to half the maximum value and then drops rapidly towards zero closer to the end of the partitions in our experiment.

The concept of creating a precomputed map of a generic storage media is useable to a wide range of applications within digital forensics. The field of file carving is the most obvious benefactor, but the concept can also be used in for example disk imaging, planning the investigation process and data rescue situations when dealing with failing hardware.

As future work we will first of all modify our experimental framework to use the proper tools needed to exclude everything but the exact LBA positions allocated by the executed file operations. We will also include other combinations of OSs and file systems in our experiments to create precomputed maps for the most popular consumer computer systems. Since the framework we have developed for the experiments can handle any system (OS and file system) that can be installed in a VirtualBox machine we will cover as much as we can of the current OS and file system market.

Finally we will continue to search for the reason behind the disruption in the middle of the plots, as well as any other matter needed to reverse engineer the behavior of the allocation algorithms in Windows versions not included in the book by Carrier (2005).

Acknowledgment

We would like to thank the anonymous reviewers for their insightful comments and suggestions. We are also thankful to the Swedish Defence Research Agency (FOI), which let us use a subset of their Cyber Range And Training Environment (CRATE) computer cluster for the experiments.

The research leading to these results has received funding from the Research Council of Norway programme IKTPLUSS, under the research and development project Ars Forensica grant agreement 248094/O70.

References

Afonin, O., 2019. Life after Trim: Using Factory Access Mode for Imaging SSD Drives. https://blog.elcomsoft.com/2019/01/life-after-trim-using-factory-access-mode-for-imaging-ssd-drives/. (Accessed 15 March 2019).

Arch Linux, 2018. Disk Encryption. https://wiki.archlinux.org/index.php/disk_encryption. (Accessed 30 December 2018).

Barbara, J., 2014. Solid State Drives: Part 5. https://www.forensicmag.com/article/2014/04/solid-state-drives-part-5. (Accessed 8 October 2018).

Breitinger, F., Stivaktakis, G., Baier, H., 2013. Frash: a framework to test algorithms of similarity hashing. Digit. Invest. 10 (Suppl. ment), S50–S58 (the Proceedings of the Thirteenth Annual DFRWS Conference).

Buckel, C., 09 2014. Understanding Flash: the Flash Translation Layer. https://flashdba.com/2014/09/17/understanding-flash-the-flash-translation-layer/. (Accessed 8 October 2018).

Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley Professional.

Carrier, B., 2014. TSK Tool Overview. http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.

Chung, T.-S., Park, D.-J., Park, S., Lee, D.-H., Lee, S.-W., Song, H.-J., 2009. A survey of flash translation layer. J. Syst. Archit. 55 (5–6), 332–343.

Collange, S., Dandass, Y.S., Daumas, M., Defour, D., 2009. Using graphics processors for parallelizing hash-based data carving. In: 2009 42nd Hawaii International Conference on System Sciences, pp. 1–10.

Dandass, Y., Necaise, N., Thomas, S., 2008. An empirical analysis of disk sector hashes for data carving. J. Digit. Forensic Pract. 2 (2), 95–104.

European Police Office (Europol), 2016. Internet Organised Crime Threat Assessment (IOCTA) 2016. Tech. Rep. European Cybercrime Centre (EC3).

Fairbanks, K., 2012. An analysis of ext4 for digital forensics. Digit. Invest. 9 (Suppl), S118–S130 (the Proceedings of the Twelfth Annual DFRWS Conference).

Fairbanks, K., 2015. A technique for measuring data persistence using the ext4 file system journal. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 3, pp. 18–23.

Fairbanks, K., Garfinkel, S., 2012. Column: factors affecting data decay. Journal of Digital Forensics, Security and Law 7 (2).

Foster, K., 2012. Using Distinct Sectors in Media Sampling and Full Media Analysis to Detect Presence of Documents from a Corpus. Master's thesis. Naval Postgraduate School, Monterey, California, USA.

Garfinkel, S., Fairbanks, K., 2012. sleuthkit/tools/fiwalk/. https://github.com/kfairbanks/sleuthkit/tree/master/tools/fiwalk. (Accessed 30 December 2018).

Garfinkel, S., McCarrin, M., 2015. Hash-based carving: searching media for complete files and file fragments with sector hashing and hashdb. Digit. Invest. 14 (Suppl. 1), S95–S105 (the Proceedings of the Fifteenth Annual DFRWS Conference).

Garfinkel, S., Nelson, A., White, D., Roussev, V., 2010. Using purpose-built functions and block hashes to enable small block and sub-file forensics. Digit. Invest. 7 (Suppl. ment), S13–S23 (the Proceedings of the Tenth Annual DFRWS Conference).

Gattol, M., 2015. Block-layer Encryption. https://web.archive.org/web/20150917051251/http://www.markus-gattol.name/ws/dm-crypt_luks.html. (Accessed 24 January 2019).

Gladyshev, P., James, J., 2017. Decision-theoretic file carving. Digit. Invest. 22 (Suppl. C), 46–61.

Hughes, J., 2009. The Four Stages of NTFS File Growth. https://blogs.technet.microsoft.com/askcore/2009/10/16/the-four-stages-of-ntfs-file-growth/. (Accessed 24 October 2018).

Jones, J., Khan, T., Laskey, K., Nelson, A., Laamanen, M., White, D., 2016. Inferring previously uninstalled applications from residual partial artifacts. In: Annual ADFSL Conference on Digital Forensics, Security and Law, pp. 113–130.

Karresand, M., 2008. Completing the Picture — Fragments and Back Again. Licentiate thesis. Linköping Institute of Technology, Linköping University, Sweden.

Karresand, M., Warnqvist, Åsalena, Lindahl, D., Axelsson, S., Dyrkolbotn, G.O., 2019. Advances in Digital Forensics XIV. Springer International Publishing AG, Cham, Switzerland (Ch. 4. Creating a map of user data in NTFS to improve file carving).

Karresand, M., Shahmehri, N., 2006a. File type identification of data fragments by their binary structure. In: Proceedings from the Seventh Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2006. IEEE, Piscataway, NJ, USA, pp. 140–147.

Karresand, M., Shahmehri, N., 2006b. Oscar – file type and camera identification using the structure of binary data fragments. In: Haggerty, J., Merabti, M. (Eds.), Proceedings of the 1st Conference on Advances in Computer Security and Forensics, ACSF. The School of Computing and Mathematical Sciences. John Moores University, Liverpool, UK, pp. 11–20.

Karresand, M., Shahmehri, N., 2006c. Oscar – file type identification of binary data in disk clusters and RAM pages. In: Proceedings of IFIP International Information Security Conference: Security and Privacy in Dynamic Environments (SEC2006). Lecture Notes in Computer Science, pp. 413–424.

Karresand, M., Shahmehri, N., 2007. Oscar – using byte pairs to find file type and camera make of data fragments. In: Blyth, A., Sutherland, I. (Eds.), Proceedings of the 2nd European Conference on Computer Network Defence, in Conjunction with the First Workshop on Digital Forensics and Incident Analysis (EC2ND 2006). Springer Verlag, pp. 85–94.

Karresand, M., Shahmehri, N., 2008. Reassembly of fragmented jpeg images containing restart markers. In: Proceedings - 4th Annual European Conference on Computer Network Defense, EC2ND 2008, pp. 25–32.
Key, S., 2012. File Block Hash Map Analysis. https://www.guidancesoftware.com/app/File-Block-Hash-Map-Analysis. (Accessed 28 April 2018).
Kornblum, J., 2006. Identifying almost identical files using context triggered piecewise hashing. Digit. Invest. 3 (Suppl. ment), 91–97 the Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).
Microsoft, 2015. Default Cluster Size for NTFS, FAT, and exFAT. https://support.microsoft.com/en-us/help/140365/default-cluster-size-for-ntfs-fat-and-exfat.
Microsoft, 2017a. System Requirements. https://support.microsoft.com/en-gb/help/12660/windows-8-system-requirements. (Accessed 30 April 2018).
Microsoft, 2017b. Windows 10 System Requirements. https://support.microsoft.com/en-us/help/4028142/windows-windows-10-system-requirements. (Accessed 30 April 2018).
Microsoft, 2017c. Windows 7 system requirements. https://support.microsoft.com/en-us/help/10737/windows-7-system-requirements. (Accessed 30 April 2018).
Microsoft, 2018. How NTFS Works. https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx. (Accessed 30 September 2018).
Pal, A., Memon, N., 2009. The evolution of file carving. IEEE Signal Process. Mag. 26 (2), 59–71.
Poisel, R., Tjoa, S., 2013. A comprehensive literature review of file carving. In: 2013 International Conference on Availability, Reliability and Security, pp. 475–484.
Quick, D., Choo, K., 2014a. Data reduction and data mining framework for digital forensic evidence: storage, intelligence, review and archive. Trends & Issues in Crime and Criminal Justice 1–11.
Quick, D., Choo, K.-K.R., 2014b. Impacts of increasing volume of digital forensic data: a survey and future research challenges. Digit. Invest. 11 (4), 273–294.
Quick, D., Choo, K.-K.R., 2016. Big forensic data reduction: digital forensic images and electronic evidence. Clust. Comput. 19 (2), 723–740.
Digital Corpora, 2018. Real Data Corpus. https://digitalcorpora.org/corpora/disk-images/real-data-corpus. (Accessed 29 September 2018).
Reiter, R., Swatosh, T., Hempstead, P., Hicken, M., November 2014. Accessing Logical-To-Physical Address Translation Data for Solid State Disks. http://www.freepatentsonline.com/8898371.html. (Accessed 8 October 2018).
Roussev, V., 2012. Managing terabyte-scale investigations with similarity digests. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics VIII: 8th IFIP WG 11.9 International Conference on Digital Forensics, Pretoria, South Africa, January 3-5, 2012, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 19–34.
Rowe, N.C., 2014. Identifying forensically uninteresting files using a large corpus. In: Gladyshev, P., Marrington, A., Baggili, I. (Eds.), Digital Forensics and Cyber Crime: Fifth International Conference, ICDF2C 2013, Moscow, Russia, September 26-27, 2013, Revised Selected Papers. Springer International Publishing, Cham, pp. 86–101.
Silberschatz, A., Galvin, P., Gagne, G., 2012. Operating System Concepts, ninth ed. Wiley.
Superuser, 03, 2017. What Block Allocation Algorithm Does NTFS Use? https://superuser.com/questions/274855/what-block-allocation-algorithm-does-ntfs-use. (Accessed 24 January 2019).
TerryE and mpack, 2018. All about Vdis. https://forums.virtualbox.org/viewtopic.php?t=8046. (Accessed 30 December 2018).
Tridgell, A., 2002. Spamsum README. https://www.samba.org/ftp/unpacked/junkcode/spamsum/README. (Accessed 27 April 2018).
van Baar, R., van Beek, H., van Eijk, E., 2014. Digital forensics as a service: a game changer. Digit. Invest. 11, S54–S62 (proceedings of the First Annual DFRWS Europe).
van Beek, H., van Eijk, E., van Baar, R., Ugen, M., Bodde, J., Siemelink, A., 2015. Digital forensics as a service: game on. Digit. Invest. 15, 20–38 (special Issue: Big Data and Intelligent Data Analysis).
Young, J., Foster, K., Garfinkel, S., Fairbanks, K., 2012. Distinct sector hashes for target file detection. Computer 45 (12), 28–35.

Martin Karresand is a PhD candidate at the Center for Cyber and Information Security, Department of Information Security and Communication Technology (IIK), Norwegian University of Science and Technology (NTNU), Norway. He is also a Senior Scientist at the Department of Information Security and IT Architecture, Division of Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), Swedish Defence Research Agency (FOI), Sweden, since 2003. He got his MSc in Engineering in 2002 and a Licentiate degree in Computer Science in 2008, both from Linköping University (LiU), Sweden. The Licentiate thesis, titled "Completing the picture – Fragments and back again", lead to a leave from FOI and an employment as a digital forensic engineer at the Swedish Laboratory for Forensic Science (SKL) between 2010 and 2013. He is a lecturer at several digital forensic courses at LiU and FOI and act as an external expert during examinations of digital forensic investigators at the Swedish National Forensic Centre (NFC). His research interests include file carving, machine learning, data mining and cyber defence.

Stefan Axelsson is an associate professor at NTNU, Norway, with a PhD in computer security from Chalmers University of Technology, Sweden, where he also got his master's degree in engineering. He has been working with Norwegian police via NTNU, and is currently an associate professor at Halmstad University, Sweden. He also has considerable experience from the industry, mainly from Ericsson, where he worked as a specialist on security of mobile IP telecommunications systems. He has authored, and co-authored, a number of research publications that together have been cited more than 2000 times. His research interests include surveillance of digital systems and data analysis supported by automatic methods (his PhD was on the topic of machine learning and information visualization for intrusion detection).

Geir Olav Dyrkolbotn is an officer in the Norwegian Armed Forces and an associate professor at Center for Cyber and Information Security (CCIS) at the Norwegian University of Science and Technology (NTNU). He is currently head of the NTNU Malware Lab and the research group for cyber defence at CCIS. Geir Olav holds a PhD in information security from Gjøvik University College (HiG) and a MSc in computer science from the NTNU. His PhD thesis was on reverse engineering microprocessor content using electromagnetic radiation. His career includes more than 25 years in the Norwegian Armed Forces, where he holds the rank of Major. His career has focused on operation, maintenance and security in tactical communication systems and the last 15 years on defensive cyber operations, computer network defense and operational security. His research interest include cyber defense, reverse engineering and malware analysis, side-channel attacks and machine learning. His current focus is on Malware Analysis, Cyber Operations, Cyber Intelligence, Digital forensics and machine learning. He is also the founder and chair of NTNU Malware Forum and a member of the project group for the Norwegian Cyber Range, working to establish a virtualized arena for education, training and exercises for the Norwegian society.