

Doctoral thesis

Doctoral theses at NTNU, 2023:171

Martin Karresand

Digital Forensic Usage of the Inherent Structures in NTFS

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical
Engineering
Dept. of Information Security and
Communication Technology



Norwegian University of
Science and Technology

Martin Karresand

Digital Forensic Usage of the Inherent Structures in NTFS

Thesis for the Degree of Philosophiae Doctor

Gjøvik, June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU

Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

© Martin Karresand

ISBN 978-82-326-7046-8 (printed ver.)
ISBN 978-82-326-7045-1 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2023:171

Printed by NTNU Grafisk senter

Abstract

Digital forensic investigators have for a long time been burdened by an increasing amount of data to handle. Many solutions have been proposed. A yet unexplored feature is to use the inherent structures left by the allocation algorithm. These structures can be used to build a map of the allocation activity at different positions in a file system. The map can be used to plan and optimize the search for valuable data. We therefore have studied the inherent structures in the New Technology File System (NTFS) as a proof-of-concept to explore the possibility to create such a map. The map can increase the efficiency of many digital forensic processes, which has been verified experimentally for sampled hash-based carving. In file carving the map can help both during fragment extraction, as well as during file reassembly. Our research can also be used to verify time stamps and categorize the writing type of files based on the allocation pattern. It furthermore brings new knowledge to the research fields of external fragmentation and data recovery.

Preface

This dissertation is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The presented work was carried out at the Faculty of Information Technology and Electrical Engineering, Department of Information Security and Communication Technology (IHK) at NTNU from 2017 until 2022. The work was supervised by Associate Prof. Dr. Geir Olav Dyrkolbotn and Prof. Dr. Stefan Axelsson. This research received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project “Ars Forensica”, grant agreement 248094/O70.

Acknowledgments

This PhD thesis would not have been written without the continuous support of my two supervisors, Associate Professor Geir Olav Dyrkolbotn and Professor Stefan Axelsson. I would like to thank them dearly for keeping me and my research on track and having faith in me when the going was tough. They both are very good role models and have always given me support, encouragement, and inspiration to bring my research forward.

Likewise Associate Professor Frank Breiting, Professor Jessica Steinberger and Associate Professor Hanno Langweg in the assessment committee helped me to significantly increase the quality of the thesis through their apprehensive, excellent and highly valuable reviews.

I would also like to thank my colleagues at the Swedish Defence Research Agency (FOI), my friends at the Swedish National Forensic Centre (NFC), my past and present fellow PhD candidates at Norwegian University of Science and Technology (NTNU) in Gjøvik, and Professor Katrin Franke, head of the NTNU Digital Forensics group and opponent on my licentiate thesis in 2008. You inspired me to embark on this journey. Thank you all!

There is also my best friend Sassa, who has always been there for me. This thesis and PhD project, and my life, had not been what it is today without you. I have enjoyed every minute of our friendship. *Domo arigato gozaimasu, Sassa-san!*

And last but not least I would like to thank my fantastic and beloved wife and our wonderful children. You stood by me no matter what. You have endured years of hardship with a husband and father who always had his mind somewhere else, deeply buried in some strange scientific problem and with no time for you. Now I am finally back and I hope to be able to make up for at least some of the lost time. And that goes for the rest of my relatives too, so here I am, signed, sealed, delivered, I'm yours (again)!

Relevant Publications

The following publications directly contribute to the thesis. They can be found in Part II.

Article A M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8

Article B M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Using NTFS Cluster Allocation Behavior to Find the Location of User Data.” In: *Digital Investigation* 29 (2019), S51–S60. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.018

Article C M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Disk Cluster Allocation Behavior in Windows and NTFS.” in: *Mobile Networks and Applications* 25.1 (Feb. 2020), pp. 248–258. ISSN: 1572-8153. DOI: 10.1007/s11036-019-01441-1

Article D M. Karresand, G. Dyrkolbotn, and S. Axelsson. “An Empirical Study of the NTFS Cluster Allocation Behavior Over Time.” In: *Forensic Science International: Digital Investigation* 33 Supplement (July 2020), p. 301008. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2020.301008

A licentiate thesis has previously been defended by the author. That work is of relevance to the current PhD project.

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

In addition to the above publications the following peer-reviewed articles of relevance to the PhD project have been published by the author.

- M. Karresand and N. Shahmehri. “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*. Piscataway, NJ, USA: IEEE, 2006, pp. 140–147. DOI: 10.1109/IAW.2006.1652088

Relevant Publications

- M. Karresand and N. Shahmehri. “Oscar — File Type and Camera Identification Using the Structure of Binary Data Fragments.” In: *Proceedings of the 1st Conference on Advances in Computer Security and Forensics, ACSF*. ed. by J. Haggerty and M. Merabti. Liverpool, UK: The School of Computing and Mathematical Sciences, John Moores University, July 2006, pp. 11–20
- M. Karresand and N. Shahmehri. “Oscar — File Type Identification of Binary Data in Disk Clusters and RAM Pages.” In: *Security and Privacy in Dynamic Environments, Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22-24 May 2006, Karlstad, Sweden*. Vol. 201. Lecture Notes in Computer Science. Springer, 2006, pp. 413–424. DOI: 10 . 1007/0 - 387 - 33406 - 8_35
- M. Karresand and N. Shahmehri. “Oscar — Using Byte Pairs to Find File Type and Camera Make of Data Fragments.” In: *Proceedings of the 2nd European Conference on Computer Network Defence, in conjunction with the First Workshop on Digital Forensics and Incident Analysis (EC2ND 2006)*. Ed. by A. Blyth and I. Sutherland. Springer Verlag, 2007, pp. 85–94. DOI: 10 . 1007/978 - 1 - 84628 - 750 - 3_9
- M. Karresand and N. Shahmehri. “Reassembly of fragmented JPEG images containing restart markers.” In: *Proceedings - 4th Annual European Conference on Computer Network Defense, EC2ND 2008*. 2008, pp. 25–32. DOI: 10 . 1109/EC2ND . 2008 . 10

From 2001 and onward the author has also written or co-authored a number of other articles and technical reports. A selection of these can be found in Appendix G.

Contents

Abstract	i
Preface	iii
Relevant Publications	vii
List of Figures	xv
List of Tables	xvii
Acronyms	xix
I. Overview	1
1. Introduction	3
1.1. Problem Description	6
1.2. Theoretical Assessment of the Map Effect	9
1.3. Aim and Goal	12
1.4. Scope and Delimitations	12
1.5. Research Questions	13
1.5.1. Prerequisites	14
1.5.2. Research Questions	14
1.5.3. Significance of Research Questions	15
1.6. Outline of Thesis	16
2. Background	19
2.1. Storage Media and Structuring	19
2.2. File System	21
2.3. Fragmentation	21
2.4. Data Allocation	22
2.5. File Writing Concepts	24
2.6. NTFS	24
2.7. Encryption	26
2.8. Virtualization	28

2.9. Maps and Their Sciences	29
3. Related Work	31
3.1. File Fragment Carving	32
3.1.1. The Licentiate Thesis Work	32
3.1.2. Other Fragment Carving Research	34
3.2. Hash-Based Carving	35
3.3. Digital Stratigraphy	37
3.3.1. Digital Archaeology	39
3.3.2. Digital Geology	40
3.4. NTFS Fragmentation	41
3.5. NTFS Data Allocation Process	47
3.6. Data Recovery	49
3.7. Data Mapping	51
4. Experimental Setup	55
4.1. Motivation	55
4.2. Static Areas	56
4.2.1. Data Collection	57
4.2.2. Data Analysis	59
4.2.3. Map Evaluation	60
4.3. Repeated File Operations	60
4.3.1. Platform	61
4.3.2. Implementation	63
4.3.3. Map Creation	67
4.4. Writing Type Behavior	68
4.4.1. Virtual Hardware	68
4.4.2. Process Description	68
4.4.3. Implementation	69
4.4.4. Bitmap Manipulation	69
4.5. HDD vs. SSD Allocation Differences	71
4.6. BitLocker Allocation Changes	72
5. Result	73
5.1. HDD vs. SSD Allocation Differences	73
5.2. BitLocker Allocation Changes	74
6. Summary of Work	77
6.1. Article A: Creating a Map of User Data in NTFS to Improve File Carving	79

6.2. Article B: Using NTFS Cluster Allocation Behavior to Find the Location of User Data	80
6.3. Article C: Disk Cluster Allocation Behavior in Windows and NTFS	82
6.4. Article D: An Empirical Study of the NTFS Cluster Allocation Behavior Over Time	83
6.5. Map Examples	84
6.5.1. Static Areas	85
6.5.2. Allocation Activity	88
6.5.3. Advanced Map Attributes	88
7. Contributions	91
7.1. File Fragment Carving	91
7.2. Hash-Based Carving	92
7.3. Digital Stratigraphy	93
7.4. NTFS Fragmentation	93
7.5. NTFS Data Allocation Process	93
7.6. Data Recovery	94
7.7. Data Mapping	94
8. Conclusion and Future Work	97
8.1. Conclusion	97
8.2. Future Work	98
Bibliography	101
II. Included Publications	123
A. Creating a Map of User Data in NTFS to Improve File Carving	125
A.1. Introduction	126
A.1.1. Related work	127
A.1.2. Contribution	130
A.2. Experimental Setup	132
A.2.1. Data Collection	133
A.2.2. Implementation	136
A.2.3. Evaluation	136
A.3. Result	137
A.4. Discussion	141
A.5. Conclusion and Future Work	144
A.6. Bibliography	145

B. Using NTFS Cluster Allocation Behavior to Find the Location of User Data	153
B.1. Introduction	154
B.1.1. Background	156
B.1.2. Related work	157
B.2. Experiment	159
B.2.1. Platform	160
B.2.2. Implementation	162
B.2.3. Map creation	166
B.3. Result	166
B.4. Discussion	173
B.5. Conclusion and future work	175
B.6. Acknowledgment	176
B.7. Bibliography	176
C. Disk Cluster Allocation Behavior in Windows and NTFS	183
C.1. Introduction	184
C.1.1. Background	185
C.1.2. Related work	187
C.2. Experimental setup	188
C.2.1. Virtual hardware	188
C.2.2. Process description	188
C.2.3. Bitmap manipulation	190
C.3. Result	191
C.3.1. Block writing	192
C.3.2. Stream writing	198
C.4. Discussion	201
C.5. Conclusion and future work	204
C.6. Bibliography	204
D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time	209
D.1. Introduction	210
D.1.1. Background	211
D.1.2. Related work	212
D.2. Experiment	214
D.3. Result	218
D.4. Discussion	235
D.5. Conclusion and future work	238
D.6. Acknowledgements	239
D.7. Bibliography	239

III. Appendices	243
E. Extended Introduction	245
E.1. Forensics	245
E.2. Digital Traces	246
E.3. Digital Forensics	247
E.4. Forensically Sound	250
E.5. Digital Forensic Models	252
E.6. Digital Forensic Challenges	253
E.7. File Carving	254
E.8. Bibliography	258
F. A Layman’s Introduction	265
G. Extended Publications	267

List of Figures

1.1. Model of I/O ecosystem	5
1.2. Efficiency improvement calculation	10
4.1. File system fill	65
6.1. Relationship between publications and research questions	78
6.2. Map of static areas; full partitions	86
6.3. Map of static areas; first 5 GB of partitions	87
6.4. Map of the allocation activity	89
6.5. Map of the allocation activity over time	90
A.1. Probability of unique hashes	138
A.2. Part of \$DATA attribute of \$MFT file	140
B.1. File system fill	164
B.2. Frequency of cluster allocation; all machines	168
B.3. Frequency of cluster allocation; same operations	170
B.4. Frequency of cluster allocation 256 GiB	171
B.5. OS file position	172
C.1. Fragment size decrease for Windows 7 using <i>BM 7:1</i> layout	192
D.1. All OS mean allocation with standard file op	220
D.2. All 256 GiB with standard file operations	221
D.3. Win 7 maximum allocation using standard file operations	222
D.4. Win 10 maximum allocation using standard file operations	223
D.5. Win 7 median allocation	224
D.6. Windows 7 mode allocation	225
D.7. Windows 10 mode allocation	226
D.8. Win 7 standard deviation of allocation	227
D.9. Win 10 standard deviation of allocation	228
D.10. Windows 7 number of fragments	229
D.11. Windows 10 number of fragments	230
D.12. Windows 7 median stats	231
D.13. Windows 10 median stats	232

List of Figures

D.14. Windows 7 max sequence size	233
D.15. Windows 10 max sequence size	234

List of Tables

1.1. Efficiency calculation	12
3.1. Categorization of related work in file carving	33
3.2. Files per fragment interval and per specific fragment rate	42
3.3. Distribution of the number of fragments in files	44
3.4. Four types of storage patterns	46
3.5. File carving/data recovery tools	50
3.6. Data recovery live Linux distributions	51
4.1. Details of hashed computers	58
4.2. Windows types in file ops experiment	61
4.3. File operation settings	64
4.4. Unallocated areas after the <i>BM 7:1</i> manipulation	70
4.5. Modified areas of <i>BM 10:2</i>	71
5.1. Last User Journal Update Sequence Number increments	73
5.2. BitLocker file allocation differences	74
5.3. New files in BitLocker	75
A.1. Partition sizes and 0x00 fill	135
A.2. Evaluation result	141
B.1. Windows version in experiment	161
B.2. File operations	162
B.3. File operation settings	163
C.1. Unallocated areas after the <i>BM 7:1</i> manipulation	190
C.2. Modified areas of <i>BM 10:2</i>	191
C.3. The 11 largest free areas in <i>BM 10:0</i>	194
C.4. The 8 block write operations using <i>BM 10:0</i>	194
C.5. The 19 block write operations using <i>BM 10:1</i>	196
C.6. The 21 largest free areas in <i>BM 10:3</i>	197
C.7. The 3 block write operations using <i>BM 10:3</i>	197
C.8. The number of fragments and sizes using <i>BM 7:1</i>	199
C.9. The number of fragments and sizes using <i>BM 10:0</i>	200

List of Tables

C.10. Accumulated amount of first fragment sizes for stream writing	201
D.1. Windows versions in experiment	216
E.1. The matrix of data deposition vs. content/metadata	247

Acronyms

5WH who, what, when, where, why and how	154
AFF4 Advanced Forensics Format v.4	52
API Application Programming Interface	42
ASCII American Standard Code for Information Interchange	21
BFD Byte Frequency Distribution	256
CNN Convolutional Neural Network	34
CPU Central Processing Unit	15
CRATE Cyber Range And Training Environment	239
DFaaS digital forensics as a service	210
DMA Direct Memory Access	26
DFRWS Digital Forensic Research Workshop	247
DoD Department of Defense	248
EFS Encrypting File System	4

Acronyms

ELM Extreme Learning Machine	35
EXT4 fourth extended filesystem	214
FAM Factory Access Mode	155
FAIR Findability, Accessibility, Interoperability, and Reusability	57
FAT File Allocation Table	213
FBE File Based Encryption	26
FDE Full Disk Encryption	4
FFA First-Free Algorithm	48
FNN Feed-Forward Neural Network	34
FFS Fast File System	41
FOI Swedish Defence Research Agency	216
FRS File Record Segment	25
FTL Flash Translation Layer	155
GPT GUID Partition Table	19
GPU Graphics Processing Unit	157

HDD hard disk drive	4
HFS+ Hierarchical File System Plus	36
JPEG Joint Photographic Experts Group	32
LBA Logical Block Addressing	209
LCN Logical Cluster Number	20
LPVA Logical Partition Volume Address	212
MAC Media Access Control	15
MBR Master Boot Record	26
MFT Master File Table	211
NFC National Forensic Centre	v
NIC Network Interface Controller	15
NIJ National Institute of Justice	252
NIST National Institute of Standards and Technology	248
NTFS New Technology File System	210
NTNU Norwegian University of Science and Technology	v

Acronyms

OS operating system	210
PUP Parallel Unique Path	257
RAM Random Access Memory	212
RDC Real Data Corpus	133
RoC Rate of Change	32
SED Self Encrypting Drive	4
SHA-1 Secure Hash Algorithm 1	215
SSD solid-state drive	238
SVM Support Vector Machine	257
TPM Trusted Platform Module	26
TxF transactional NTFS	26
UEFI Unified Extensible Firmware Interface	28
VCN Virtual Cluster Number	20
VDL valid data length	213
VM Virtual Machine	13

XTS XEX Tweakable blockcipher with ciphertext Stealing 27

Part I.

Overview

1. Introduction

Dr Watson finds Sherlock Holmes crawling on his knees in the corner of a large parking lot, carefully examining every square meter of the ground in a strictly sequential pattern.

— What are you doing?

— I dropped Professor Moriarty’s smartphone when I exited my car over there.

— But why are you searching here, then?

— Patience, my dear Watson! I will eventually reach my car and find the phone.

Sherlock Holmes would definitely benefit from a map showing where to find Moriarty’s smartphone. The map should show the probability of the phone’s placement at different positions within the parking lot. When Holmes finally finds the (probably broken) phone he would also benefit from a map of the phone’s internal storage, showing the most probable position of Moriarty’s incriminating data.

Sherlock Holmes’s crime fighting colleagues around the world would definitely benefit from maps of digital storage media content, because they do like Holmes’s did in the parking lot and search the media sequentially [1]. The application area of a map can actually be expanded to anyone searching within digital data. Having access to a map showing the probable position of the sought after data would mean a real improvement. Conti et al. [2] have declared that

maps of binary objects can be used to assist humans in navigating to regions of interest. They can also be used to highlight certain desired regions and filter those that are less important to the analyst. [2, p. S4]

Already 2,500 years ago the Chinese general Tzu pointed out the importance of situational awareness, i. e. having access to information on your status, surroundings and enemies [3]. Information dominance is key to success and consequently who knows more wins. A common artifact laden with useful information is the map. Maps can for example be used for

decision making, for navigation, for education, for recreation, for information and for a host of further applications. [4, p. 1]

1. Introduction

Hence you cannot have situational awareness without access to a good map.

Currently the situational awareness in the digital forensic process is decreased by the fact that not all available information is used. A yet unexplored source of information is the inherent structures caused by the file system allocation algorithm [5, 6]. The approach is of special interest to file carving, but most fields dealing with retrieval of digital data from storage media will benefit.

This thesis study the concept of creating a map of the allocation activity in a generic (storage media) *partition* or volume¹. The concept is meant to be used for creating generic maps of specific combinations of a file system, operating system (OS), usage model, file system age, etcetera.

The file data allocation process, and hence the mapping concept, is applied at the logical level of a partition in the I/O ecosystem (see Figure 1.1). The concept is therefore independent of the partitioning of the storage media, the storage media type (hard disk drive (HDD), solid-state drive (SSD) or any other media format). Neither is it affected by Full Disk Encryption (FDE), because FDE is transparent to the file data allocation process [7]. FDE is either implemented in hardware by the storage media controller (called Self Encrypting Drive (SED)), or software based *block device encryption* (for example BitLocker, dm-crypt and FileVault [8]). Also software based *stacked file system encryption* (for example Encrypting File System (EFS) and eCryptfs) is working transparently to the file data allocation process. It creates an encrypted container, which is regarded as an ordinary file by the I/O ecosystem.

The partition maps can be used in any well-performed digital forensic process to speed up the access to relevant stored data. They can also be reused in many different situations, in the same way as a geographical map. The speed increase is achieved by reading blocks of data in the most relevant order based on the information in the map. In this way the probability of finding relevant data early in the process is increased, without skipping any data. To test the applicability of the mapping concept New Technology File System (NTFS) is used as a proof-of-concept.

There are (at least) two schemes used for data block (disk sector) addressing at the logical level of the I/O ecosystem. The Logical Block Addressing (LBA) is used for addressing the full storage media through the storage media controller. The Logical Partition Volume Address (LPVA) is used within a partition (see Figure 1.1). The mapping concept works at the LPVA level.

The mapping concept is future proof, because for each new hardware generation a new map can be created based on the same concept. The inherent structures created in a partition by the allocation algorithm(s) forms the foundation of the mapping concept.

¹Sometimes the terms partition and volume are defined differently (see Section 2.1). However, we will refer to both terms as a (storage media) *partition* in the thesis, as not to confuse the reader with other uses of the term “volume”.

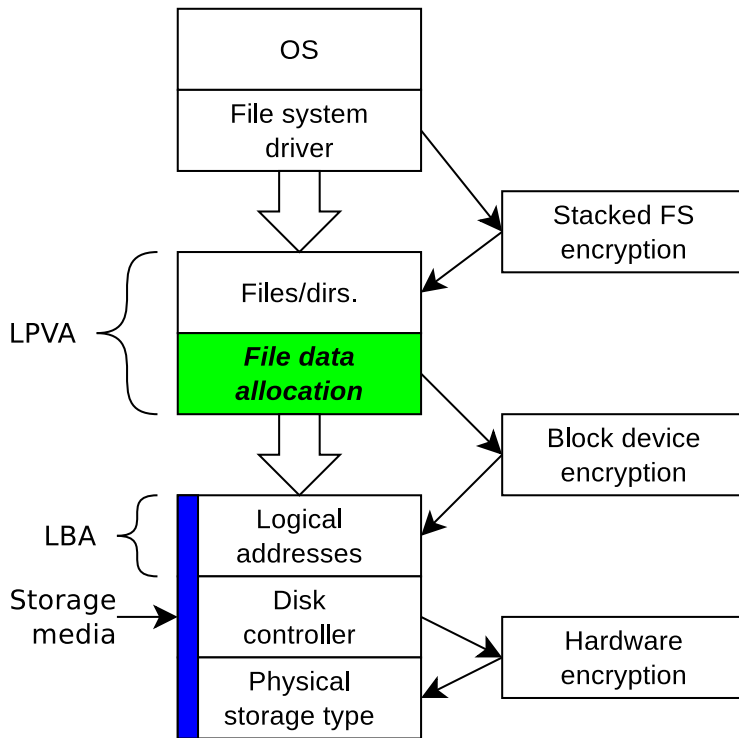


Figure 1.1.: A theoretical model of the I/O ecosystem and the placement of the thesis work (the green rectangle with the text “File data allocation”). Three types of storage encryption are also shown. Without encryption the process follows the big arrows. When using encryption the process instead passes the specific encryption type box. Neither the encryption types, nor the storage media type, affects the file data allocation step.

1. Introduction

General applicability is achieved by using a black box model, where the actual allocation pattern is studied, regardless of how it was created and by what I/O ecosystem. As long as it is possible to identify the partition boundaries in a storage media the concept can handle any I/O system and partition layout. We also show, as a proof-of-concept, how NTFS partitions can be identified in storage media.

In case of a digital forensic investigation, an investigator has to fully understand how data is arranged and placed; otherwise, interpretation of the data is questionable. This is certainly true and might seem to contradict the use of a black box model. However, interpretation of data is done for individual files and the black box model is applied at the file data allocation level, below the file level (see Figure 1.1). The file data allocation level is important for the file carving process to enable the file fragments to be properly reassembled. Such knowledge can for example be used to improve the ability to identify, collect and reassemble file fragments from broken file systems. It is also applicable to a wide range of research fields within and outside of digital forensics. Using a black box model does not interfere with the actual allocation pattern, it only hides the inner workings of the I/O ecosystem. Consequently our mapping concept is generically applicable.

An example of the generic applicability of the mapping concept is shown using the recently introduced 4 KiB Master File Table (MFT) records in NTFS. Due to them the \$MFT file can now be four times as large as when using 1 KiB records. This will possibly affect the placement of the maximum allocation activity, both through its larger size, but also through a higher allocation activity from the MFT records themselves, for example due to resident files. However, as long as the mean file size is much larger than an MFT record the bulk of the allocation activity will take place outside of the \$MFT file and consequently not affect the mapping concept. Furthermore, the mapping concept uses a black box model and can therefore handle any new formats by simply creating a new map incorporating the change.

The rest of the chapter presents the problem description, a theoretical assessment of the map effect, the aim, goal, scope and delimitations of the project. Also the research questions are given and motivated. The chapter is ended with an outline of the thesis itself. Readers needing a more thorough introduction to digital forensics and related research fields are recommended to read Appendix E in addition to Chapter 2. There is also a layman's introduction to the area in Appendix F.

1.1. Problem Description

Most fields related to digital forensics are burdened by the constantly increasing amount of data to handle [1, 9–20]. In the digital forensics field the situation has led to an increasing backlog of cases [1, 21, 22]. The negative effects of the backlog are noticeable

and the backlog is

already a hindrance frequently encountered in modern policing. Contributing factors to the backlog include the volume of cases, the volume of data, limited resourcing, limited manpower, alongside an overly arduous digital forensic process. [1, p. 5]

The increase in the amount of data to handle is driven by the digitalization of society [5]. Another factor is the growing capacity of storage media [23] in combination with their (relatively) slow interfaces [24, 25]. The field is also burdened by the tradition of searching storage media sequentially [1].

The reason for the sequential search strategy is the fact that the read speed of a disk is higher in sequential mode, than in random mode. Each read operation require the full chain of instructions to be processed. For HDDs there is also an extra movement of the disk head added to the processing time [26, 27].

The random read speed penalty has lead to a focus on developing faster algorithms. There is however a limit on the possible algorithmic improvements. Further development is therefore getting harder and harder. Meanwhile there are valuable resources wasted due to inefficient digital forensic processes [1, 21].

The increasing influx of cases, larger storage media sizes and higher number of items per case is also affecting the working environment of the digital forensic investigators. Therefore

[w]e can conclude that the pressure of cases, which leads to an increase in the total volume each examiner is asked to investigate in the same time, is one of the factors behind the delay of investigation. [28, p. 14]

The current situation with increasing backlogs is also affecting the rule of law, which is one of the corner stones of democratic societies. Consequently

[s]uch delays in processing evidence are harmful and will inevitably bog down the criminal justice system, giving offenders time to commit additional crimes and causing immeasurable damage to falsely accused individuals. [22, p. 1353]

When different digital forensics practitioners were asked to list the significant challenges of the field most of them listed problems surrounding data recovery, structuring of data, data reduction and unusual patterns in data [29]. The listed challenges show the need for information on how data are actually structured by the allocation algorithm.

Within file carving, a key component of digital forensics, the processing time is a major problem [30]. The situation can be compared to the famous needle in the haystack. Only that this time the needle is broken into pieces, there are volumes of irrelevant needle fragments present and the haystacks are growing.

1. Introduction

To reassemble a file the investigator first has to find all the file's fragments and nothing but the file's fragments. This is done through data type identification or hash comparisons. Then the correct order of the file's fragments has to be found [5, 31]. Consequently the increasing amount of data causes an explosion in fragment identification and ordering operations [32–38].

Yet another problem facing a digital forensic investigator is how to handle broken or corrupt storage media [5, 31]. Either (a part of) the data blocks are unreadable, or the file system is broken, or both alternatives. The consequences are affecting both the processing time, amount of extracted data and reliability of the results.

The situation gives rise to several consequences:

- The data acquisition process requires the integrity of the source and destination to be checked, usually by hashing the data. An intermittent reading error causes the hashing process to abort. Therefore the storage media either has to be omitted from the investigation, or its evidential value is diminished (the process is no longer forensically sound).
- The evidential value of any lost parts of the storage media is unknown.
- There is currently no way of knowing where to focus the remaining read operations of a failing disk without using the file system. This causes extra read operations, further damaging the disk.
- Reassembly of carved file data is done based on the the assumption of low fragmentation, a homogeneous data structure of the specific file type and a sequential allocation of the fragments. This assumption is seldom true [39].

The use of FDE can be compared to a broken or corrupt storage media from a digital forensic investigator's point of view. However, in the case of FDE the media can be seen as completely broken, because all data are unavailable. This is an increasing problem and has been for at least 15 years. Currently the best ways of mitigating FDE are to perform live (on-scene) forensic acquisition or persuade the suspect to give up the encryption key [9, 40–42].

The problems presented above limits the ability of the digital forensic investigator to fully utilize the evidence. It also prolongs the time taken to analyze the data and will in the end decrease the evidence value of the investigation. This leads to a self-fulfilling loop where the citizens' trust in the legal system is eroding.

By also utilizing information on the behavior of the allocation algorithm the efficiency of the forensic process will increase. Information on the allocation activity in different positions of a partition can greatly speed the investigative process up. This is for example important during live forensics of encrypted storage media, where time might be short for different reasons.

The allocation activity can be translated to the probability of finding user data in different positions of generic storage media. In that way the process can be focused to the areas where it matters most, without first reading a large amount of irrelevant data. The map format presents the information in a well known and accessible format.

Once again the need for a better understanding of the data allocation process and the resulting inherent structures is highlighted. Without situational awareness any critical mission is a lost cause, hence the idea of a map showing the allocation activity at different positions in a generic partition is well worth exploring. A map can be used to first plan the work, then tackle and finally conquer the massive amount of data. Exactly in the same way as a map would be used in classical warfare.

1.2. Theoretical Assessment of the Map Effect

Searching for something where it is more probable to find is better than searching based on any other parameter, as in the case with Sherlock Holmes and the dropped smartphone. This is the essence of the PhD thesis. The thesis transfers the concept to the digital forensic field of file carving and uses Windows NTFS as a proof-of-concept implementation. However, the concept is applicable to any file system that allocates storage media blocks in a deterministic way. It can also be used in any situation where knowledge on the allocation frequency at different positions in a file system (partition) is beneficial.

A typical application area for a map is the digital forensic field of hash-based carving. Sampling is currently used to speed the process up, but a higher speed (lower sampling frequency) means a lower detection rate. The use of a map will introduce the possibility to vary the sampling frequency over the partition and in that way improve the efficiency of the carving.

If we do not know where to increase or decrease the sampling frequency we have to use a static sampling frequency, which is the current situation in hash-based carving. The sampling frequency is chosen to give the optimal balance between speed and detection rate calculated over the whole partition. Figure 1.2 shows the static sampling frequency as a dashed red line. To theoretically estimate the efficiency improvement we assume we have a map that looks like the solid green line in Figure 1.2 (it is inspired by Figure 6.4). The map enables us to use a dynamic sampling frequency that follows the allocation frequency distribution of the map.

As can be seen in Figure 1.2 the static sampling frequency leads to over sampling at the beginning and end of the partition, and under sampling in the middle. Over sampling wastes time and under sampling decreases the detection rate. Consequently the use of our mapping concept increases the speed and/or the detection rate of hash-based carving.

1. Introduction

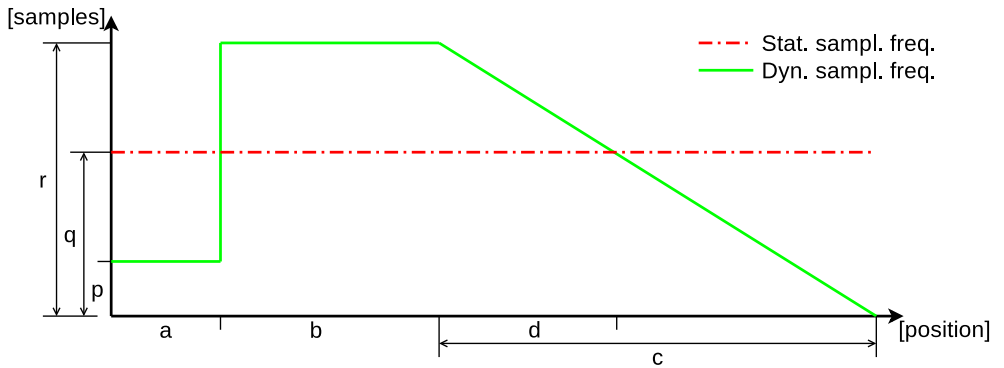


Figure 1.2.: A theoretical example showing the differences between a static and dynamic sampling frequency for hash-based carving. The time taken for collecting one sample is the same for both methods. The static sampling frequency is over sampling at the start and end of the partition, and under sampling at the middle. This is based on the assumption that the green solid line might also represent the allocation behavior of a generic partition. The figure shows a theoretical example and therefore does not contain any values on the X and Y axes. Even though the figure shows a line diagram the fictive values of the axes represent discrete LPVAs on the X axis and samples on the Y axis.

1.2. Theoretical Assessment of the Map Effect

Although the line diagram in Figure 1.2 only represents a theoretical usage model of a map, it still shows the principle and possible improvements of using the mapping concept. As long as the allocation activity of a file system is deterministic and generates a non-uniform allocation activity distribution, the mapping principle will improve any process where stored data are read without the use of the file system.

Based on Figure 1.2 the theoretical improvements of the mapping concept can be calculated. There are three typical use cases, where the static sampling frequency is either

1. set at the low starting value of the dynamic sampling frequency,
2. set at the maximum of the dynamic sampling frequency, or
3. set at a level giving an equal total amount of samples as the dynamic sampling frequency.

The areas under the curves in Figure 1.2 represent the total samples used for each sampling type. It is therefore possible to use the area difference to calculate the deviation from the optimal sampling rate (the dynamic sampling curve), compared to the actual amount of samplings for the static sampling. The equations for calculating the over sampling ratio, S_o , and under sampling ratio, S_u , as well as the distance d can be seen in Equations (1.1) to (1.3).

$$d = \frac{c(r - q)}{r} \quad (1.1)$$

$$S_o = \frac{a(q - p) + 0.5q(c - d)}{q(a + b + c)} \quad (1.2)$$

$$S_u = \frac{b(r - q) + 0.5d(r - q)}{q(a + b + c)} \quad (1.3)$$

From Figure 1.2 we estimate $a = 8$, $b = 16$, $c = 32$, $p = 4$ and $r = 20$. The rest of the variable values, as well as the over and under sampling rates, are shown in Table 1.1.

As can be seen in Table 1.1 40% of the samples are wasted when using static sampling with the same detection rate as for the maximum of the dynamic sampling. When we use the same total amount of samples for the two sampling models the static sampling wastes 26.7% of the samples and get a detection rate 26.7% lower than the dynamic sampling. These results are based on theoretical estimations of the real world and consequently only give an indication of the possible improvement using the mapping concept (a dynamic sampling rate) compared to a static sampling rate.

1. Introduction

Table 1.1.: The over and under sampling when using static sampling relative to dynamic sampling based on Figure 1.2. The values of d and q are also given.

Static sampling level	d, q	S_o [%]	S_u [%]
Low	$d = 25.6, q = 4$	11.4	205.7
Maximum	$d = 0, q = 20$	40	0
Equal	$d = 12.8, q = 12$	26.7	26.7

1.3. Aim and Goal

The aim of the research is to improve the digital forensic process by exploring the inherent structures left in a partition by the allocation algorithm. The concept can be generalized to a study of the allocation behavior and the patterns it leaves within storage media. Using that knowledge the idea is to create a map of the allocation behavior at different positions within a generic partition, regardless of the OS and file system.

Any extra knowledge gained during the process is meant to accompany the map and further improve the digital forensic process. Ideally the map should be usable in any situation where knowledge on the allocation pattern and behavior of a system is beneficial. The concept is therefore meant to be applicable also outside of the digital forensic field.

The goal of the research is to construct a proof-of-concept map of the allocation activity at different positions in an generic NTFS formatted partition. During that process the feasibility of a map will be explored. Any features related to the allocation process will be studied alongside the feasibility study. To improve the applicability of the proof-of-concept results live data from real computers will be used as much as possible. Preferably the result should have a wide coverage of application areas.

By constructing a map we will give the digital forensics community a versatile tool with many areas of application. We have not yet found any other research pursuing the same idea. Our work will therefore open a new, yet unexplored, area of research.

1.4. Scope and Delimitations

The research presented in the thesis is delimited to studies of NTFS, which is the default file system in the Microsoft Windows OS [43–45]. In January 2023 Microsoft Windows had a market share of more than 74% for desktop computers [46].

The mobile phone market is larger than the desktop computer market and would therefore seem natural to use as a proof-of-concept. However, the storage media of a mobile phone is much harder to access at the low levels of the OS stack required in our experiments, than the storage media of a desktop computer. Consequently we find NTFS formatted desktop computer storage media to be the best alternative for a feasibility study of the mapping concept, with accompanying proof-of-concept implementations.

Our study on static areas² in NTFS is delimited to 30 unrelated real world NTFS formatted main partitions³ of Microsoft Windows home and office computers. This limits the foundation of the research, which would preferably have been much bigger to secure the statistical significance of the results. Increasing the number of computers would possibly result in a decreased (or even zero) number of fully static areas. However, the collected data set is enough to prove the feasibility of the mapping concept.

We used a limited amount of Virtual Machines (VMs) and file operations for the experiments on allocation behavior. Even though the significance of the results therefore might be challenged, they are large enough to work as proof-of-concept. The results show that it is possible to create a map of the allocation activity at different positions of a partition. The focus of the research has been a high level of authenticity. We have used a framework that makes the experiments easy to extend. An extension would add better redundancy and higher resolution, but the overall configuration would be the same.

1.5. Research Questions

The research gap presented in Section 1.1 gives rise to the following question:

Can the inherent structures introduced by the allocation algorithm of a file system be used to improve the digital forensic process and if so, how?

As described earlier in Chapter 1 the *how* can be answered by the introduction of a map showing the allocation activity within a partition.

The scope of the question is large because of the many different file systems and allocation algorithm implementations currently existing. However, as long as the data allocation algorithm used in a file system does not change, the mapping concept can

²A static area is defined as an area containing the same data at the same position in different partitions containing the same type of file system.

³We use the term *main partition* to represent the partition in which the OS resides. This partition is called the *boot partition* by Microsoft. From Microsoft Windows 7 and onward the partition with the boot loader is called the *system partition* and is separate from the boot partition. Before Windows 7 the system and boot partitions were the same [47].

1. Introduction

be used for any file system. Consequently it is possible to use a single file system as a proof-of-concept. We have chosen to use NTFS because of its ease of access and its dominant market share for computers.

This section first presents the prerequisites for the formulation of the research questions, which then are presented in a separate section. Furthermore, the significance of each of the research questions and their connections to the contribution of the PhD project are discussed. The contributions of the research is presented in Chapter 7.

1.5.1. Prerequisites

To be able to create a map of the allocation activity in a NTFS formatted partition some prerequisites have to be fulfilled. First of all we must know if there are any structures to be used at all. We also need to know if they are generic, i. e. if the structures are present regardless of OS version, size of the partition, use case of the file system, etcetera.

Secondly we need to know the distribution of the allocation activity over the file system. The map will be of no value if the distribution is even. At the same time the distribution should be as generic as possible. The same is true for the structures presented above.

It is also desirable to have a better understanding of the detailed allocation behavior at the file level. Are there any differences depending on file type, writing behavior, file size, etcetera? Such information will for example give typical fragment sizes, changes of the sizes, and distances between fragments. The information can direct the investigator to the most probable positions of the remaining fragments of a file. It will furthermore indicate the proper ordering of the fragments during reassembly.

It is also good to get a better understanding of the allocation behavior over time and different OS versions. Is the distribution fixed or varying during the life time of a partition. Does it vary between different versions of Microsoft Windows? By incorporating information on the aging of a file system the precision of the map will be higher. The situation can be compared to the slow movement of the magnetic poles of the earth, the tectonic movements, the lithospheric flexure movement due to the last glaciation, etcetera.

1.5.2. Research Questions

Different OSs might use differing allocation algorithms and hence cause their own specific data patterns. However, the proposed mapping principle is the same, only the allocation activity in different areas vary. Due to the market share of Microsoft Windows, NTFS is used as the foundation for the research. The main research question of the PhD project therefore can be formulated as:

How can the inherent structures created by the allocation algorithm be used to create a map showing the allocation activity at different positions in the NTFS formatted main partition of a generic Microsoft Windows computer?

The main research question gives rise to a number of new research questions. They together cover the prerequisites for the creation of a map. All questions are answered in the publications (see Part II) written during the PhD project. The restrictions of the main research question are implicit for all new questions. The new questions are:

RQ:1 *What generic (static⁴) structures are there?*

RQ:2 *How is the allocation activity distributed?*

RQ:3 *What differences in allocation behavior are there at the file level?*

RQ:4 *How does the allocation activity distribution vary between different Microsoft Windows versions?*

RQ:5 *How does the allocation activity distribution vary over time?*

The answers to Research questions RQ:1 to RQ:5 together show if and how the main research question can be answered. The motivation for each research question and their significance for the main research question is discussed in Section 1.5.3.

1.5.3. Significance of Research Questions

Research question RQ:1 is based on the assumption that there are too many parameters affecting the life of a Microsoft Windows computer to generate any common structures in a generic NTFS partition. Typical parameters are unique hardware configurations, for example serial numbers of the Central Processing Unit (CPU) and the Media Access Control (MAC) address of the Network Interface Controller (NIC). There are also unique user settings, as well as user behavior affecting the computer. If the hypothesis on the lack of file system structure is true, a map cannot be created. If the hypothesis is false, i. e. there are static areas in a generic NTFS formatted main partition, the foundation for a map is present.

The hypothesis behind Research question RQ:2 is based on the assumption that the allocation activity is evenly distributed over the file system. If the hypothesis is true there are no allocation differences to base a map on. If the hypothesis is false, there are

⁴The term *static* is used to describe an area containing the same data at the same position in different partitions (and even in different hardware). Please observe that the version of the OS in the computers might also differ.

1. Introduction

areas with higher allocation activity and it is possible to create an allocation activity map.

Data can be written in two ways, either as a stream of unknown size or as a big block with known size. The hypothesis behind Research question RQ:3 is based on the assumption that functions in the OS and file system hide how data are written. Therefore the system will always show the same allocation behavior, regardless of the type of writing.

If the hypothesis behind Research question RQ:3 is true the map would be of less value due to a lack of important information. If the hypothesis is false, i. e. there are differences in how block versus stream written data are allocated, a more versatile map will be produced. This for example benefits the file carving field, where identification and reassembly of file fragments will be easier. The information will also indicate the causal order of files, which can be used to check the validity of time stamps.

The hypothesis behind Research question RQ:4 is based on the assumption that there are no differences in allocation behavior between different versions of Microsoft Windows (from Windows 7 and up). The version number of NTFS has been the same since Windows XP [48, 49]. Hence NTFS is a well established file system with an already fully optimized allocation algorithm.

If the hypothesis behind Research question RQ:4 is true it is not possible to connect a NTFS partition to a specific Microsoft Windows version. If the hypothesis is false, it would be possible to identify the OS version used in connection with a partition, for example when attributing a disputed storage media. The information can also be used to create a specific map for each OS version and in that way increase the precision of the map.

The hypothesis behind Research question RQ:5 is based on the assumption that NTFS is not subject to aging effects. Consequently there should not be any differences in the behavior between early and latter allocations. The idea is that the allocation activity will be evenly distributed until the partition is fully utilized and then concentrated to areas where files have previously been deleted.

If the hypothesis behind Research question RQ:5 is true it will not be possible to estimate the age of a file system from the allocation pattern alone. If the hypothesis is false it will be possible to create maps adjusted for the age of a file system, as well as estimate the file system's age from the allocation pattern.

1.6. Outline of Thesis

In Part I of the thesis Chapter 2 gives a short background of the work and Chapter 3 presents a selection of related work relevant to the PhD project. Chapter 4 then presents details of the experiments executed in the PhD project, with Chapter 5 presenting the

results of the experiments on allocation differences between HDDs and SSDs, as well as the differences induced by the use of BitLocker FDE. In Chapter 6 the publications of the PhD project are listed and set into context. Chapter 7 presents the contributions of the PhD project. The conclusions drawn from the research work and suggestions for future work are given in Chapter 8.

Part II of the thesis presents the articles published during the PhD project. The articles are complete, but have been adjusted to the layout of the thesis and also had some minor editorial changes. The changes are described in the introductory text preceding each article.

Part III contains an extension of the background on digital forensics with related fields in Appendix E, as well as a layman's introduction for non-computer scientists in Appendix F. Lastly an extended list of publications by the author is presented in Appendix G.

2. Background

This chapter should give the reader the necessary background needed to understand the rest of the thesis. It covers storage of data, file systems, data allocation, NTFS and the sciences surrounding maps.

2.1. Storage Media and Structuring

The storage media is (mostly) divided into one or more partitions. The layout is held in a partition table placed at the beginning of the storage media [50]. Modern GUID Partition Table (GPT) based partition tables can contain a large number of partitions, but the Microsoft Windows implementation only allows up to 128 partitions [51]. A partition is a collection of consecutive sectors of a storage media.

A volume is created from one or more storage locations and may or may not have consecutive addresses allocated to it. It therefore can consist of several partitions from different storage media, but can also be divided into several partitions. The confusion is manifested by Carrier in a figure where a volume contains three partitions, each holding a volume[50, p. 58]. However, often the two terms are used as synonyms [50]. We use the term *partition* for both partitions and volumes in the thesis to avoid confusion with other uses of the term “volume”.

The physical construction of an HDD and SSD differs greatly. A major constructional difference is the wear leveling used in SSDs, where data is regularly moved around to even out the wear on vital components. However, at the logical, file system, level (see Figure 1.1) the storage media controller hides the underlying hardware differences [52–57]. van der Meer et al. writes:

Wear leveling, a technique employed by SSD firmware to extend the lifetime of the device, distributes file blocks evenly over the physical storage. However, this is handled transparently by the firmware and thus does not affect allocation of blocks at the file system level. [53, p. 2]

Kumar writes the following regarding the wear leveling process:

To make things consistent with the operating system, the controller will remap the block’s logical address on the fly. [52, p. 7]

2. Background

The fact that the storage media type is not affecting the file data allocation pattern has also been proven experimentally (see Sections 4.5 and 5.1). The allocation patterns for the test files were identical in the HDD and SSD. The only major difference between the partitions were the time stamps of the files.

Storage media is divided into blocks of data called sectors, allowing shorter addresses to be used by addressing the blocks instead of separate bytes. This allows larger capacity storage media to be used without exhausting the address space. The size of the sectors are mostly based on multiples of 2, with 512 as the de facto standard [58], but large (newer) disks often use 4 KiB sectors [59, 60].

The file system can use a logical sector size different from the size of a storage media sector. Consequently a disk with 512 B storage blocks might have a logical block size of 4 KiB and vice versa [60, 61]. The actual physical size of a sector is slightly larger than the logical, because each sector has a checksum attached to it to enable detection (and possibly correction) of errors. This process is handled by the disk controller and therefore is transparent to the user [62].

The storage media controller converts between the addresses given by the OS and the LBA addresses of the storage media. The partitions have their own logical address space called LPVA [50]. A LPVA address is an abstraction of the underlying physical layer of the storage media to simplify the work of the file system. The LBA addresses are measured in disk sectors from the physical start of the storage media. The LPVA addresses are instead measured in disk sectors from the start of a partition. The address types might correspond, but need not do [50].

The Microsoft Windows term for an LPVA is Logical Cluster Number (LCN) [48]. Microsoft also uses the term Virtual Cluster Number (VCN) for the addresses of the data blocks within a file. Consequently a VCN is unique only to its file and is used to keep track of the ordering of the file data blocks [48]. Since LCN (and VCN) are Microsoft specific terms the term LPVA will be used throughout the thesis to maintain a general applicability of the mapping concept. However, NTFS usually use data blocks (clusters) of 4 KiB, while disk sectors might be either 512 bytes or 4 KiB. Consequently an LCN address might be eight times lower than an LPVA.

The storage media controller is also used to hide damaged parts of the storage media from the file system. In HDDs the addresses of the damaged parts are held in the G-list and P-list. The P-list contains damaged sectors found during post-manufacture testing. The G-list is used for sectors that become corrupt during the use of the disk [63, 64]. Similar functionality can be found in SSDs [65].

2.2. File System

A file system is an integral part of an OS. Its main function is to keep track of the files in the computing device [66]. Files are often divided into two types. *System files* are files that are part of or created by the OS of the computing device. *User files* are files that are created by processes induced by the user(s) of the computing device [66].

Files are also divided into binary files (allowing all possible symbols) and text files (containing only printable symbols) [66]. Text files often contain only symbols from the American Standard Code for Information Interchange (ASCII) character set. This is currently being replaced by Unicode, which includes 149,186 characters as of version 15.0 [67]. The Unicode character set covers symbols from most languages of the world [68].

The data held in a file system are divided into *metadata* and *file data*. The metadata of each file are usually held in a file record (a system file or data base). The metadata of a file contain for example the file name, parent directory, time stamps and pointers (addresses) to the positions of the file's data [66]. The storage media arranges data in sectors. However, file systems sometimes use their own data blocks, which might differ in size from the storage media sectors. For example Microsoft Windows NTFS uses a block called *cluster*, which usually is 4 KiB in size [69]. The NTFS address space is therefore measured in clusters instead of sectors.

The file records are often held in a table, for example called File Allocation Table (FAT) and MFT in Microsoft's FAT32 and NTFS respectively, and inode table in Linux's ext2 [70, 71]. The file table is searched when a file should be located and its file meta data record is retrieved. Then the pointers to the blocks holding the files data are used to read the file into Random Access Memory (RAM). Often the file table is held in RAM to minimize the file system I/O [66].

The files held in the file system are usually organized into a hierarchical (tree) structure of directories. Each level in the tree can hold files, even at the root of the tree. The directories are also containing metadata, for example their name, as well as time stamps and addresses of the file records held within the directory [66].

To keep track of which blocks are currently occupied with data a bitmap is used. The bitmap contains one bit per logical block. If a block is allocated (occupied) the corresponding bit is set to 1 and vice versa. This function allows the file system to quickly find free blocks to allocate when data are added [66].

2.3. Fragmentation

During the file data allocation process the avoidance of fragmentation is of utter importance. There are two types of fragmentation occurring in file systems, *internal* and

2. Background

external. Internal fragmentation occurs when the size of a file is not evenly divisible by the size of the smallest storage block to be used. Therefore an extra data block has to be allocated to store the remainder of the file [58]. Using NTFS as an example a 4,097 B large file will be stored in two 4 KiB clusters, wasting 4,095 B of space. The same situation might occur in other file system too, using even bigger blocks of data.

External fragmentation occurs when a file does not fit into any of the remaining unallocated areas and has to be split into several pieces. The allocation algorithms differ in how well they handle external fragmentation. A higher fragmentation rate leads to lower access speed [58].

SSDs are less affected by external fragmentation than HDDs, which have moving parts. An HDD has a typical seek time of 15 ms and an SSD 0.1 ms. Consequently even mild external fragmentation is noticeable on HDDs. An SSD has to suffer from heavy external fragmentation before the delay is being noticeable [72].

There are different reasons for file system fragmentation, for example low disk space, file editing, appending data to files, file system forced fragmentation and wear leveling in flash based storage media. The file system forced fragmentation occurs in UNIX file systems built on extents. Wear leveling creates fragmentation, but not on the LPVA level of the file system, only at lower layers [73–75].

Knowledge on the effect of data fragmentation is important in digital forensics. When carving files scattered blocks of data need to be identified and reassembled using only information derived from the fragments themselves. The fragmentation destroys some of the ordered structures created by the allocation algorithm. At the same time the data fragmentation leads to a more efficient use of the available storage. The fragmentation also affects the map creation process by scattering the allocation activity over the partition. Consequently it decreases the resolution of the map.

2.4. Data Allocation

There are three main algorithms used to allocate free space in computer storage. The algorithms are *contiguous*, *linked* and *indexed allocation*. Contiguous allocation allocates a file as one contiguous block of storage media sectors or file system blocks. This method is only suitable for system where files are never deleted or modified [58]. A typical example is a CCTV recording system working in a round-robin fashion.

Linked allocation creates a linked list of a file's data. This strategy can handle all types of file operations. However, a reading error will break the linked list, causing the rest of the file's data to be lost [58].

The indexed allocation strategy is currently the most popular. It works by using a separate index of the positions of a file's data blocks. The strategy handles external fragmentation well, but might still require regular defragmentation. There is also a

risk of disk space being wasted due to internal fragmentation. A typical example is a small file requiring a full index metadata block to hold just a few index posts [58, 66, 76].

When new file data are allocated different algorithms are used depending on the OS and file system. The internal interactions between the different parts of the I/O ecosystem for a combination of OS and file system might be complex. However, by regarding them as a black box their combined internal behavior can be studied regardless of their complexity. The black box principle is used in this thesis to make the mapping concept independent of the OS and file system combinations and their inner workings. The concept builds on the observable file data allocation behavior of an OS and file system in combination.

The more popular allocation algorithms are *first fit*, *next/nearest fit*, *best fit*, *worst fit* and *quick fit* [58, 66, 76]. They might be differently implemented in different OSs and file system drivers, but the main principles of their functions remain. The following list presents their specific features:

First fit starts the search for free space at the first position in the file system. The goal is to find a free space large enough to hold the entire file. If none of the free spaces are large enough, a fragment of the file is written at the first free area and the search continues for the next free area. The procedure is repeated until all fragments of the file are written to disk [58, 66, 76].

The first fit allocation strategy causes the first part of the file system to be more heavily utilized than the rest. Eventually the area of maximum allocation activity will move when the partition is filled up. The active allocation area will therefore slowly progress towards the end of the disk.

Next fit (also called *nearest fit* [76, p. 544]) is similar to the first fit allocation strategy. However, the search for free space is started from the position of the last allocation position. In that way the allocations are evenly spread over the file system. The writing speed is also improved in HDDs due to less head movements [66].

The next fit algorithm fragments files for the same reason as first fit [66]. However, the LPVAs of fragments might wrap around using next fit, especially when the file system is heavily utilized.

Best fit allocates the free space best fitting the new file. This lowers the amount of wasted space and keeps the free areas as small as possible [58, 66, 76]. When the large free spaces are used up new files will be fragmented.

All free spaces must be known in advance to enable the selection of the best fitting space [58, 66, 76]. This can be solved by a sorted list kept in RAM.

2. Background

Worst fit is the opposite strategy to best fit. It allocates new files to the worst fitting free space. i.e. the biggest of those large enough to fit the file. In this way the areas of remaining free space are kept as large as possible, but are still used. Therefore the probability of keeping smaller files unfragmented increases compared to best fit. The strategy is slow unless a sorted list of free spaces are kept in RAM [58, 66].

Quick fit use several lists of free blocks. Each list covers a certain size range. The search for the best fitting free space is therefore extremely fast [66]. However, the use of multiple lists makes file deletion complex. The lists must then be searched for adjacent free spaces to merge. Without merging them the file system quickly becomes fragmented. Quick fit is suitable when the file sizes are fairly even [66].

The listed strategies are generally applicable and for example also used for allocation in RAM. The first fit and best fit algorithms generally have a better storage utilization than worst fit. The first fit algorithm is the fastest during normal use [58, 66].

2.5. File Writing Concepts

Files can be written to disk in two ways; either as a stream of data, or as a single block [77]. During stream writing the final size of the file is unknown. Consequently the free space allocation cannot be optimized. This often leads to file fragmentation, but the effect is diminished by the internal buffering of the OS. During block writing the OS knows the file's size in advance and can optimize the allocation accordingly [78].

The specific write behavior is application dependent. A word processor normally use block writing and streaming media from the internet use stream writing. Block writing might also be used for temporary backing up files in case of a power loss or hardware failure. However, the exact behavior has to be decided individually for each application [78].

2.6. NTFS

This section gives a brief overview of Microsoft NTFS. A "\$" sign preceding a term indicates that it is a file name. Consequently the term \$MFT represents the concept of a file table as used in NTFS and the term \$MFT represents the file containing the MFT [48]. Readers interested in a detailed description of the I/O ecosystem of NTFS are recommended to read the book *Windows Internals part 2* by Allievi et al. [48]. Also for example Carrier [50] and Tanenbaum and Bos [66] have information on NTFS, although somewhat dated.

NTFS is the default file system of current Microsoft Windows versions [43–45, 50]. In NTFS everything is regarded as a file, even the MFT itself. It can be fragmented and distributed over the partition. Its starting address is given in the boot sector of the partition [48, 50].

Each file in the file system has at least one record in the MFT. A record is called a File Record Segment (FRS) by Microsoft [79], a file record by Allievi et al. [48] and an MFT entry by Carrier [50]. We will call them MFT records, because that is the most common term according to Google and it is also sometimes used by Allievi et al. [48, p. 663]. An MFT record can either be 1 or 4 KiB in size [48] and the first 42 bytes contain a record header [50]. The rest of the space is filled with file related data held in file attributes [48, 50]. If the metadata of a file require more space than available in a single MFT record multiple MFT records can be used. The original MFT record is called base record and contains links to the the extra MFT records [50]. In that case the attribute of the base record containing the links is called an attribute list [48].

Microsoft uses an index allocation strategy for the file system metadata [69, 80]. The risk of internal fragmentation during index allocation is mitigated by storing the data of smaller files within the MFT records¹ themselves. Such files are called resident files [69].

Microsoft Windows XP is said to use the best fit allocation strategy for NTFS [50]. However, experimental results have shown that a first fit strategy is used for newly formatted partitions. The strategy is switched to the best fit when the file system is more heavily used [84, 85].

When formatting a NTFS partition 12.5% of the space is reserved for the MFT. The smallest allocatable unit in NTFS is called *cluster* and is usually 4 KiB in size [69]. Consequently there can be either one or four MFT records in a cluster. Using 4 KiB MFT records will cause the \$MFT to grow faster, but the 12.5% reserved space will be the same and the \$MFT will still grow with one MFT record (one allocation) per file. There is room for more than 30,000,000 files (4 KiB MFT records) within the reserved space in a 1 TB partition. As long as the mean size of the files is much larger than an MFT record the main part of the allocation activity will take part outside of the \$MFT. Consequently the size of the MFT records is not affecting the mapping concept or our results to any greater extent.

NTFS uses the \$Bitmap file to keep track of the allocation status of each cluster. Record 0 is the MFT itself and record 1 is a backup of the first four records of the

¹The maximum size of an internal \$Data attribute in an MFT record varies depending on the size of the other attributes. Most sources give a maximum internal \$Data attribute size of 600 to 700 bytes for 1 KiB MFT records [50, 81–83]. Microsoft reports a 900 byte limit for 1 KiB records [69]. The attributes of a 4 KiB MFT record will probably not grow exactly four times in size. Consequently the maximum residential file size will probably be approximately 3.6 KiB for 4 KiB records. However, we have not been able to verify that experimentally nor theoretically.

2. Background

MFT. The \$Bitmap file is record number 6 and record 8 is a list of bad clusters [50].

The I/O process of NTFS is protected by a journaling system, where each operation (called *transaction*) is made atomic. Consequently either the file operation is executed in its entirety, or not at all. The transactional model of NTFS is called transactional NTFS (TxF) [48].

2.7. Encryption

Data-at-rest encryption is important to protect stored data from unauthorized access. The simplest protection is offered by encrypting individual files, which are protected as long as they are encrypted. A better solution is to either encrypt the storage media, or create an encrypted container holding all files that need protection. There are three main types of data-at-rest encryption functions affecting more than one file at a time.

Stacked file system encryption (or File Based Encryption (FBE)) is a software based container function creating a large encrypted file with a virtual file system inside. The virtual file system can then be mounted and used as an ordinary disk for file storage. The container stays encrypted as files are read, written and modified through the container software, which acts as a proxy between the user and the storage (the encrypted container) [86, 87]. Typical examples of stacked file system encryption softwares are Microsoft's EFS [88] and eCryptfs [87] in Linux.

Block device encryption (or FDE) is a software based function that encrypts a full storage media or partition. As for the container format the encryption or decryption is done on-the-fly when data is written to or read from the encrypted block device [86]. This type of encryption is often combined with a hardware module called a Trusted Platform Module (TPM) used to handle the encryption keys [89]. Typical examples of block device encryption software is Microsoft's BitLocker, dm-crypt in Linux and FileVault in MacOS [8].

Hardware based encryption (or SED) performs encryption of the complete storage media. The encryption process and keys are all handled inside the storage media controller. The keys are not stored permanently outside of the controller, a fact that mitigates a number of attacks against the SED (cold boot, Direct Memory Access (DMA) and evil maid attacks for example). In that way also the Master Boot Record (MBR) and all partitions can be encrypted without affecting the boot process of the digital device [8, 86].

All three encryption types work either above (FBE) or below (FDE and SED) the file data allocation layer. In other words,

FDE and the underlying block device work with the disk sector as an atomic and independent encryption unit, which means that FDE can be transparently placed inside the disk sector processing chain. [7, p. 1]

Furthermore,

FDE concerns data at rest stored in the disk only. Data is encrypted before being stored in the disk and decrypted after being loaded from the disk which means that the disk should always store encrypted data. [86, p. 2][90, p. 13]

It is also stated that current FDE, which is length preserving and uses standard encryption algorithms,

[...] significantly simplifies disk encryption management as the disk mapping does not change with encryption.

[...]

Moreover, in a disk, each sector is addressed by the Logical Block Address (or LBA) and to simplify the integration of encryption, this mapping is kept intact when data is encrypted. [91, p. 1]

Consequently the file allocation process is neither affected by the use of FDE, nor SED and FBE. This has also been confirmed experimentally for FDE (see Section 4.6).

A weakness of standard block device encryption is the fact that two storage media sectors with equal content might get the same encrypted content (*deterministic encryption*). That can help an attacker to guess the content with higher probability. Likewise there is no cryptographically sound protection against unauthorized modifications of the encrypted data (*authentication of encryption*) [91]. The standard cryptographic solution would be to store a salt in each sector, but that would be hard in already used storage media, because each sector is fully occupied by data [90, 92, 93].

In current block device encryption there are some features implemented that somewhat mitigates the risk of deterministic encryption and authentication of encryption. This is done through the encryption mode used by the FDE. Currently XEX Tweakable blockcipher with ciphertext Stealing (XTS)² is the most commonly used mode. It has acceptable resistance to deterministic encryption and acceptable authentication of encryption [86, 90, 91].

BitLocker is the default FDE in Windows and is automatically enabled in Microsoft Windows 8.1 and above for all computers supporting Modern Standby [94]. However, the encryption is not activated until the user logs into a Microsoft or an Azure Active Directory account. For the automatic encryption to work the following requirements will have to be fulfilled (they are valid for Windows 10, version 1709 and above) [95]:

²XEX stands for *Xor-Encrypt-Xor*.

2. Background

- A TPM v. 1.2 or v. 2.0 chip must be present
- Unified Extensible Firmware Interface (UEFI) Secure Boot is active
- Platform Secure Boot is active
- DMA is protected

Furthermore, the BitLocker setup requires the storage media to be partitioned into (at least) two partitions. This has been the standard partitioning scheme at installation since Microsoft Windows 7 [96].

2.8. Virtualization

Virtualization is used to enable several independent virtual computers to be run simultaneously on the same hardware. In that way the hardware can be used more efficiently. The foundation of the virtualization is the hypervisor, which isolates the virtual computers from each other. The computers need not run the same OS and can have different virtual hardware configurations. There are two types of hypervisors [97]:

Layer 1 hypervisors, also called bare metal hypervisors, act as a thin software layer between the hardware and the virtual computers. The hypervisor acts as an OS, so no other software is needed for the virtualization, making layer 1 virtualization almost as fast as running the virtual computers directly on real hardware. Layer 1 hypervisors are therefore often used in cloud computing and enterprise environments. Examples of layer 1 hypervisors are Xen, Hyper V and KVM [98].

Layer 2 hypervisors are easier to use and have a wider range of functionality than layer 1 hypervisors. However, they need a separate OS to run, because they are implemented as standard software. This makes them slower than layer 1 hypervisors and therefore often are used in private computers and non-enterprise environments. Typical examples of layer 2 hypervisors are VirtualBox, VMWare Workstation and QEMU [98].

Oracle's VirtualBox, a layer 2 virtualization software, supports BitLocker encryption on its virtual disks from version 7.0.0, since a virtual TPM and Secure Boot module have been implemented [99]. However, automatic activation of BitLocker requires the "Modern Standby" standard to be fulfilled and the term "Modern Standby" is not used in the VirtualBox User Manual v. 7.0.6, not even the string "tandby"³ is present [100]. Consequently a VirtualBox virtual disk running Windows 8.1 or higher will not be

³The search was done using Firefox internal search function, which does not implement wildcards. Therefore the "S" was left out to avoid possible case sensitivity.

BitLocker encrypted unless the user manually enables it, at least in theory. We have not verified this experimentally.

The TRIM command is used in SSDs to help the garbage collection and wear leveling processes of the storage media controller to optimize the performance of the disk. TRIM is sent by the file system to tell the controller that a block of data is no longer needed and can be erased when appropriate [101]. VirtualBox allows a guest OS to send TRIM commands, which are forwarded to the physical disk. However, this is only applicable to dynamic virtual disks. A static virtual disk does not implement TRIM, even if the required flags `--nonrotational` and `--discard` are used, since its size is static and cannot be shrunken [102–104].

2.9. Maps and Their Sciences

Maps are used in a wide range of situations, from mushroom picking excursions to large military battles. They contain a large amount of valuable information and also have an information linking function, creating new (aggregated) information. Traditionally maps are printed on paper, but digital maps are rapidly taking over [105].

A map has two main features: position and its attributes. The position is fundamental and orientates the projection in space. The attributes of the position are then added. These can for example be height above sea level, terrain type, vegetation, population, or climate type. The oldest verified maps date back to the Babylonian era. What is believed to be a hunting map has been found engraved on a mammoth tusk, which is dated to 25,000 BC [106].

The topography field studies the surface of objects. The hills and depressions of land are together called reliefs. The reliefs are sub-divided based on physical features such as height, form, size, slope, etcetera. In maps the height of a piece of land is often depicted using contour lines [107].

A cartographer maps the reliefs and features of a space as accurate as possible. Usually cartographers create maps of the Earth, but any space (virtual or real) is possible to map [108].

The geological stratigraphy field studies the history of our planet through the layers within its crust. The field is currently divided into eight groups based on the speed of creation of different layers. These time intervals can vary from a few seconds to 10^8 years. Traditionally stratigraphy deals with structures that are formed in 10^2 – 10^7 years [109].

3. Related Work

We have not been able to find any other work directly matching the PhD project, i. e. using the allocation activity of a generic NTFS partition to predict the probability of finding (user) data in different areas of a real NTFS partition (see Section 3.5). This chapter therefore presents a selection of research fields in separate sections. The selection is based on the main contributions of our work. To expand the search we have also looked outside of the digital forensics field. Therefore for example the Usenix File and Storage Technologies (FAST) conference has been scanned for relevant work on file allocation, but none has been found.

The set of articles presented in each section is not exhaustive. The articles are chosen to give an overview of the relevant research fields. The file fragment carving section starts with a summary of the work done during the licentiate studies [39, 110–114].

The related work within the file carving research field can be categorized in different ways (see Appendix E.7). We therefore have chosen to use a simple division into file fragment carving and hash-based carving and present the work in chronological order. This is done to help the reader get an overview of the development of the research fields. A scientific categorization of the articles based on a taxonomy by Poisel et al. [115] can be seen in Table 3.1. The authors have divided the research field into five classes:

Signature-based approaches build on the concept of magic numbers, i. e. specific byte sequences in the header and footer of files. For example a JPEG file starts with `0xFFD8` and ends with `0xFFD9`. The category also include methods from the hash-based carving field.

Statistical approaches utilize different statistical metrics for classification, for example entropy or the Byte Frequency Distribution (BFD).

Computational intelligence based approaches use machine learning and artificial intelligence for classification. Typically algorithms such as k-Nearest Neighbor and Support Vector Machine (SVM) are used.

Approaches considering the context use information from surrounding fragments for classification. The method is based on the assumption that the external fragmentation is low in most file systems.

3. Related Work

Other approaches contain for example methods to visually separate data types and methods based on combinations of the other approaches.

The category *signature-based approaches* in the taxonomy by Poisel et al. corresponds to the *hash-based carving* category used by us. Our *file fragment carving* category corresponds to a combination of the *statistical approaches* and *computational intelligence based approaches* in the taxonomy. Please observe that the work by Ali and Mohamad [132] contains components from the signature-based, statistical and computational intelligence based approaches in the taxonomy and therefore is categorized as *other approaches*. We categorize the work as file-fragment carving due to its direct use of fragment data.

3.1. File Fragment Carving

The section first presents the research work done during the licentiate studies. Then the rest of the related work within the file fragment carving field is presented.

3.1.1. The Licentiate Thesis Work

During the licentiate studies different features for data type categorization were studied. The goal was to find simple, fast and accurate algorithms suited for high entropy data. The main part of the work was done on Joint Photographic Experts Group (JPEG) files.

The evaluated features are histograms of single and byte pairs respectively (the BFD and 2-gram algorithms). Also the rate of change between bytes in a data block is included (the Rate of Change (RoC) algorithm) [39, 110–114]. All algorithms work on NTFS clusters (4 KiB blocks of data). In this way the risk of getting two or more file types in one data block is decreased. However, remnants of deleted file data at the end of an NTFS cluster might still confuse the algorithms.

The 2-gram algorithm has the highest detection rate for JPEG data fragments. The algorithm creates a histogram by moving a two byte long sliding window one byte at a time over the data block. The algorithm is trained on 1 MiB data blocks to compensate for the large set of possible byte pairs ($(2^8)^2 = 65,536$). The values are then scaled to fit a 4 KiB data block.

The 2-gram algorithm is the most complex of the three. It is suitable for structured file types where specific byte combinations should be present or missing in a file. The BFD and RoC algorithms handle eight times smaller data blocks than the 2-gram algorithm. The BFD is insensitive to byte order. It can classify data types where only a subset of the byte values are of importance. The RoC algorithm incorporates both the order and value of the bytes. It is suitable for data types with causal structures, rather than specific value sequences.

Table 3.1.: The articles in Sections 3.1 and 3.2 structured in accordance with the taxonomy proposed by Poisel et al. [115].

Category	Articles
Signature-based approaches	Garfinkel and McCarrin [32] Garfinkel et al. [37] Tridgell [116] Kornblum [117] Dandass et al. [118] Collange et al. [38] Foster [36] Young et al. [35] Canceill [119] Taguchi [120] Hirano et al. [121] Gutierrez-Villarreal [122] Roussev and Garfinkel [123] Garcia [124]
Statistical approaches	Karresand [39] Karresand and Shahmehri [110] Karresand and Shahmehri [111] Karresand and Shahmehri [112] Karresand and Shahmehri [113] Karresand and Shahmehri [114] Veenman [125] Calhoun and Coles [126] Ahmed et al. [127]
Computational intelligence based approaches	Li et al. [128] Fitzgerald et al. [129] Bhatt et al. [130] Bhat et al. [131]
Approaches considering the context	
Other approaches	Ali and Mohamad [132]

3. Related Work

The three algorithms were tested in different settings to evaluate their performance regarding fragment identification. The algorithms were also used to detect the camera make and model of a JPEG data fragment, although with poor results. The licentiate project showed that it is possible to identify and reassemble fragments of JPEG images containing restart markers with simple means.

3.1.2. Other Fragment Carving Research

Veenman [125] use the entropy of data, histograms and Kolmogorov complexity of 4 KiB file fragments to determine their type. The result show that histograms have the highest detection rate versus false positives of the chosen algorithms.

Calhoun and Coles [126] compare different statistical metrics (the frequency of ASCII codes, entropy, mode, mean, standard deviation and correlation between adjacent bytes) in file carving situations. The authors also classify data based on the longest common sub-strings and sub-sequences between fragments.

Ahmed et al. [127] use the byte frequency distribution together with the cosine similarity metric for data classification. Their result is improved relative the use of the Mahalanobis distance metric.

Li et al. [128] use a SVM in combination with the byte frequency distribution to classify data. The authors find that the best results are achieved using the byte frequency distribution alone.

Fitzgerald et al. [129] use combinations of statistical metrics (for example histograms of one and two byte sequences, entropy and Kolmogorov complexity) to create feature vectors. These are fed into a SVM for classification. The authors' method outperforms many previous methods. However, they do not evaluate the contribution of each of the chosen feature vectors, but instead leave it as future work.

Bhatt et al. [130] use SVMs in a hierarchical structure to improve the efficiency of file fragment classification. The SVMs at the top level are trained on general file types. The specificity then increases towards the bottom. The best parameter is the mean of the BFD, twice as important as the length of streaks of equal bytes, which comes second. The principle achieves good results for .csv, .png, .swf, .txt and .xml files. The worst results are achieved for .doc, .jpg, .pdf and .ppt files [130].

Bhat et al. [131] use histograms of byte pairs (2-grams) when carving files. Their framework use a Feed-Forward Neural Network (FNN) to classify the fragments. A Convolutional Neural Network (CNN) is shown to give worse results than an FNN. The authors also test the framework using single byte histograms, which are not as good as the byte pairs. The result show that the latter reach an accuracy of $\geq 89\%$ for all file types except .xls and .doc [131].

Ali and Mohamad [132] use a combination of structure (signature) and content (statistical) based approaches to reassemble file fragments. To extract the correct frag-

ments a combination of entropy, BFD and RoC is used together with an Extreme Learning Machine (ELM). The classification accuracy is > 90% for both the JPEG image databases used (Digital Forensic Research Workshop (DFRWS) 2006 and 2007). Half of the images from the DFRWS 2006 database were completely recovered and the rest were partially recovered. From the DFRWS 2007 database one image was completely recovered, eleven partially recovered and one unreadable [132].

3.2. Hash-Based Carving

Hash-based carving compares hashes of known file blocks and blocks from a suspects disk. In that way also partially overwritten or damaged files can be identified. The process is easily parallelized by dividing the storage media into chunks that are handled independently on separate hardware.

A problem with hash-based carving is the tremendous amount of comparisons to make. However, most of the calculations can be done in advance. The hashes of the known file fragments only have to be calculated once. The hash and search algorithms are also highly optimized and execute fast. Furthermore, a hash value is much smaller than the corresponding raw data¹. This effectively decreases the amount of data to compare. However, there is a theoretical risk of hash collisions decreasing the reliability of the result [32–38].

Comparing small parts of many files to every data block of a storage media is time consuming. Therefore sampling is sometimes used. The sampling frequency (the detection rate) is balanced with the block size, probable file sizes and the time taken by the comparisons. A well balanced search will quickly find a suspicious file with high probability, even if only one fragment has been found [32–38].

During the DFRWS 2006 Carving Challenge [133] Garfinkel [32] was one of the first to use hashes for file carving purposes. By using hashes of parts of files from the internet he could find equal hashes in the disputed image.

Garfinkel's solution to the DFRWS 2006 challenge lead to the development of the `frag_find` tool by Garfinkel et al. [37]. The authors discuss the optimal size of the data blocks to hash. They furthermore elaborate on the suitable size of hashed blocks, reaching the conclusion that 4 KiB blocks are the best [37]. Since the introduction of Windows NT 4.0 the default minimum allocation unit in NTFS has been 4 KiB [48, 134].

Hashing and the comparison of file fragments can be traced back to the `spansum` tool created by Tridgell [116]. The `spansum` tool inspired Kornblum [117] to study

¹In reality the smallest amount of raw data to hash is one sector of 512 bytes, which is converted to a hash value of (currently) at most 512 bits (SHA3-512). Hence the data are compressed at least by a factor of eight.

3. Related Work

piecewise hashing and what is now known as *approximate matching*. Dandass et al. [118] further studied the concept of using hashes for file carving through an empirical analysis of disk sector hashes. Collange et al. [38] introduced the term hash-based carving and used a Graphics Processing Unit (GPU) to compare the hashes.

Foster [36] studies the file carving problem of data shared across files. She states that “the block of NULs is the most common block in our corpus” [36, p. 15], relating them to the NULL padding of files. The large amount of data to handle is also discussed. Young et al. [35] extend the research from Foster’s article. Young et al. discuss the optimal block size, how to handle a large amount of data, efficient hash algorithms, good data sets to use for training and evaluation, and the problem of common file blocks.

Random sampling can be used to improve the speed of hash-based carving [32, 36, 37]. A high sampling frequency increase the detection rate, but decrease the execution speed. By regarding the problem as sampling without replacement a balance can be found [119]. Canceill [119] study the parameters affecting random sampling. Apart from the obvious external parameters disk capacity, sector size and amount of stored data, the detection speed and rate are affected by the sampling block size, transaction size and amount of transactions to check, according to Canceill. Taguchi [120] finds that using 64 KiB data blocks is the fastest way to reach 90% confidence. However, the value might change when the experiments are extended, according to Taguchi.

Hirano et al. [121] study the amount of unique 512 byte hashes in NTFS (Microsoft Windows 8.1), Hierarchical File System Plus (HFS+) (MacOS X 10.9) and fourth extended filesystem (EXT4) (CentOS 6.5). They find that there are 51% distinct sectors in the NTFS partition. The HFS+ and EXT4 partitions contain 88.5% and 84.7% distinct sectors respectively. Hirano et al. also find that there are 43.7% distinct files in the NTFS partition. The HFS+ partition contains 96.5% distinct files and the EXT4 partition 94.3%.

Gutierrez-Villarreal [122] improves the filtering of data blocks with equal hashes found in unrelated files (non-probative blocks). He specifically studies the *ad hoc* rules created by Garfinkel and McCarrin [32]. The rules cover histograms of 4 byte values, the treatment of white-spaces, and ramping behavior in data sequences. Gutierrez-Villarreal improves the performance by replacing the three rules with one. This is achieved by exchanging the 4-byte histograms with 2-byte histograms.

Gutierrez-Villarreal also finds that an entropy threshold can be used to detect for example JPEG files. Furthermore, the same 131 byte large sequence (with different alignment) is found in 10 out of the top 50 block matches in the database. Consequently general filtering rules do not work, they have to be tailored for each specific file type.

The findings on tailored filtering to detect specific file types by Gutierrez-Villarreal support the result by Roussev and Garfinkel [123]. They find that a number of file types are too complex to be possible to detect using a generic method. Roussev and

Garfinkel therefore study the structure of zlib, jpeg and mp3 compressed files and create successful detectors for them [123].

Garcia [124] use five digital forensic file data sets to study the occurrence of hash collisions between fragments of $size = 2^x; x = [4, 5, 6, \dots, 12]$. Two of the data sets contain JPEG files. These files are also compressed using zip to study if the hash collisions are transferred during compression. The files are furthermore stripped of their headers to allow only the image data parts of the JPEG files to be compared. Garcia also uses rolling hashes of the same sizes to further extend the study.

Garcia's research shows that although the data sets all contain unique files there are hash collisions both within and between the sets. The collisions are also transferred to a high extent when the files are compressed. The collision are especially frequent for fragments ≤ 64 bytes in size. The stripping of JPEG headers almost eradicates the hash collisions. However, when using rolling hashes of ≤ 256 bytes there still are collisions present [124]. Consequently there is a risk of misattribution in hash-based carving.

3.3. Digital Stratigraphy

Digital stratigraphy studies the layering of storage media, like soil strata studied in geology and archaeology. The layers occur when old data are overlaid with new data, but remnants of the old data are (partially) preserved. Together the layers form a pattern that can be used to create casual and temporal time lines. These can be used in digital forensic investigations, information archiving and to recover lost data. Digital stratigraphy is important for the creation of our map. A high number of layers corresponds to a high allocation activity.

Casey [135] introduces the term *stratigraphy* in the digital forensic realm. This is done to show the similarities between the works of an archaeologist, an arson investigator and a digital forensic investigator. All three professions have to work with fragments of an original item, which has deteriorated or been willfully destroyed. However, the two first professions have well established methods and tools to interpret the layers. Consequently the digital forensics field would benefit from adapting and incorporating these methods, according to Casey.

Casey also points out the tools used by an archaeologist to determine a time line of events or items as of special interest to the digital forensics field. He states that

how data is positioned and overlaid on the disk may give a sense of when the document was created. [135, p. 15]

The statement is exemplified by a case where a blackmailer was caught because of this phenomena. Parts of the blackmail letter were found in the slack space of another letter

3. Related Work

received by the blackmailer's bank manager at a later date. Hence data in an older strata could be timestamped with the help of a newer strata [135].

Gladyshev and Patel [136] propose the use of finite state machines to model digital forensic applications and processes. In that way it is possible to verify the soundness of an action and to find a chain of actions leading to a specific state. Likewise the authors write that the finite state machine can be used to backtrack the state of a file system when a file has been deleted. However, they do not present any implementation or test of such a function.

Jun and Guo [137] proposes the addition of a logging mechanism to Unix file systems to create a current view of the stratigraphy of the file system. The model is presented and validated through an example. Their model is also said to give temporal information and casual ordering of the layers for specific files. However, Jun and Guo leave the actual implementation of the model as future work. Furthermore, their model requires an extra functionality added to the file systems in advance.

Casey further develops the concept of digital stratigraphy. He writes that

[s]tratigraphy is the scientific study of layers (a.k.a. strata) in geology and archaeology with the aim of determining the origin, composition, distribution, and time frame of each stratum. Applying this concept to data stored on a disk can be fruitful in some investigations. [138, p. 506]

Casey continues by explaining that if remnants of a file is found in the slack space of another file the first file is supposed to be older than the overwriting file, which is not always true. Different file operations and maintenance of the file system might change the order in which the files were written to disk. For example will a defragmentation operation destroy the relationship [138].

However, a defragmentation operation can also help. Casey [139] presents a case where a forged document was created after the file system was defragmented. However, the forger gave it a date before the defragmentation took place. All files but the forged document were found collected neatly together on disk. This showed that the forged document was created after the defragmentation was run [139].

We can extend the use of the digital stratigraphy concept to also cover the behavior of the allocation algorithm. Casey [138] presents a case where one contiguously written file is surrounded by fragments of another file. Due to the behavior of the allocation algorithm the contiguous file is likely to be written before the fragmented file, according to Casey.

Darnowski and Chojnacki [140] backtrack the allocation history in a hypothetical NTFS partition 10 blocks in size. They show how the layering of data can be used to detect the previously allocated data blocks of a now deleted file. This is done by reversing the latest steps taken by the allocation algorithm. The authors conclude by stating that they

hold a view that under certain assumptions it is possible to use NTFS file allocation algorithm for efficient data recovery. [140, p. 39]

Casey [6] describes the next fit allocation strategy (called *next-available* in the article) used in for example the Microsoft Windows FAT12, FAT16, FAT32 file systems. He also covers the best fit allocation algorithm used in NTFS. Casey points out that the best fit strategy might allocate later parts of a file to lower LPVAs than earlier parts of the file.

Casey focuses on NTFS's behavior regarding valid data length (VDL) slack. This type of slack occurs when the file system has allocated more space to a file than is actually required [6]. This typically occurs when a file is not properly saved to disk after being allocated space in the file system. Unused space within the file's allocated space is still regarded as file data. The space is therefore protected from being reallocated and might even survive deletion of the file itself. VDL slack is currently not properly handled by the digital forensic tools, according to Casey [6].

Casey [6] also deals with file tunneling, which occurs in NTFS when a newly deleted file is replaced by a new file with the same name. The metadata of the old file are retained and used for the new file instead. This causes the new content of the file to be seemingly backdated. The file tunneling phenomena might occur regardless of the source of the new file, according to Casey.

Harfield and Schofield [141] discuss the connection between archaeology and digital forensics. Digital stratigraphy can be used to study malware development as well as generally improve the cyber security of systems. However, digital stratigraphy is neither defined nor explained by Harfield and Schofield. The article is written from an archaeological point of view and treats the problem on a high level. The authors argue that the field currently is focused on the technological side of the problem and would benefit from a more heuristic (archaeological) point of view, where also the social side of crime is incorporated [141].

Owens and Padilla [5] focus on the new challenges that meet archivists that work with digital media and web content. New dynamic web technologies serve individualized content to the visitor. Hence there are continuously different versions of the information available at the same time. Owens and Padilla urge the archivists to be as thorough when working with digital material, as with analogue material. The rapid changes and concurrent versions of information challenge the historians and archivists to make sense of the complex stratigraphy in digital sources [5].

3.3.1. Digital Archaeology

Ross and Gow [142] introduce the term digital archaeology in the late 1990's. The authors focus on data recovery and argue for more research efforts put into the field.

3. Related Work

They also see the need for an index of storage media quality and collection of best practice information. Ross and Gow also recommend archivists, librarians and information scientists to look outside of their standard sources of scientific news. In that way they will gain knowledge on media investigations and durability of storage media [142].

Farmer and Venema [143] give a definition of digital archaeology, stating that

[d]igital archaeology is about the direct effects from user activity, such as file contents, file access time stamps, information from deleted files, and network flow logs.[143, p. 13]

They argue that digital information is destroyed by automatic processes run by the OS in the same way as geological effects are destroying archaeological remains.

Graves [144] also discuss digital archaeology. The term is used by the authors as a synonym for file carving and also to generally refer to the digital forensics field. Graves recommends the concepts of archaeology to be applied in digital forensics. The proposed concepts are not revolutionary from a digital forensics point of view. The term digital archaeology could easily be exchanged with digital forensics and file carving where applicable.

Pollitt [145] argues for the exchange of the term digital archaeology (and digital geology, see Section 3.3.2) with the metaphors of history and historiography. He argues for a paradigm shift, because the simple metaphors of archaeology restricts the field. Also the lack of a solid theoretical foundation hinders the development, according to Pollitt. He instead proposes the use of hermeneutic and narrative theories within the domain.

The meaning of digital archaeology seems to change into the opposite meaning in the mid 2010's. The shift happens when archaeologists start to use the term to describe the introduction of digital tools into their own domain. We have not found any strict digital forensic use of the term after that.

3.3.2. Digital Geology

The term digital geology is rarely seen within the digital forensics domain. It is used by Farmer and Venema [143] to state that

[d]igital geology is about autonomous processes that users have no direct control over, such as the allocation and recycling of disk blocks, file ID numbers, memory pages, or process ID numbers.[143, p. 13]

Pollitt [145] also uses the term in a direct comment on the work of Farmer and Venema.

The term seems to disappear from the digital forensics domain sometime in the mid 2010's. Unlike digital archaeology the term is not reused with a new meaning.

3.4. NTFS Fragmentation

Smith and Seltzer [146] study the problem of achieving realistic file system aging in experimental environments. Their method is said to be generically applicable, but is only validated on the UNIX Fast File System (FFS). The proposed method utilizes snapshots from real file systems collected during several months or even years of activity. Sequences of file operations are interpolated between the snapshots. Then an ad hoc amount of short lived files are added. In that way Smith and Seltzer try to mimic the fragmentation occurring in live file systems. After simulating a seven months long usage period the proposed method underestimates the fragmentation rate by 7%, according to the authors. They also write that

[d]ecisions that a file system makes today (for example, which blocks to allocate to a new file) may affect the file system for months or years into the future. [146, p. 204]

Douceur and Bolosky [147] present a study of different statistical metrics of file system features, among them NTFS. Approximately 5.7 TB out of 11.1 TB of their data belong to files from NTFS partitions. The result is however shown as an aggregation of all file systems. It shows that the median file size is 4 kB, the median age of a file is 48 days and that the file systems on average is half full [147].

Garfinkel [148] studies the problem of fragmentation from a file carving point of view. The author uses the Garfinkel corpus². He finds 2,143,553 complete files in 324 disks. Of these files 594,237 come from NTFS partitions and 72,574 (12.2%) of the files are fragmented [148]. The included NTFS partitions contain 590 files (0.1%) that are split into more than 1,000 fragments [148].

We have calculated the average number of files per fragmentation rate interval for the NTFS files based on Table 2 in Garfinkel's article [148, p. S4]. Both the number of files per fragment interval, taken from the article, and the average number of files per specific fragment rate are shown in Table 3.2.

As can be seen in the *Avg files./interval* column in Table 3.2 the distribution of files with a specific number of fragments is continuously decreasing. This is an indication of the allocation algorithm trying to avoid fragmentation. The behavior is not mentioned in Garfinkel's article.

Garfinkel furthermore reports that certain file types are more prone to fragmentation than other. Email, media and certain system files related to logging are usually heavily fragmented. However, the highest fragmentation rates are found in system `.dll` and `.cab` files, which Garfinkel thinks is caused by system patches and upgrades.

²The Garfinkel corpus is said to originally contain 1,005 disk drives. Of them 750 are used in a survey of credit card numbers. The disks are bought on the second hand market [149].

3. Related Work

Table 3.2.: The number of files per fragment interval, taken from the NTFS column in Table 2 in [148], and a calculation of the average number of files per specific fragment rate.

# Fragments	Files	Avg # files./frag. rate
0	521,663	521,663
2	22,984	22,984
3	6,474	6,474
4	3,653	3,653
5–10	13,139	2,190
11–20	7,880	788
21–100	11,901	149
101–1,000	5,953	6.6
1,001–	590	NA

Garfinkel also checks the size of the gap between fragments for bifragmented files. The gap sizes correspond to the smallest allocation block size of FAT and NTFS in almost all cases. The author observes that

fragmentation does appear to go down as drive size increases, but that many large drives have significant amounts of fragmentation. [148, p. S6]

A large drive, i. e. disk, is defined as > 20 GB by Garfinkel.

Cohen [150] describes the fragmentation phenomenon mathematically. This is done by dividing a file into blocks of the same size s as the sectors of its disk. If the file is traversed sequentially position x in the file corresponds to a position y in the sequence of its allocated disk sectors, i. e. $x \pmod s = y \pmod s$ [150, p. 122]. The property means that a file is stored in correct order within the set of allocated disk sectors. However, there is no requirement for the allocation sequence to be ordered. The property fits any allocation algorithm, as long as the file is divided into equally sized blocks during allocation and the allocation is started from the beginning of the file. All allocation strategies described in Section 2.4 fits that requirement, but have extra requirements on the ordering of the allocated sectors.

The now discontinued tool PassMark Fragger [151] can be used to fragment individual files in a controlled manner. It uses standard file defragmentation Application Programming Interfaces (APIs) and therefore should be safe to use. According to the developer it is possible to control both the size of the fragments, the number of fragments, as well as the positions of the fragments [151].

Meyer and Bolosky [152] study space saving through deduplication. They find that 4% of the files from a series of real NTFS computer disks are fragmented. This is

less than earlier reported by Garfinkel [148]. However, 25% of the files contain more than 170 fragments. This indicates a low overall fragmentation rate, but a high rate for individual files. The highly fragmented files appear to mostly be log files, according to Meyer and Bolosky.

Pahade et al. [153] survey the available methods for multimedia file carving. Such files are extra prone to fragmentation due to their larger than average size. The authors focus on evaluating the performance of the surveyed algorithms and do not go deeper into the fragmentation behavior of the file system. Pahade et al. use the PassMark Fragger tool [151] on a 2 GB NTFS partition and create different fragmentation patterns in the form of

- non-fragmented files,
- linear fragmentation, where fragments of different files are concatenated,
- first fit fragmentation, where fragments of different files are positioned according to the first fit allocation strategy,
- scattered files, where the file fragments are linearly allocated in accordance with the layout used in RAM,
- random fragment files, where the file fragments are randomly allocated in accordance with the layout used in RAM.

Apart from the different fragmentation patterns presented, Pahade et al. [153] do not bring any new information on the matter.

Scanlon [1] proposes the use of deduplication to speed the digital forensic process up. The more data put into the deduplication database, the higher the performance increase. The proposed system increases the efficiency of the acquisition process by hashing only parts of data blocks to detect duplication, instead of full blocks as usually done. The analysis phase is made more efficient by centralization of the deduplication database. By only having to read and analyze each new data block once the workload is reduced for all involved law enforcement agencies [1].

Bahjat and Jones [84] study the temporal information hidden in the allocation pattern of NTFS. A time frame for the life of a deleted file is found by using the timestamps of existing files with fragments neighboring the deleted file's fragments. The accuracy of the method is given as 80% confidence within ± 20 days for the creation date and 90% confidence within ± 5 days for the deletion date. The accuracy is negatively affected by heavy use of the file system, defragmentation and the type of file system, according to Bahjat and Jones [84].

van der Meer et al. [154] study the problem of collecting real-world file system data from users in a privacy preserving way. The authors also provide new information on

3. Related Work

the fragmentation rate of NTFS. To do this they collect fragmentation data from 220 privately owned real-world laptop computers together having 334 disks.

According to van der Meer et al. there are four possible fragmentation types. Our interpretation of the fragmentation types is given in the below list. The principle behind each fragmentation type is illustrated using the form [A,B,C,D,X], which is inspired by Figure 1 [154, p. 2] in the article. The characters (data blocks) A to D belong to the same file and X belongs to another file, or is unallocated. The fragmentation types are

- without fragmentation [A,B,C,D]
- non-contiguous fragmentation³ [A,X,B,C,D]
- out-of-order fragmentation [D,A,B,C]
- out-of-order and non-contiguous fragmentation [A,X,D,B,C]

van der Meer et al. show that 46% of the fragmented files in the collection are fragmented out-of-order. The distribution of files of in-order and out-of-order fragmentation is given in Table 3.3, inspired by [154, p. 4, Table II]. The values are the same, but the layout is changed to improve the readability.

Table 3.3.: The distribution of the number of fragments for fragmented files [154, p. 4, Table II]. The layout of the table is changed to improve the readability.

# Fragments	In-Order	Out-of-Order	Total
2	41.84	14.92	56.76
3	7.92	10.29	18.21
4	2.15	6.43	8.58
5	0.77	4.25	5.02
6	0.29	2.67	2.96
7	0.13	1.27	1.40
8	0.07	0.90	0.97
9	0.04	0.65	0.69
10	0.04	0.49	0.53
≥ 11	0.34	4.55	4.89

As can be seen in Table 3.3 the amount of out-of-order fragmented files is higher than in-order for files having ≥ 3 fragments. Furthermore, the amount of out-of-order

³This is called *in-order fragmentation* in the latter parts of the article.

fragmented files relative to the in-order is increasing as the number of fragments increases [154]. This is expected. The more fragments, the higher the probability for an out-of-order allocation.

The increase in out-of-order fragments as the number of fragments increases is in line with the best fit allocation strategy. van der Meer et al. validate their out-of-order rates by checking the rates of file systems with more than 10,000 files and find them to be approximately 40%. They also check with two newly created VMs and get out-of-order fragmentation rates of approximately 45% [154].

The fragmentation rate of 2.3% reported by van der Meer et al. is lower than the 6% reported by [148]. Even though the fragmentation rate is lower the actual amount of fragmented data has increased, according to the authors. The reason is the increase in storage media capacity during that period [154].

Cho [155] proposes a method to create a controlled fragmentation of files for experimental (and other) purposes. Judging from the flow chart [155, p. 14] in the article the method first fills a newly created NTFS file system with a controlled number of resident and non-resident files in a controlled number of directories. When the filling phase ends a number of delete, copy and move operations are performed. The last step is to evaluate the method by defragmenting the file system and checking the cluster status to see the result [155]. The method does not seem to allow checking the allocation status after each file operation, but can be used to prepare a file system for further experiments.

van der Meer et al. [53] study the fragmentation rate of 220 real-world laptops, where 84% contain an SSD and 67% an HDD. They together contain 729 NTFS formatted and unencrypted partitions. The study is focused on different fragmentation metrics of NTFS. Of special interest is the out-of-order fragmentation presented in the previous article ([154]). The new article also extends the work from the previous article on a privacy preserving method to study fragmentation in real-world computers. van der Meer et al. also update the results on file fragmentation rates presented by Garfinkel [148].


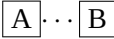

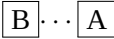
van der Meer et al. [53] update and rename the storage patterns that were previously called fragmentation patterns [154]. The updated patterns can be seen in Table 3.4, which is inspired by Figure 1 in [53, p. 2]. The definitions of the storage patterns are similar, but they are now more stringent.

van der Meer et al. [53] describe two different types of file system fragmentation. *Fragmentation of free space* refers to the remnants of free space left after a non-fitting allocation. *File fragmentation* refers to the traditional definition of file fragmentation, i. e. external fragmentation. The fragmentation types are used in the definition of two fragmentation metrics. The authors first define the *degree of fragmentation* as:

definition 1 (*degree of fragmentation*). *The degree of fragmented files is the number*

3. Related Work

Table 3.4.: The four types of storage patterns of a file as given by van der Meer et al. in Figure 1 in [53, p. 2]. The letters A and B depict non-overlapping data blocks of a file, where A starts at a lower LPVA than B. Please observe that an in-order contiguously stored file is not fragmented.

	Contiguous	Non-contiguous
In-order		
Out-of-order		

of fragmented files divided by the total number of files. The total number of files [is] defined as:

- I. all MFT entries, OR
- II. all MFT entries with data, OR
- III. all MFT entries with blocks assigned, OR
- IV. all MFT entries with ≥ 2 blocks assigned. [53, Definition 1]

van der Meer et al. prefer the last alternative (IV) for the total number of files, because that is the most restrictive setting. By using that definition the degree of fragmentation will never be underestimated [53]. They then define the *percentage of internal fragmentation* of a fragmented file as:

definition 2 (% of internal fragmentation). The percentage of internal fragmentation of a file f of at least 2 blocks is [the] ratio of the number of fragmentation points vs. the number of blocks minus one, i.e.:

$$\text{intfrag}(f) = \frac{\text{fragpoints}(f)}{\text{blocks}(f) - 1} \cdot 100,$$

where $\text{blocks}(f)$ denotes the total number of blocks of file f , and $\text{fragpoints}(f)$ is the number of times where, when reading a block of file f , the next block of f is not the next block on disk. [53, Definition 2]

Finally the authors define the *percentage of out-of-order'ness* of a fragmented file as:

definition 3 (% of OoO'ness). The percentage of out-of-order'ness of a fragmented file (f) is the ratio of the number of times the next fragment occurs prior to the current vs. the total number of fragmentation points, i.e.:

$$\text{OoO}'\text{ness}(f) = \frac{\text{backfragpoints}(f)}{\text{fragpoint}(f)} \cdot 100,$$

with $\text{fragpoint}(f)$ defined as before, and where $\text{backfragpoints}(f)$ denotes the number of times the next block of file f is stored earlier on disk than the current block. [53, Definition 3]

van der Meer et al. find that 74% of the files at least two blocks in size are smaller than 100 kB. They also find that 75% of all files 1 MB–100 MB in size are fragmented out-of-order. There is also a correlation between file size and fragmentation rate, although not perfect. The average OoO'ness is $\frac{1}{3}$ for files > 50 kB in size. A number of files are extremely fragmented, having up to 20,000 fragments [53].

van der Meer et al. furthermore study fragmentation among different file types, such as videos, images, documents and databases. They find that images are often fragmented out-of-order (with exception for `.bmp`, `.png` and `.raw`). For video files (`.avi`) the fragmentation rate is low (1.8%), but a few `.avi` files are heavily fragmented. Among the document files `.pdf` files have a higher fragmentation rate than standard word processor files like `.odt` and `.docx`. van der Meer et al. furthermore report that all databases are fragmented above average. They also find that generally 56.8% of the files are split in two fragments and that 41.8% of all fragmented files are fragmented in-order [53].

van der Meer et al. also study the size of the gaps in bi-fragmented files. To measure the gaps three different distance metrics are used. For in-order files the *carving distance* is found to better show the peaks in number of spaces at power of two distances⁴. The start of the first power of two gap is determined by the size of the first fragment, which is in line with the NTFS allocation algorithm, according to van der Meer et al. The general trend for gap sizes is however a decrease in size as the number of fragments increases [53].

Furthermore, van der Meer et al. write that the correlation between the amount of fragmentation and the utilization level of NTFS is 0.46 for both SSDs and HDDs. Hence also less utilized partitions might be heavily fragmented, and vice versa.

Finally van der Meer et al. find that the top ten unfragmented file types are system files that do not contain any user data. Also the most fragmented file types (≥ 1000 fragments) are system files with extensions such as `exe`, `log`, `xml`, `dat` and `dll` [53].

3.5. NTFS Data Allocation Process

The data allocation algorithm used by Microsoft Windows in combination with NTFS is governing the placement of data at specific LPVA positions. Hence the allocation algorithm is of key importance to the creation of a map of user data, because a good

⁴A power of two distance d is defined as $d = 2^n$, where $n \in \mathbb{Z}^+$ and is measured in blocks (a block corresponds to a Microsoft Windows cluster of 4 KiB) [53].

3. Related Work

understanding of the algorithm both increase the resolution of the map and also enables a digital forensic investigator to understand the process that lead to the current state of the file system. In that way it might (as suggested in a number of articles) be possible to backtrack the state of the file system and make predictions on the casual ordering of files.

Willassen [156] describes a method using a first fit allocation strategy to detect antedating of files. To do that a tool is built based on the method and then validated by four subjects with different levels of computer knowledge. Two of the subjects are ordinary computer users, one is a digital forensic investigator and one is an advanced computer user with programming skills. None of the subjects managed to avoid detection of their file antedating attempts [156].

Willassen concludes the article by presenting different methods that can be used to create antedating not detectable by the tool, but also writes that the methods are highly specialized and not applicable in real life. He also uses Locard's exchange principle [157–161] to argue that there will be patterns left by manipulation that can be detected [156].

A PhD dissertation by Willassen [162] presents the formal foundation of timestamps and methods to improve their applicability in digital forensics. The author also formulates the criteria needed to use the allocation strategy of a file system for checking timestamps. Furthermore, he briefly discusses the allocation strategy of NTFS and states that it is best fit in Windows XP and that a first fit allocation strategy is used within the MFT [162].

Willassen's dissertation also contains a short section on the impossibility to use the allocation pattern alone for time stamping of files, it is necessary to use an extra, external, source of information in combination with the allocation pattern [162].

Darnowski and Chojnacki [77] study the allocation algorithms used for data block allocation, as well as for allocating MFT records. Information on the two allocation algorithms is found through a series of experiments on the writing behavior of NTFS. Differently sized files are written and deleted in predetermined patterns and the effect on the NTFS file system on a 4 GB USB thumb drive is analyzed using the WinHex tool. The data allocation strategy is found to be best fit and the MFT allocation strategy is shown to be first fit (called First-Free Algorithm (FFA) in the article) [77].

All experiments in the article by Darnowski and Chojnacki are done as block writing. However, they also shortly explain the behavior of the allocation algorithm during stream writing, where the size of each written block of data is said to increase up to a preset limit. The authors briefly mention the idea of backtracking a NTFS file system from its current state using knowledge on the allocation algorithm's behavior [77].

Chojnacki and Darnowski [163] continue the work presented in their earlier article ([77]) by creating a Finite-State Machine that mimics the data allocation behavior of NTFS during block writing. The authors argue that the model can be used to backtrack

the allocation behavior of a NTFS partition using additional information on the deleted files. The model is able to exclude any data blocks that belong to the OS from the analysis [163].

Bouma [164] presents a study of the effect of file operations on the timestamps in NTFS in a Bachelor's thesis from 2019. The work is mainly focused on how the MFT is affected by the file operations, which includes how the allocation process of the MFT is working. According to the author a first fit strategy is used and if there is no free area to be allocated to a new MFT entry the \$MFT file is extended by 256 KiB [164].

Furthermore, the method used to increment the 2 byte sequence numbers in the MFT records (described by Carrier [50]) is wrong. Bouma has found that the number only is incremented when a file is marked as deleted, not each time a file is allocated to an entry [164].

Ghotge and Nema [165] from Microsoft published a white paper in 2004 describing the pre-allocation process of NTFS. The whitepaper covers the allocation behavior during stream writing⁵ according to Darnowski and Chojnacki [77] and Chojnacki and Darnowski [163]. As the title of the article indicates the content seems to be directed to the pre-allocation functionality of NTFS, where files of unknown size (during stream writing) are to be written to disk. NTFS reserves space for the file in advance and then fills the space as long as the stream is active (until the file ends) [77, 163].

In the Linux ext3 file system pre-allocation is used to reduce fragmentation and lower the allocation work load on the system by doing fewer allocations of larger blocks at a time, according to Silberschatz et al. [58]. The same reasons might be valid for NTFS too.

3.6. Data Recovery

Data recovery [166, 167] is a multi-disciplinary application field closely related to file carving. The field also uses information, methods and tools from other file system and digital storage related areas when needed. The difference to file carving is the scope of the recovery. The focus in file carving is on single files or types of files, with a strict requirement on a forensically sound process⁶. Data recovery aims at salvaging as much data as possible and can accept that data might be changed or otherwise affected during the process.

Data recovery is mostly applied outside of the digital forensics domain. It is used by archivists, librarians and any other profession handling data that might be lost. There

⁵We have not been able to find the publication and therefore have to rely on second hand information, thus the real reason for the stream writing allocation behavior in Microsoft NTFS is unknown to us.

⁶A forensically sound process is defined as “[t]he application of a transparent digital forensic process that preserves the original meaning of the data for production in a court of law.” [168, p. 10]

3. Related Work

are also both commercial and open source software that can be used to recover data from broken storage media, even by individuals and non-professionals. Many of the tools are also used within the file carving field. Table 3.5 shows typical examples of tools [169–171] used in both fields.

Table 3.5.: Typical tools used in both file carving and data recovery. The selection is made using our own preferences and internet lists of the “top xx tools” in data recovery. The table is not exhaustive.

Name	OS
Mondo Rescue [172]	Linux
GNU ddrescue [173]	Linux
foremost [174]	Linux
safecopy [175]	Linux
extundelete [176]	Linux
testdisk [177, 178]	Windows, Linux, Mac
photorec [179]	Windows, Linux, Mac
scalpel [180, 181]	Windows, Linux, Mac
EaseUS Data Recovery Wizard [182]	Windows, Mac
Disk Drill [183]	Windows, Mac
Recuva [184]	Windows

There are also live Linux distributions⁷ aimed at data recovery containing a suitable set of tools. The distributions can be installed on a USB stick, which then is used to boot a computer that needs repairing. Please observe that this method only allows repairs of logical defects. Many times these distributions are the work of a single enthusiast, which has not the capacity for long term maintenance. The list in Table 3.6 contains a selection of live Linux distributions for data recovery that are still actively maintained (in 2022). The selection is based on material from DistroWatch.com [185].

Owens and Padilla [5], Dietrich and Adelstein [31], Ross and Gow [142], and Farmer and Venema [143] have written articles that relate to data recovery. Due to the multi-disciplinary qualities of the data recovery field the articles have already been presented in other sections.

⁷A live Linux distribution is a fully functional OS held on a CD, USB or other type of storage media. The live distribution can be booted and used without affecting or using any storage media on the booted computer. In that way a computer with broken or corrupt storage media can still be booted and data from the media can then be rescued.

Table 3.6.: A list of actively maintained data recovery live Linux distributions in 2022. The distributions contain a large selection of data recovery (and other digital forensic) tools. The selection is based on material from Distrowatch.com [185].

Name	Main use
Kali [186]	Penetration testing
ALT Linux Rescue [187]	System recovery
Finnix [188]	System recovery
GRML [189]	System recovery
Kaisen Linux [190]	System recovery
System Rescue CD [191]	System recovery
Clonezilla Live [192]	Media recovery
GParted Live [193]	Media recovery
Parted Magic [194]	Media recovery
Plop Linux [195]	Media recovery

3.7. Data Mapping

The mapping of data in storage media is closely related to the subject of this thesis, although the term as it is used in the related work articles comprise a wider meaning than creating a map of the allocation activity of a generic storage media partition. In the articles the term is for example used to describe activities related to finding all fragments of a specific file, the high level activity of scanning the data of a disk and to increase the speed of forensic acquisition and analysis by using the existing file system to determine what LPVA areas to prioritize during the investigation.

Conti et al. [2] present a mapping of binary data in the form of two-dimensional byte maps, where each byte corresponds to a pixel in an image. The mapping can be applied to any binary object to visualize its internal structure, even to full disk images. They also propose that a database of data types (a fragment corpus) should be assembled to enable identification of different data types in the map of a binary object. They restrict the application to single objects, hence their mapping concept cannot be used to predict and plan ahead in general situations, only in each specific case.

Key [196] describe an EnScript module for the EnCase software [197]. The module is used to create a map of the recoverable sectors of a file found in a file system. It is unclear if the module uses any information on the allocation pattern of the file system when creating the map. It can handle situations where other tools do not work, for example partially damaged files. However, the module is very CPU intensive and

3. Related Work

therefore can only create maps of a few files at a time, according to Key Woods and Lee [198] mention that

disk images can be used to produce unified maps or hierarchies of both allocated and unallocated space on the original device. [198, p. 2]

The concept is not further explained and it is unclear whether the maps show a hierarchical view of the file system or simply which (logical) sectors of the storage media that are allocated (or not), in a similar fashion to the \$Bitmap file of NTFS. Woods and Lee use of the plural form of the storage media (disk images). This might indicate that their maps indeed are generic representations of the allocation activity at different logical addresses, making them similar to the mapping concept presented in this thesis.

Beek et al. [199] present the digital forensics as a service (DFaaS) system Hansken and Baar et al. [200] have written about Hansken's predecessor Xiraf. Both Hansken and Xiraf utilize the concept of non-linear extraction of data from images of suspects' disks. The idea is similar to what is presented by Schatz [13] and Schatz [201]. The principle is to first extract the MFT records of a NTFS partition. The records are then used to find other interesting areas of the file system [199].

Beek et al. also suggest that the analysis process is used to influence the imaging process by having specified parts being prioritized. The authors use file names and other higher level metadata found in the MFT records to prioritize [199]. Hence the method is dependent on a working file system and is therefore not file carving in a strict sense.

The linear model used when reading disks during acquisition is limited by the lower writing speed of a disk, especially HDDs. Schatz [13] describes a new approach where the process of acquisition and analysis is non-linear and concurrent using several destination disks. To keep track of what storage media areas have been acquired a map is used. The map is virtual and a feature of the Advanced Forensics Format v.4 (AFF4) forensic file format. The process requires fully readable file systems and working disks and is meant to utilize the full transfer speed of the involved disks and interfaces [13].

The principle with a map showing already acquired data blocks ready for analysis is later on used in a US patent from 2019, also by Schatz [201]. It describes a system for utilizing the full speed of the data transfer interfaces of computer storage for forensic activities [201].

Russinovich [202] presents the `DiskView` tool from the Microsoft Sysinternals suite [203]. The tool can be used to view the allocation status of file data in NTFS. By clicking on a cluster in the view information on the occupying file is given. The tool shows a momentary view of the allocation status of clusters in a storage media (and its partition(s)), but cannot give information on the allocation activity at different positions. The toolkit is currently downloadable for free. There is also an online version available [203]. `DiskView`'s functions are comparable to parts of the `Sleuth Kit` toolkit [204].

Gladyshev and James [11] study file carving from a decision-theoretic point of view. The authors suggest a model where storage media is sampled with a frequency based on different properties of the disk and the file type that is to be found. In some specific situations the carving model outperform standard linear carving algorithms, but the solution is not generally applicable at the time of writing [11]. The model uses the distribution of data on disk (in the relevant partition), but it is not clear whether it takes advantage of any inherent structures introduced by the allocation algorithm.

4. Experimental Setup

As a part of the work to answer the research questions in Section 1.5.2 different experiments were executed. The following sections present the setup for each of the experiments in chronological order.

There are different ways of (technically) naming Windows releases. However, the terminology is not fixed, for example the terms *version* and *build* numbers are sometimes mixed [205, 206]. The precision is also lower when using the (technical) version numbers compared to the product names, for example Windows version 10.0 includes at least five product names (Windows 10 and 11, plus Windows Server 2016, 2019 and 2022) [206]. Although the same OS core might be used within different product names, there can be features enabled in one product that is not included in another. These features might affect our experiments. We therefore have chosen to use the product names in the thesis.

4.1. Motivation

We study NTFS, but the mapping concept is applicable to any combination of OS and file system. However, the research results from the experiments are unique to NTFS and only used as proof-of-concept. They are therefore not directly transferable to any other file system.

The easiest way to study the allocation algorithm of a file system would be to use its source code. However, that is not feasible for (at least) three reasons. First of all the creation of our map should not be dependent on the availability of the source code. That would restrict the mapping concept to a subset of the existing file systems.

Furthermore, the compiled source code should give the same allocation behavior as in a real system. However, we do not know if the behavior will match a real system and the only way of guarantee that is to use a real system. Any deviation from the actual behavior of a real system will possibly degrade the fidelity of the experiment. Consequently the source code path is a detour on the way to the creation of the map.

The source code can also be used to create a model of the allocation behavior. However, there is a risk of introducing errors during the modeling process. That would cause the map to fail meeting

one of the main functions of the map — to provide truthful information. [207, p. 1]

4. Experimental Setup

The best, if not the only, way of reaching the highest fidelity is to study the behavior of the allocation algorithm of a file system as implemented by its founder (Microsoft in our case). And this should be done in settings as close as possible to the real world. The best way is therefore to use a black box scenario, where the effect of the input on the output of the algorithm is studied, nothing else. In that way any implementation issues, bugs and unforeseen interactions of the components of the I/O ecosystem are handled automatically. Hence using the source code is the wrong way to go.

4.2. Static Areas

To be able to determine if there are any static areas present in an NTFS formatted main partition of a standard Microsoft Windows computer we first collect live data from real computers. We then compare the hash values at different LPVA positions in the partitions. This enable us to calculate the probability of finding unique data at different positions in a generic partition. To do this we divide each partition into 128 equally sized areas and calculate the mean probability for each area. Finally the mean of the means for the areas are calculated. The mean probability calculations are done to generalize and scale the map into a usable format.

To protect the privacy of the user we use the Secure Hash Algorithm 1 (SHA-1) algorithm to hash each 512 byte sector of all 30 NTFS formatted main partitions included in the experiment. The hashing of data is done locally at each source computer and thus only the resulting hashes leave the computers. The use of a hash value also decrease the amount of data to handle. We use the SHA-1 algorithm because it currently offers the best balance between speed, collision risk and hash size among a selection of hash algorithms. The choice is based on an empirical evaluation of the hashing speed in modern hardware.

SHA-1 maps 512 bytes of data onto a 20 byte long hash string and thus there is a theoretical risk of collisions. If we apply the Birthday Paradox to our experimental setup, the risk of a collision is approximately $1.1 \cdot 10^{-28}$ and hence negligible. We therefore assume a unique SHA-1 hash represents a unique piece of data.

The theoretical risk of collisions comes from the fact that 512 bytes of data are compressed into a 20 byte long hash and therefore the results might contain false positives. The problem can be viewed as a Birthday Paradox, where N is the number of possible hashes, n is the number of hashes, i. e. the total amount of sectors we have hashed (as a worst case scenario), and $P(Collision)$ the probability of a collision, which can be calculated as

$$P(Collision) = 1 - \frac{N!}{N^n \cdot (N - n)!}$$

and with $N = 2^{160}$ and $n = 18, 210, 308, 798$ the probability of at least one collision

is approximately

$$P(\text{Collision}) \approx 1 - e^{-n^2/2N} \approx n^2/2N \approx 1.1 \cdot 10^{-28}.$$

Our approximation is based on Stirling’s approximation of factorials, which gives acceptable results when dealing with very large numbers. Since the SHAttered [208, 209] attack is 100,000 times faster than a brute force attack using the birthday paradox the risk of an intentional collision is higher, but the attack is unfeasible in our situation.

Even though the SHA-1 algorithm is broken [208, 209] from a cryptographic point of view the risk of an intentional collision is also negligible, because the amount of computing power required to create a collision is out of reach for the common user [208, 209]. Furthermore, such an attack would require an attacker to create a large amount of collisions for a majority of the storage media in the source data of the map. It would be much simpler to fill the disks with shared and unique data in an intentional pattern. This is however mitigated by collecting the source data from non-related sources. Finally the mapping process is not limited to the use of SHA-1, any hashing algorithm will do, as long as all mapping data is hashed using the same algorithm.

4.2.1. Data Collection

A convenience selection is used to collect the live data. We do not use the Real Data Corpus (RDC) because the time stamps on the RDC web site [210] indicate that the last update of the data set is made in 2011. Therefore our data set is more up to date, containing also versions 8 and 10 of Windows¹.

We collect data from 30 partitions of 26 computers (23 consumer grade and 3 office grade). The data are collected using the `dcfldd` disk imaging tool set to log the hash of every 512 byte disk sector to a file. We also retrieve the partition layout of each disk. The included computers use three different language packs (Swedish, Finnish, English) and the OS types range from Microsoft Windows 7 to Windows 10, covering the Enterprise, Professional, Ultimate, Home and Educational products. Some of the computers have been upgraded from an earlier Windows version to Windows 10. Five of the computers are in our possession and we therefore have access to their raw content². The Windows product version of each computer can be seen in Table 4.1.

We use real computers for the experiment to increase its fidelity. The drawback is a lower degree of control of the collected data. In some cases we lack information on whether a disk is an HDD or an SSD. However, we collect the data at the LPVA level and therefore avoid any low level differences [53–57] (see also Section 2.1).

¹Windows 8 was introduced at the end of 2012 [211] and therefore cannot exist in the RDC, neither can Windows 10.

²The raw data images are exempted from sharing according to Findability, Accessibility, Interoperability, and Reusability (FAIR) principles to protect the owners’ privacy.

4. Experimental Setup

Table 4.1.: The OS type and other information on the 26 disks used in the static area experiment. Some computers are upgraded, some are office computers and some contain two partitions that have been used.

File	OS	Comment
A	Win 10 Home	
B	Win 10	Upgr Win 8
C	Win 7 SP 1	
D	Win 10	
E	Win 7 Home Premium	
F	Win 8.1	Storage disk, no OS
G	Win 10 Edu, Build 1709	Office computer
H	Win 10	Probably Home
I	Win 10 Edu	
K	Win 10	Upgr Win 7?
M	Win 7	
N	Win 10	New office computer
O	Win 10	Upgr Win 7?, 2 part
P	Win 7 Pro, Build 7601	
Q	Win 8.1	2 part
R	Win 10	Upgr Win 8, 2 part
S	Win 7 Pro, Build 7601	2 part
T	Win 7 Ultimate, Build 7601	
U	Win 10	Upgr Win 7 Home
V	Win 7 Pro	Sector (0x268C2AEE00) is damaged
W	Win 7 Enterprise	Office computer
X	Win 7 SP1 Pro	
Y	Win 7	
Z	Win 8.1	
AA	Win 10 Home, Build 16299	
AC	Win 10 Pro	

From our point of view the only difference between an HDD and an SSD disk is their filling of unused areas, which can be either old data, 0x00 or 0xFF depending on how the TRIM command is implemented in the SSDs [212–217]. Hence an HDD will more often give us old data from currently unallocated clusters than an SSD. Since the experiment focus on unique data any 0x00 and 0xFF filling is automatically filtered out.

If a large amount of the unallocated sectors contain old and unique data our results will be affected. This will be the case if a disk is erased using a random pattern and then reformatted and reused. For this to happen the scenario shall be true for a significant part of the partitions in our data set. We therefore check with the users if they have done any large file system cleaning close to our data collection.

The sizes of the disks in our experiment range from 64 GB to 1 TB. The largest NTFS formatted partition from each disk is extracted. In four cases there are an extra storage partition present, which is extracted too (see Table 4.1).

To determine whether any of the partitions in our data set have been completely filled with data we study the last 20 GB of each partition. If we find anything but 0x00 or 0xFF filling there we assume the partition has been full at least once during its life time. The size of 20 GB is chosen to be a suitable trade-off between a large enough amount of data and the risk of including the OS area for the smaller partitions.

Each studied partition contain a number of 1 KiB³ MFT records. These records contain for example time stamps, file names and allocation data making them highly unique and therefore affecting our results. We therefore perform a survey on 27 Windows computers not included in our data set to estimate the mean number of MFT records in an NTFS partition. Each file and folder in an NTFS partition is represented by, at least, one MFT record⁴).

4.2.2. Data Analysis

To prepare the data for further analysis we combine the retrieved hash values into a single file, which is then sorted in ascending order of hash value. We then extract the unique hashes from the file, thus any 0x00 and 0xFF filled sectors are automatically filtered out. After the extraction of unique hashes we sort them in order of ascending LPVA position and separated them in individual files based on partition identity. The data for each partition are then divided into 128 equally sized areas. We then calculate the proportion of unique hashes in each area for each partition.

³The size of an MFT record is defined in the boot sector of an NTFS partition. The de facto standard size is 1 KiB [50], but recently also 4 KiB records have been introduced [48]. None of the disks in our experiment uses 4 KiB records.

⁴If a file has many attributes, for example alternate streams or is heavily fragmented, the file system creates a new MFT record to hold the extra information [48].

4. Experimental Setup

The mean, median and standard deviation of the probability for each area in each partition are then calculated. The process enables us to combine the results of the calculations regardless of the differing partition sizes. The resulting values together form a map of the probability to find unique data (user data) in a generic NTFS formatted partition.

4.2.3. Map Evaluation

To evaluate our map we run an experiment simulating a hash-based carving scenario. The idea is to compare the performance of sampling according to our map to a uniform sampling distribution. As ground truth we use partitions not included in our data set. The partitions are extracted from storage media of four old office computers that have been used to handle large amounts of data. The partitions are divided into 128 equally sized areas using the mapping process. We use the distribution of unique data in the four partitions to pick a random integer called *target*. We then use our map to pick a random integer *map* and the uniform distribution to pick a random integer *uni*. All random integers are selected within the same total range representing the LPVA positions of a fictive 16 MiB partition, although with bias for *target* and *map*. If $map = target$ our map gets one hit, if $uni = target$ the uniform distribution gets one hit. The small size is chosen to increase the probability of a hit. The experiment is executed using Python 2.7 and the random library in a Debian Stretch (v. 9) computer.

We iterate the random sampling process 10^9 times for each of the four partitions to stabilize the result. The low number (30) of partitions used to create the map does however affect the evaluation since it is a small population to build a model from. Likewise our set of partitions forming the ground truth is small and the result is therefore affected by any individual variations of the partition content. Another factor affecting the result is the fact that the four partitions used as ground truth are taken from office computers that have been scrapped and therefore have well used disks. They therefore contain a lower amount of 0x00 and 0xFF filling.

4.3. Repeated File Operations

The repeated file operations experiment is based on an iterative process where we randomly create, delete, expand and shrink files in VMs running different Microsoft Windows product versions. The experiment is using 32 VMs in eight nodes in the CRATE cyber range at the Swedish Defence Research Agency (FOI) [218]. The aim is to empirically study how the allocation frequency varies at different LPVA positions and how the file allocation pattern develops over time.

We are given 16 days by FOI to execute the experiment. Most, but not all, of the

VMs will then have completed 10,000 iterations. We are also given the opportunity to run three VMs with larger (256 GiB) virtual disks on three extra nodes in the cyber range.

Due to instability in the VBoxManage interface and unforeseen popup windows appearing in the VMs several of them have to be manually restarted during the course of the experiment. This might affect the result of the experiment, but since there are eight VMs per Windows product version and four of each use the same writing pattern the effects of the unplanned reboots are diminished.

We let 16 of the 32 VMs use exactly the same file operation pattern to test if there is any deterministic behavior connected to the allocation. This is done by searching for similarities in the allocation patterns of the VMs. Hypothetically the allocation patterns should be equal, since the VMs of each Windows version are exact copies of each other. Due to unforeseen behavior (machine hangs during boot, system messages locking the shut down process, etcetera) of the VMs a total of five VMs are disqualified from the similarity test, giving eleven that are possible to compare. However, these VMs are also subject to small disturbances, hence the foundation for the deterministic allocation behavior study is weakened.

4.3.1. Platform

The experiment uses 32 freshly installed VirtualBox v. 5.2.20 VMs running Windows versions 7, 8, 8.1 and 10 with 64 GiB NTFS formatted partitions. The exact Windows product versions can be seen in Table 4.2. Each node in CRATE is running Gentoo Linux 10.1 with kernel 4.18.13. 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 FOI and we are not allowed to make any changes to the host nodes. We therefore cannot install any specialized forensic software, such as the Sleuth Kit [204], on the nodes. The VBoxManage interface is used for the communication between the host and the VM.

Table 4.2.: The four types of Windows used in the repeated file operations experiment.

Product name	Version/Build #
Windows 7 Professional SP 1	7601
Windows 8 Enterprise	9200
Windows 8.1 Enterprise	9600
Windows 10 Home	17763

4. Experimental Setup

We run four VMs in each node, one for each version of Windows in our test (see Table 4.2). The VMs are copied using `scp`, not cloned, between the nodes and hence identical. The success of each copy operation is checked using `sha1sum`.

To enable us to extract the \$Bitmap file after each process iteration the VMs are configured to use fixed size virtual disks. This type of disk is given its full size directly when created. It is therefore unaffected by the virtualization layer of VirtualBox [219, 220]. Consequently the virtual disk files can be handled by standard Linux file carving tools, such as `dd`⁵. We also switch off host caching to decrease the delay induced by double caching when the changes to the virtual disk are written to the physical [219].

The virtual disks are emulating HDDs, which is the default behavior of VirtualBox v. 5.2.20 [221, p. 157]. This also rids us of any TRIM commands from the VMs. In that way only the weekly defragmentation, which is run automatically by Windows, affects the virtual disks. Since the virtual disks are constantly used and the VMs rebooted within a few minutes there is no time for the automatic defragmentation to start. Consequently the disks are more or less unaffected by the defragmentation. Furthermore, we aim for a scenario as close as possible to the real world and use a black box principle for the mapping concept. Consequently the defragmentation process will be a natural part of the life of an NTFS file system and therefore does not interfere with our aim.

The virtual disks are not automatically encrypted by the OS or manually by us, because neither block device encryption, nor stacked file system encryption, affects the logical layer, i. e. the allocation strategy. Both types of encryption act either above or below the file system and data allocation layers (see Figure 1.1). This is valid for NTFS, as well as other file systems [7, 86, 222–224]. Adding encryption to the virtual disks would therefore only put an execution overhead on the experiment, without affecting the data allocation.

We use 64 GiB virtual disks, allowing us to have four VMs in each node and still have space for 40 000 \$Bitmap file copies of 2 MiB each on the host's disk. A 64 GiB disk is small compared to the current standard disks, but still large enough to be found in cheaper or older computers equipped with SSD disks.

An extra experiment is executed to generalize the results. The experiment uses three VMs with 256 GiB disks running Windows 7, 8.1 and 10. Windows 8 is excluded due to its slow power cycle. Each VM is setup in the same way as the 64 GiB VMs and use the same file operation pattern as in the similarity test. The slowest booting VM of the three, Windows 10, only executes 8,331 iterations before we have to stop it. Therefore the results of all VMs are limited to that amount of iterations.

⁵There is a VirtualBox specific header at the beginning of the `.vdi` file containing the virtual disk. This header has to be skipped to reach the actual disk part. The header size is measured in blocks of 1 MiB. Static virtual disks have two blocks large headers [220].

Each VM has Python 2.7 installed together with four Python scripts that create, delete, increase and decrease files. Each VM has a separate virtual disk shared with the host holding the scripts, which therefore do not affect the allocation pattern of the VM's own virtual disk. This configuration is used to isolate the machines from each other and avoid them accessing the same file at the same time. There is also an auto started `.bat` script that writes a file to the shared disk. The file is created each time the VM has finished booting and deleted before next iteration.

4.3.2. Implementation

Each iteration of the experimental process contains the following steps, which are controlled by a Python script on the host node:

1. Boot a VM.
2. Randomly (with bias) either create, delete, expand or shrink a file within the VM's NTFS file system.
3. Shut down the VM.
4. Extract the \$Bitmap file from the virtual disk (using `dd` externally from the host computer)

Since each iteration requires the VM to be rebooted it takes several minutes to complete. There are also extra waiting time inserted at critical moments to compensate for any variations in execution time during an iteration.

The type of file operation executed in the VM is based on a configuration file containing a precomputed weighted (biased) random selection of operations. The bias of each file operation can be seen in Table 4.3. The bias value of an operation is calculated as $bias_{op} = \frac{factor_{op}}{\sum factors}$. The bias values $\{0, 0, 0, 0, 1, 1, 2, 3\}$ will for example give 50% create operations, 25% delete operations and 12.5% increase and decrease operations respectively.

The size of the files operated on is also varied within a size range. The *size factor* in Table 4.3 is multiplied with a random number in the range 1 to *random range*. If the same size factor is given multiple times the probability of its use is increased. Based on the random range 1 to 1,024 the size factors $\{8, 8, 128, 2,048\}$ will generate twice as many files in the size range 8 to 8,192 sectors, as in the ranges 128 to 131,072 and 2,048 to 2,097,152 sectors.

The size and file operation bias factors are used to simulate the behavior of different types of users. 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

4. Experimental Setup

Table 4.3.: The settings used to generate the file operations 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	Identic. behavior	Uniq. behavior
Size factors [512 B]	{8, 2,048}	{8, 8, 128, 2,048}
Writes [$x/40$]	10	10
Deletes [$x/40$]	9	9
Increases [$x/40$]	11	14
Decreases [$x/40$]	10	7
Random range	1,024	1,024
Write {start, stop}	{0.05, 0.3}	{0.05, 0.3}
Delete {start, stop}	{0.95, 0.7}	{0.95, 0.7}
Total size [512 B]	112,000,000	112,000,000

amount of large files and a user storing a large amount of images will probably create mostly small to medium sized files.

There are also limits governing the users behavior when the file system is empty or becomes full. During that period the user will also create and modify different files to a certain degree. The *write start/stop* and *delete start/stop* limits in Table 4.3 is used when the virtual disk is being emptied or filled up. If the current amount of data (controlled by the main script) in the VM falls outside of the start limit multiplied with the *total size* it triggers write or delete operations until the stop limit multiplied with the total size is reached.

All file operations configuration files are evaluated before use. This is done graphically by plotting the utilization curve for the virtual disk, i. e. the sum of the currently allocated data blocks after each file operation. By doing this we can see if the curve corresponds to our idea of the typical user activity we want to emulate. We want the partition to be almost full at least once during the experiment, but not too many times. Likewise there shall be a steady increase in the utilization of the partition, but not too steady or rapid. We favour sequences with a certain degree of high frequency ripple along the utilization curve. This is meant to correspond to the user actively creating, modifying and deleting files on a regular basis and in that way increasing the external fragmentation of the file system. The degree of utilization for the similarity test can be seen in Figure 4.1.

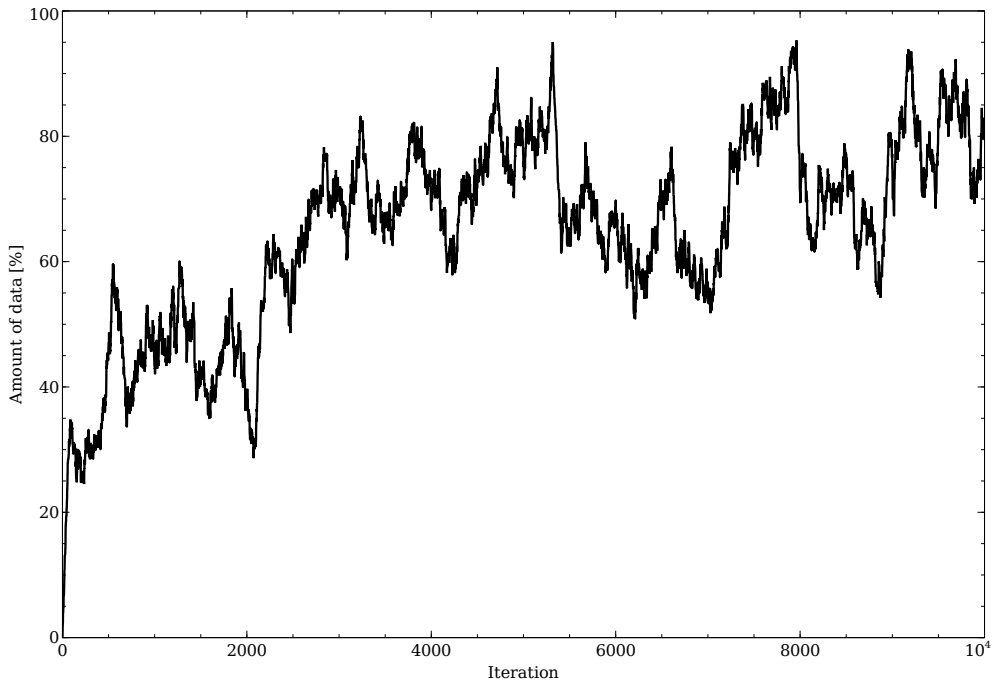


Figure 4.1.: The simulated user behavior, i. e. the total amount of data stored in the NTFS partition after each iteration according to the file operations configuration file used in the similarity test.

4. Experimental Setup

The script on the host checks if a VM is started before it sends a file operation command. There is also a check of the exit status of the VM scripts. If the exit status of a script indicates an error the iteration counter is decremented and the file operation is repeated. This behavior might induce extra allocation changes due to the repeated power cycling of the VM. We accept them because occasionally a real user might also be forced to reboot a computer.

Every file operation is logged externaly to the VM. 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 file size difference for the increase and decrease operations can be calculated from the stored transactions if needed. The log file is stored on the disk shared between the VM and the host and hence does not interfere with allocation algorithm of the virtual disk.

The write operation scripts (create, increase and decrease) are set to write one 512 byte sector at a time, i. e. we do not actively cache any write operations. Each file contains the file operation sequence number and the current sector's position within the file (similar to the LCN). This enables us to see the raw write pattern in the virtual disk file if needed. The iteration sequence number is used as the file name to further increase the traceability.

All write operations are stream writing operations, i. e. the size of the file is not known in advance. This corresponds to for example a file being downloaded from the internet. The reason for writing each file one sector at a time is to try to deprive the allocation process of any knowledge of the file's final size. Therefore the allocation process can only be optimized for each write (which might be delayed due to internal caching). This can induce a more stochastic behavior of the allocation algorithm and possibly hide any deterministic behavioral patterns from us. However, that would lead to an underestimation of the experimental results, which on the other hand is better than an overestimation.

The create and decrease file operation scripts both write new files (they use the `wb` flag in the Python open command). A file size decrease might therefore lead to deallocation of the original clusters and allocation of a smaller amount of new clusters. It might even lead to deallocation and allocation of the same clusters depending on the type of allocation algorithm used. The increase script appends new data at the end of an existing file, using the `ab` flag. Therefore files that have had their size increased might contain two or more sequence numbers.

The allocation status of the file system is collected through the `$Bitmap` file of the VMs. The file is extracted as the last step in each iteration. Thus the only difference between two consecutive `$Bitmap` file copies are the allocation changes induced by the latest file operation and any active system processes. Since their activity is part of the standard system behavior we let them run as normal. The resolution of the `$Bitmap` file is 4 KiB per bit, but we use 512 B blocks when writing. We therefore have set the

size factors to multiples of eight in the experiments.

Each VM is installed with its specific Windows OS using standard parameters. Then the Python 2.7 executable is added to the VM together with and an auto started .bat script. The .bat script writes a small token file to the disk shared by the VM and the host to signal the completion of the boot process. The script is small enough to fit into an MFT record and hence does not require any new cluster allocation (outside the MFT) in the VM. 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. The path environmental parameter of the VM is modified to incorporate the Python installation. Finally the security level of the OS in the VM is lowered to allow access without a password.

We use the Linux dd tool to extract the \$Bitmap file as raw data. Consequently the position of the file must be known and static, which it is. To find the position of the \$Bitmap file we use the `istat` tool from The Sleuth Kit before the virtual disks are copied to the host nodes. This procedure is required because we are not allowed to install new software on the cluster nodes.

The fact that we can access the data of the virtual disks externally without having to use `dislocker` [225] or similar tools proves that the disks are not (automatically) BitLocker encrypted. Furthermore, VirtualBox implemented the TPM and Secure Boot functions required by BitLocker in version 7.0.0, which was released 10-10-2022 [99]. Our latest article was published in July 2020.

4.3.3. Map Creation

When the experiment is finished we do a differential analysis of each consecutive pair of \$Bitmap copies. This gives us the LPVA position of the allocation changes for each file operation, including system files. The probability of a new file causing more allocations than its system ditto is high, because otherwise the OS would be inefficient. Hence the system file allocation changes are negligible compared to the file operation changes.

We are interested in the allocation frequency at each LPVA position. Consequently only allocations are relevant and the deallocations are thus neglected. The remaining posts are then merge sorted in order of LPVA position.

The final step is to create a map in a similar way as described in Section 4.2. The storage media is divided into equally sized areas and the mean allocation frequency of each area is calculated. The areas together form the map, which easily can be plotted. The resolution of the map is given by the number of areas.

4.4. Writing Type Behavior

The experiment specifically studies the writing allocation behavior, which includes stream and block writing (see Section 2.5). We therefore study the allocation algorithm's behavior in different situations regarding for example disk utilization, file system fragmentation and file size. To achieve this the \$Bitmap files of two VMs are manipulated to emulate different states of file system fragmentation. On a higher level the experiment will test whether Microsoft Windows in combination with NTFS is using the best fit allocation strategy, as indicated in the literature [50, 162].

4.4.1. Virtual Hardware

The experiment is executed using two VirtualBox v. 5.2.20 VMs running Windows 7 and Windows 10. The included Windows OSs are selected to cover both older and newer Windows releases. The reason for excluding Windows 8 and 8.1 is their similarity in behavior to Windows 7. The similarity has been seen in previous experiments. The Windows 7 machine has a fixed size 64 GiB disk and the Windows 10 machine a 1 TiB ditto, to cover both small and large disks. Each VM has a folder shared with the host. The virtual disks are loop mounted with read/write access rights to enable us to use standard digital forensic tools like the Sleuth Kit.

The disk of the Windows 7 machine has already been used in an earlier experiment (see Section 4.3). This has given it a heavily fragmented file system corresponding to an old and well used (home) computer. There is one large area of free clusters remaining at the end of the partition. To even out the fragmentation pattern we split the unallocated area into five smaller areas of approximately 262,000 clusters (1 GiB) each. The Windows 10 machine is newly installed and the file system therefore only contains files from the installation. It represents a new office computer.

4.4.2. Process Description

As in the repeated file operation experiment described in Section 4.3 each VM is power cycled after each file operation. This is done to ensure that all file operations are reflected in the virtual disk file on the host. The experiment starts by retrieving a copy of the \$Bitmap file from the still powered down VM. This \$Bitmap copy shows the status of the file system prior to the first file operation. Then the following steps are repeated for each file write operation:

1. The VM is powered on.
2. A file signaling that the boot process has finished is written to the shared folder.
3. A file write operation is executed.

4. A file signaling that the file operation has finished is written to the shared folder.
5. The VM is powered down.
6. Use the `vboxmanage showvminfo` tool to check for a complete and successful power down of the VM.
7. The allocation information related to the newly written file is extracted using the `istat [204]` tool.
8. The `$Bitmap` file of the VM is copied to the host using `icat [204]` tool.

To allow the VM, as well as the host, to properly write all files to disk a ten second long pause is introduced between the steps 1 to 3 of each iteration.

4.4.3. Implementation

To generate both stream and block writing situations a Python 2.7 script is used. Block writing is implemented through writing a file to an array in RAM, which is then written to disk in a single write operation. Stream writing is implemented by directly using Python's own file write operation, writing one 512 B sector of the file at a time. Each 512 byte block is uniquely identifiable by a consecutive number together with a file identifier. The marking is used as a backup in case we have to read the raw data directly from the virtual disks. The marking is also used to verify that the `istat` tool reports the allocation pattern in the same order as the clusters are written.

We write files of different sizes to determine if the allocation algorithm behaves differently depending on the size of the file. In the first round of the experiment we use 4, 128, 511, 512, 513 and 1,024 MiB files. The 511 and 513 MiB files originates from the first version of the Python script, where the type of writing is dependent on the size of the file. If the file size is ≤ 512 MiB the file is block written and if it is larger it is stream written. The second round of the experiment also includes 12, 96, 384, 768 and 1,536 MiB files to give a more even distribution of the file sizes. The Python script is also updated to allow different writing strategies regardless of file size.

4.4.4. Bitmap Manipulation

To cover for possibly different allocation behaviors depending on the available amount of unallocated space in the file system we manipulate the `$Bitmap` file of the virtual disks. The `$Bitmap` file of the Windows 7 machine is manipulated once and the Windows 10 `$Bitmap` file three times, because it has not been used to the same extent.

The Windows 7 VM's main partition is heavily fragmented from the beginning, but still contains a contiguous unallocated area of approximately 5.3 GiB. This area is

4. Experimental Setup

divided into five smaller areas (see Table C.1), all other unallocated areas are kept in their original state. The modified layout is called *BM 7:1* throughout the text. The unmodified layout is never used.

Table 4.4.: The unallocated areas in the original 5.3 GiB space after the *BM 7:1* manipulation.

Start [cluster]	Size [cluster]
15,372,418	262,000
15,634,546	262,143
15,896,818	262,144
16,159,090	262,145
16,421,363	262,160

The unused Windows 10 VM main partition of 1 TiB contains two large unallocated areas of approximately 497 GiB and 511 GiB respectively. This original, unmodified, layout is called *BM 10:0* throughout the text.

After the initial file writing operations to *BM 10:0* we manipulate the \$Bitmap file to fragment the allocation layout. The 497 GiB area is divided into 501 equally sized blocks of 120,000 clusters (468.75 MiB) each. The 51 GiB area is divided into 1,026 blocks of increasing size, from 120 clusters (480 KiB) to 123,120 clusters (approximately 481 MiB) in steps of 120 clusters. The two areas together contain 123,342,120 (approximately 471 GiB) unallocated clusters after the modification. All other unallocated areas in the partition are unmodified. We refer to this manipulation setting as *BM 10:1*.

In the *BM 10:2* \$Bitmap layout the available unallocated space is decreased even more. This is done by first restoring the VM disk to its original state and then creating the unallocated areas shown in Table C.2 from the two initially unallocated areas.

The *BM 10:2* layout is meant to test the block writing allocation behavior by forcing the allocation algorithm to choose between a few very large areas and many small. The seemingly odd values used in Table C.2 are chosen to avoid creating unallocated areas with sizes being multiples of 2. The total amount of unallocated space in the manipulated area is 11,715,760 clusters (44.7 GiB) after the modification.

However, the *BM 10:2* \$Bitmap manipulation is not enough to generate any significant fragmentation during the block writing process. We therefore manually decrease any remaining free areas larger than 120,000 consecutive clusters with a factor 10. This gives a total of 3,361,315 clusters (12.8 GiB) of unallocated space in the manipulated

Table 4.5.: The modified areas of *BM 10:2*.

Spaces	Function	Tot. [cluster]
511	$\text{int}(120,000/x + 27; x = [512 : -1 : 2])$	711,541
23	120,000	2,760,000
512	$7x + 13; x = [0 : 511]$	922,368
16	7,999	127,984
1	17	17
29	$\text{int}(1,800,000/x + 17; x = [1 : 29])$	7,131,462

area. This \$Bitmap manipulation is referred to as *BM 10:3*.

4.5. HDD vs. SSD Allocation Differences

An experiment is carried out to empirically check if there are any differences in the file data allocation (at the logical layer) between an HDD and an SSD. The experiment is executed using a computer running Microsoft Windows 10 Enterprise (build 1809), a Seagate Barracuda 2 TB (ST2000LM015) HDD, a Toshiba OCZ TR150 480 GB (TRN150-25SAT3-480G) SSD and a Debian 11 (Bullseye) computer. A total of 64 GiB data (512 files à 128 MiB) are written to two 128 GiB partitions, one in each storage media. The writing is done using the Windows computer and a Python script.

The HDD also contains another partition placed in front of the partition used for the experiment. The HDD and SSD partitions therefore start at different LBAs.

The experiment consists of the following steps:

1. Create two equally sized partitions in an HDD and SSD respectively using `fdisk` in Linux.
2. Format the partitions as NTFS using the Microsoft Windows computer.
3. Write an equal number of equally sized files to the disks using the Microsoft Windows computer.
4. Extract the file allocation data from both partitions using the `istat` tool from the Sleuth Kit in Linux.
5. Filter out the allocated cluster addresses with the help of the `head` and `tail` tools in Linux.
6. Check the allocation data for equality using `sha1sum`.

4. Experimental Setup

We also compare the MFT numbers of all files using the `fls` tool from the Sleuth Kit.

4.6. BitLocker Allocation Changes

The BitLocker experiment is executed to check if BitLocker v. 2.0 affects the file data allocation of already used storage media when being activated. The experiment uses the same hardware as the HDD vs. SSD experiment (see Section 4.5), except for the HDD.

The experiment includes the following steps:

1. Format the existing 128 GiB partition on the SSD as NTFS using the Windows computer.
2. Write 512 equally sized (128 MiB) files named 0 to 511 to the partition using the Windows computer.
3. Erase the odd numbered files using the Windows computer.
4. Create an image of the partition using `dd` in the Linux computer.
5. BitLocker encrypt the full partition using the Windows computer.
6. Use the `dislocker` tool in the Linux computer to mount the encrypted partition.
7. Create an image of the `dislocker` mounted partition using `dd` in the Linux computer.
8. Deactivate BitLocker (decrypt the partition) using the Windows computer.
9. Create an image of the partition using `dd` in the Linux computer.
10. Use the `istat` tool in the Linux computer to extract all MFT records in the partition.
11. Check and compare the `istat` data of all the previously written files from before, under and after the BitLocker encryption using `sha1sum` in the Linux computer.

The deletion of every second file is meant to give BitLocker the possibility (and reason) to change the file data allocation. Since all files have the same size they theoretically can be moved and compacted into half the allocated space. Any change will be manifested in either the MFT files or the created files, or both.

5. Result

The result of the three experiments presented in Sections 4.2 to 4.4 are summarized in Chapter 6. The full text can be found in Articles A to D.

The results of the two new experiments presented in Sections 4.5 and 4.6 are described below. The first experiment is run to check for differences in the file allocation pattern between an HDD and SSD. The second experiment is executed to check for differences in the file allocation pattern of a partition when it is unencrypted versus BitLocker encrypted.

5.1. HDD vs. SSD Allocation Differences

The allocation pattern of the HDD and SSD partitions are exactly the same. Each file is allocated to the same LPVAs of the partitions. Also the MFT record numbers of the files in each disk are the same. The main difference between the partitions are the time stamps, which differ since the disks were not populated at the same time. Also the *Last User Journal Update Sequence Number* of the MFT records differ. The journal sequence numbers of the HDD are increasing relative to the sequence numbers of the SSD. This is done in intervals, which can be seen in Table 5.1.

Table 5.1.: The *Last User Journal Update Sequence Number* of the HDD is incremented with a higher rate than in the SSD. The table shows the interval lengths and values.

Interval length	Diff. increase
420	0
13	80
18	72
1	136
55	72
1	136
4	72

5. Result

We use the `binwalk` tool to check for binary differences in the \$Bitmap files of the two partitions. There is one more allocated cluster in the HDD than in the SSD, at cluster 761,923. Otherwise the \$Bitmap files of the two disks are identical. The extra cluster in the HDD belongs to the SYMEFA.DB file, which is part of the Symantec Endpoint Protection antivirus product [226–228].

5.2. BitLocker Allocation Changes

The SHA-1 sums of the `istat` data for all files are identical. This means that all files have exactly the same cluster addresses allocated both before, under and after the BitLocker encryption, i. e. the file data allocation is identical during the whole process.

We use the `binwalk` tool to check for any differences between the \$Bitmap files from before, under and after the BitLocker encryption. There are several differences, three allocations are reset when the partition is decrypted and two allocations remain. The differences can be seen in Table 5.2.

Table 5.2.: Differences in allocation shown in the \$Bitmap file before, under and after BitLocker encryption. Each bit in the \$Bitmap represents one cluster, 4 KiB, in the file system. 0x01 = 1 allocation, 0x07 = 3 allocations, 0x10 = 1 allocation and 0xFF = 8 allocations.

Position [cluster]	Before [0x]	Under [0x]	After [0x]
40,977	01 00 00 00 00	FF FF FF FF 07	01 00 00 00 00
303,121	01 00 00	FF FF 01	01 00 00
462,064	00	10	10
565,265	01 00 00	FF FF 01	01 00 00
761,896	00	07	01

The majority of the differences shown in Table 5.2 are unique to the BitLocker encryption. Three areas are only changed during the BitLocker encryption, one is retained after the encryption has been deactivated and one differs in all three states. The largest newly allocated area is 136 KiB at cluster 40,977. There are also two larger areas allocating 64 KiB each. All allocation changes are done in an area between the start of the partition and the \$MFT file, which is situated at cluster 786,432.

The files and directories occupying the extra allocated areas during the BitLocker encryption can be seen in Table 5.3. There are seven files and one directory, which is divided into two fragments.

Table 5.3.: Files created by BitLocker, i. e. new allocations during the encrypted stage. There are seven files and one *directory* in the new allocations.

Pos. [clust]	Len. [clust]	Name
40,977	16	FVE2.e40ad34d-dae9-4bc7-95bd-b16218c10f72.1
40,993	2	FVE2.24e6f0ae-6a00-4f73-984b-75ce9942852d
40,995	16	FVE2.da392a22-cae0-4f0f-9a30-b8830385d046
303,121	16	FVE2.e40ad34d-dae9-4bc7-95bd-b16218c10f72.2
462,068	1	<i>System Volume Information</i> (2^{nd} part)
565,265	16	FVE2.e40ad34d-dae9-4bc7-95bd-b16218c10f72.3
761,896	1	<i>System Volume Information</i> (1^{st} part)
761,897	1	FVE2.aff97bac-a69b-45da-aba1-2cfbce434750.1
761,898	1	FVE2.aff97bac-a69b-45da-aba1-2cfbce434750.2

The System Volume Information directory does not exist before the BitLocker encryption, but remains after the partition is decrypted again. The directory contains the seven new files. The files are all related to BitLocker [229]. Two files, FVE2.{aff...750}.1 and FVE2.{aff...750}.2, contain information other than zeros. The rest are zero filled.

6. Summary of Work

The results of the PhD project are presented in four articles:

Article A M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8

Article B M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Using NTFS Cluster Allocation Behavior to Find the Location of User Data.” In: *Digital Investigation 29* (2019), S51–S60. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.018

Article C M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Disk Cluster Allocation Behavior in Windows and NTFS.” in: *Mobile Networks and Applications 25.1* (Feb. 2020), pp. 248–258. ISSN: 1572-8153. DOI: 10.1007/s11036-019-01441-1

Article D M. Karresand, G. Dyrkolbotn, and S. Axelsson. “An Empirical Study of the NTFS Cluster Allocation Behavior Over Time.” In: *Forensic Science International: Digital Investigation 33* Supplement (July 2020), p. 301008. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2020.301008

The term LBA has been corrected to LPVA in the articles. This is also pointed out in the text introducing each article.

The articles form a logical continuation of the previous research work done during the licentiate studies [39, 110–114]. The new articles provide the foundation for the mapping concept, which is central to the PhD project. It is built on a black box model to make it OS, file system and hardware independent, as well as future proof. Article A also contains an experimental evaluation of the mapping concept. The evaluation compares the performance of sampled hash-based carving with and without using a map.

The mapping concept and each article are presented in separate sections below. The connections between the research questions and publications are shown in Figure 6.1. The main contributions of the PhD project are given in Chapter 7.

As can be seen in Figure 6.1 the main research question is answered by a combination of all four publications, which follow a logical path towards the complete answer to the question. Article D answers both Research questions RQ:4 and RQ:5. This is done

6. Summary of Work

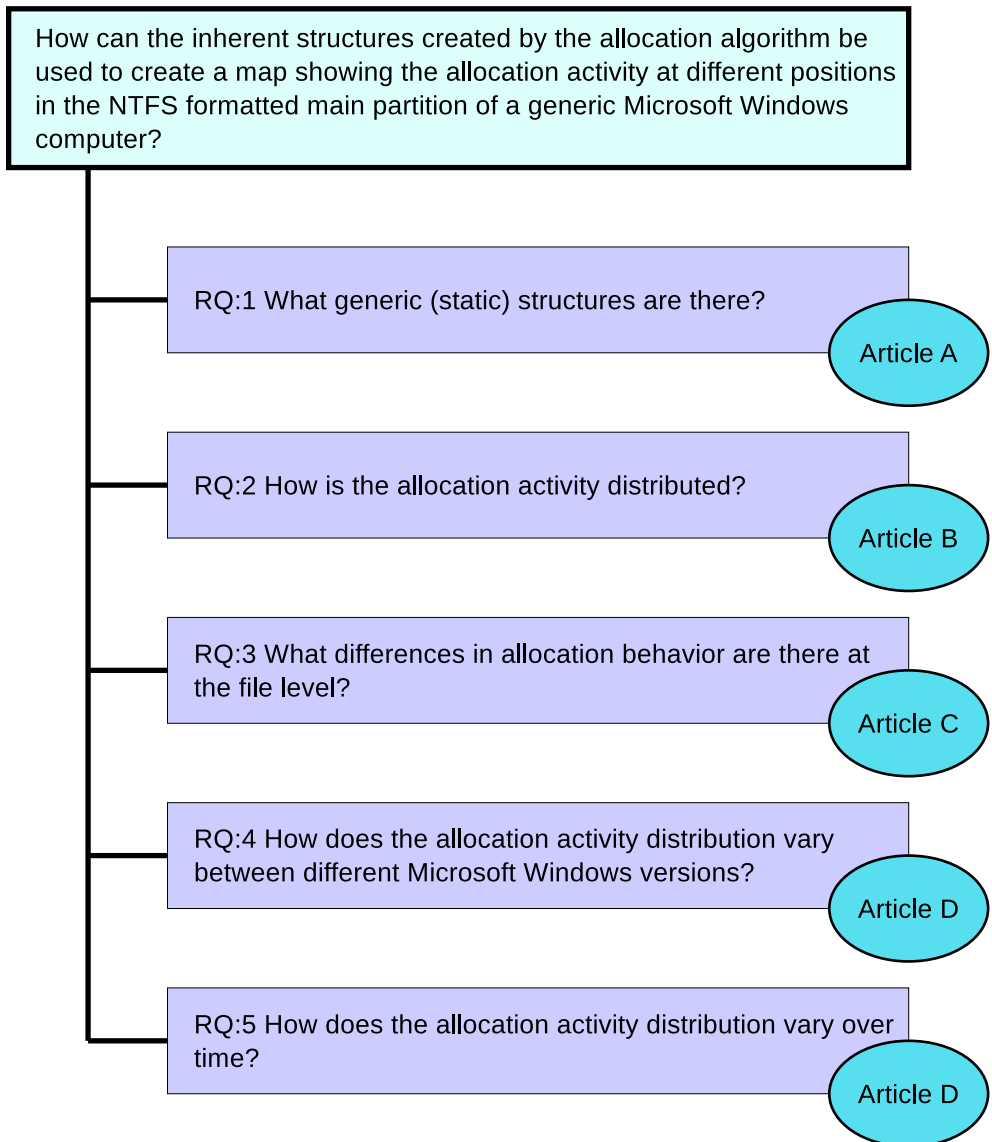


Figure 6.1.: The relationship between the main research question, Articles A to D and the Research questions RQ:1 to RQ:5.

6.1. Article A: Creating a Map of User Data in NTFS to Improve File Carving

to utilize the synergies between the experiment and analysis due to their procedural similarities. Articles B and D are based on the same experiment, but the analysis of the experimental data differ.

All data collected during the experiments will be made available in accordance with the FAIR principles as soon as possible. During the mean time the data are available for download by contacting the author. Since it is more than 889 GB compressed data the best transfer technique has to be agreed upon in each individual case.

6.1. Article A: Creating a Map of User Data in NTFS to Improve File Carving

Article A compares the cryptographically hashed (SHA-1) content of each 512 byte sector of 30 unrelated real-life NTFS formatted disk partitions. Sectors at the same LPVA having the same content are called *static* sectors, which are used to show that there are generic, static, structures present in the partitions. The purpose is to find the allocation activity at different LPVA in a partition, built on the assumption that user data are the main source of file allocations. User data are also assumed to be highly unique, since the probability of two users creating exactly the same file content is negligible. However, different types of file sharing will cause the same data to be present in several partitions.

The computers used in the experiment belong to family, friends and coworkers of the authors. The contributors were chosen based on a convenience selection. The extraction of the hash values are done in a privacy preserving way to protect the contributors. The hashing of the sectors were executed on the source computer and the hashes were written to a portable disk. Consequently no cleartext data ever left the contributor's computer. Only the resulting hashes and their corresponding LPVAs were stored and used in the analysis.

The analysis was made by sorting and filtering the hash data in different ways. Unused parts, i. e. filled with 0x00 or 0xFF were automatically filtered out, because of the focus on unique data. However, some partitions had been used before and reformatted, which left old file data in the unallocated space of the partitions. We therefore checked the last 20 GiB of the partitions to see if they contained 0x00 or 0xFF filling or not. We then knew that some of the unique data were currently not allocated, but had been before. However, a static area at the end of a partition still means that sometime during the lifetime of a collection of partitions the same data have been written to the same LPVA, although some parts of the static area are currently not allocated.

The first static area was found close to the start of the main partition. The second was found at the beginning of the MFT, in cluster number 786,435. The second area contains the last part of four MFT records (MFT No. 12–15), counted from the start

6. Summary of Work

of the MFT. Since these MFT records fall within the initial part of the MFT also the previous 12 MFT records are fixed in position. Consequently the MFT start position was located exactly 3 GiB into each partition, in cluster number 786,432.

During the analysis of the hashed sectors we also found several semi-static¹ areas. Since we only have access to the hashed data we cannot see whether they contain the same data, or are the result of hash collisions. The semi-static areas cover many consecutive sectors with varying hashes, consequently the hash collision alternative can be ruled out, although not theoretically.

To evaluate the mapping concept we conducted an experiment using hash-based carving and sampling. The experiment checked whether the map or a uniform distribution better modeled four real life disks used as ground truth. The mean improvement using the map was 2%, with a variation between -5% to 9%. However, the map was created from a small number of mostly privately owned home computers. The four ground truth disks had been used for storage of data in an office. We therefore expect the results to improve when the map is based on a larger number of heterogeneous disks.

Article A answers Research question RQ:1 by identifying two static areas covering 30 unrelated NTFS partitions. The areas show that there are generic structures in NTFS. Thus the first requirement for the creation of a map is fulfilled. Furthermore, the result has been achieved in a real world setting, which increases its value.

The semi-static areas presented in Article A contributes to the field of hash-based carving with focus on hash collisions and misattribution. They can however also be used in other research and application fields, for example they give a better understanding of the NTFS allocation process. They can also be used within the data persistence and data reduction fields, since they give information on data that rarely change.

6.2. Article B: Using NTFS Cluster Allocation Behavior to Find the Location of User Data

The work presented in the article collects statistics on the file allocation activity at different LPVAs by executing 10,000 file operations (create, delete, increase and decrease) in a random pattern in 32 VMs. The allocation activity statistics can be used to guide a forensic investigator to the most probable areas for finding user data (assuming user data is the most frequent data to be written to disk).

¹A semi-static area is having the same content (hash values), but not position, in several unrelated partitions. It is simply content shared among computers. There is of course a theoretic possibility that equality is fictive and instead the result of a hash collision. However, the probability of finding hash collisions covering that amount of diverse sectors is infinitesimal.

6.2. Article B: Using NTFS Cluster Allocation Behavior to Find the Location of User Data

Each VM was equipped with a 64 GiB main partition². Four different versions of Microsoft Windows (7, 8, 8.1 and 10) were used in the experiment. It ran more than 8,000 iterations consisting of a series of steps. First the VM was started and a file operation (creation, deletion, increase or decrease of the file's size) from a randomly and pre-created list was executed. The list was meant to model the behavior of different types of users, for example home, file sharing and multimedia handling. The virtual computer was then stopped and the \$Bitmap file was extracted. After the extraction the virtual computer was started again and the procedure was repeated with the next file operation in the list.

The extracted \$Bitmap files were used to analyze the allocation frequency at each LPVA. By comparing consecutive \$Bitmap files the LPVA of the new allocations for each file operation could be found. The number of allocations per LPVA were calculated and converted into a map of the allocation activity of a generic NTFS formatted partition. The map can be used by a forensic investigator to increase the efficiency of his or her work by directing it to the most active area allocation wise.

The allocation activity was found to be low at the first 10 to 20 GiB of a partition, which corresponds to the approximate size of a Microsoft Windows installation. Then the activity increased rapidly to a maximum, followed by a slow decrease towards the end of the partition. The decrease in allocation activity towards the end is almost linear and at one point it goes below the allocation activity at the start of the partition. A partition should therefore preferably be read in falling allocation activity order, possibly switching to the start of the partition before reading the very end.

The general allocation behavior was similar between all Windows versions. However, Microsoft Windows 10 lacked a large dip in the allocation activity in the middle of the partition, something the other versions showed. The exact reason for the dip has not been possible to find.

Article B [231] answers Research question RQ:2 and shows that the allocation activity varies in an NTFS partition. The first part has a low allocation activity. The major part of the activity is seen approximately 10 GiB into a partition. That area starts with a steep increase in allocation activity, which then slowly decreases towards the end of the partition. Consequently the second requirement for the creation of a map is fulfilled.

The main contribution of the article is focused on the location of the main allocation activity in NTFS. The knowledge gained strengthens the mapping concept, because it confirms the findings from Article A and also gives more detailed information on the allocation activity at different positions within an NTFS formatted partition.

²Depending on the Microsoft Windows version one or more small extra partitions are created during the installation phase. This has been the default behavior since Windows 7 [96] and allows the main partition to be automatically BitLocker encrypted. Consequently the main partition was slightly smaller than 64 GiB.

6.3. Article C: Disk Cluster Allocation Behavior in Windows and NTFS

Article C presents research on the allocation behavior differences between stream written and block written files. During block writing the final size of the file is known to the OS. It therefore can optimize the allocation pattern of the file. Text documents are typical candidates for block writing.

During stream writing the data to be written is streamed to the OS, which therefore does not know the final size of the file to be written. Consequently the OS has to allocate free space based on an estimated size and the allocation pattern cannot be optimized. An example of a stream written file is streamed multimedia downloaded from the internet. Internal buffering can help mitigate the stream writing allocation problem to a certain degree, but not completely. The effect of the different types of writing becomes stronger with heavier external fragmentation of the file system.

We conducted an experiment writing files to two VMs running Microsoft Windows 7 and 10. The Windows 7 VM was taken from an earlier experiment and consequently its virtual disk was already fragmented to a certain degree. The Windows 10 VM was freshly installed and consequently its virtual disk was unfragmented. To control the fragmentation of the virtual disks we manipulated the \$Bitmap file in different patterns, forcing the OS and file system to fragment the files when writing. We then used a Python script to vary the writing type and size of the files.

The experiment shows that the fragment size decreases during block writing and increases (from very small values) for stream writing. The information can for example be used in the file carving field by enabling a more detailed search for file fragments. The information on how the sizes of fragments increase or decrease during allocation of a file helps the detection of fragments from the same file, where the detection of data type is extended by the probable size of the next fragment. The information will therefore also contribute to the reassembly step of file carving by decreasing the amount of sequences to test.

Our experiment also shows that the allocation algorithm in NTFS is not strictly best fit. This is described in the literature, although also contradicted in other literature. The behavior is more complex, for example large unallocated areas are divided into a few smaller areas that are allocated individually. This behavior has also been seen in data from previous experiments.

The research results can also be used to determine the causal order of files in a partition. During block writing the areas to allocate are chosen based on the best fit algorithm, allocating the largest available unallocated areas first. The next block written file therefore gets smaller fragments. Depending on the number of file operations and writing types executed in between the two block writings the result will be more or

6.4. Article D: An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

less reliable. The relationship between the writing type, allocation pattern and causal ordering of files is interesting and has been left as future work.

Article C [78] answers Research question RQ:3. It shows that there are easily detectable differences between block and stream writing. Consequently the third key requirement for the creation of a map is fulfilled.

The main contribution of the article is the information gained on differences in allocation behavior in NTFS depending on the way a file is written. The information increases the resolution of the map by offering a separation depending on how files are written, i. e. the type of file on a high level.

6.4. Article D: An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

Article D continues the research presented in Article B on the allocation behavior of NTFS. This time the focus lies on allocation behavior differences over time, as the file system ages. The research also includes studies of allocation behavior differences between Windows products, as well as between different sizes of storage media. The new analysis found new areas with low allocation activity. Furthermore, the maximum and median fragment sizes in Windows 7 showed linear properties over time, which was not the case for Windows 10. No differences in allocation behavior were found between small and large storage media partitions.

We also noticed that the majority of the allocation activity took place in an approximately 20 GiB wide band of the partitions. When the file system grew older the band widened and its center moved towards the end of the partitions. The same widening of the allocation activity band was not seen at lower LPVA positions.

The allocation algorithm also prioritized to fill in gaps in the already allocated areas, instead of using the large unallocated areas closer to the end of a partition. This sometimes lead to external fragmentation, although there were unallocated areas large enough to avoid fragmentation. Consequently the unallocated areas at the end of a partition are used to a much lower extent, than the beginning and middle parts of a partition.

By adopting the maps to different Windows versions, as well as the age of the file system, the digital forensic investigation will become more efficient, because of more detailed information in the map. Likewise an archivist can plan the work with old or corrupt storage media in a better way, improving the chance of a successful data recovery.

Article D [232] answers both Research question RQ:4 and Research question RQ:5. The article shows that the allocation algorithm of NTFS is not strictly best fit. This behavior is described earlier by Bahjat and Jones [84] and maaartinus et al. [85] and

6. Summary of Work

contradicts the book by Carrier [50]. The article also shows that the allocation behavior differs between Microsoft Windows 10 and the older Windows versions used in the experiment. Article D furthermore gives information on how NTFS ages. It also shows differences in allocation behavior at larger changes in file system utilization. The main contribution of the article is a more detailed information on the behavior of NTFS. The information can for example be used to increase the resolution and precision of the mapping concept.

6.5. Map Examples

The mapping concept is an integral part of the PhD project's research goal. Like for geographical maps our maps show the position of different attributes. In our case the attributes are connected to the allocation process of a file system and OS.

Our current maps are only proof-of-concept implementations of the mapping concept. They present examples of what different types of maps might look like, nothing more. Typical examples are Figures B.2 to B.4 and D.1 in the articles. Real, usable, maps have to be based on a much larger amount of data. Because of the complexity of the map creation process (see Section 2.9) the maps should preferably be created in cooperation with a cartographer.

A general problem with our mapping concept is the large address space. A 4 TB NTFS partition covers 1,000,000,000 LPVA addresses (using 4 KiB sectors). A full resolution map covering n file operations has to have a resolution of $1,000,000,000 \times n$ pixels. Consequently the map has to be scaled, but how this should be done is still to be decided. We have tried to section the partition's LPVA space into groups for which we calculate a mean allocation frequency value. However, the calculations are affected by any static features of the partitions, such as the start of the MFT at 3 GiB into a partition. If the size of the groups vary (the total number of groups are constant regardless of partition size) the result of the first group(s) will be affected. However, the effect of the static MFT position is decreasing with larger partition sizes, because of the larger groups used (assuming a fixed number of groups).

Another challenge to the mapping concept is the range of the size of storage media. There might still be storage media below 100 GiB present in working laptops, while at the same time a desktop computer can be equipped with storage media in the range of tenths of TiB. And they are still partitioned into just a few partitions, often with one covering most of the storage media. To be able to create a truly generic map from all those storage media we need to find a way of combining them without affecting the precision of the map. The division that we use of the partition space into groups (see Articles A and B) will not work for such a large set of partition sizes. However, two NTFS partitions with the same usage pattern and total amount of data will have

approximately the same space occupied, regardless of the size of the partition. This is because of the the allocation behavior where filling in gaps in existing allocated areas is prioritized over the use of unallocated areas further into the partition.

To help the reader this section gives alternatives to the maps presented in the articles. Once again the presented maps are simplified proof-of-concept models.

6.5.1. Static Areas

The positions of the static and semi-static areas within a generic partition are of importance to many areas of application. A static area is static in both content and position. The varying attribute is the number of partitions with the same static areas. A typical example is shown in Figure 6.2. The bar charts shown can be used as rudimentary maps of the partitions used in the experiments. The terms chart and map will therefore be used interchangeably.

The bar chart in Figure 6.2 is scaled with a factor of 18,984,550 and divided into 100 groups (10 GB each). This is done to increase its readability. The y-axis of the chart shows the mean sizes (in partitions) of the static areas in each group. There can be several groups of static areas at each position. Therefore the chart is depicted as a stacked bar chart.

There are also some structures at the start of the positional range in Figure 6.2. However, the resolution of the chart is too low to give a clear view of the structures. We therefore have created a chart of the first 5 GiB with higher resolution, which is shown in Figure 6.3. This time the scaling factor is 189,845, dividing the full positional range into 10,000 groups.

The structures at the start of the partitions are clearly visible in Figure 6.3. The two static areas covering all partitions can be seen, but the scaling (the use of the mean value) hides their full extent. To do that we would have to use a 1:1 scale of the chart, making it unusable.

Semi-static areas are harder to map, because they do not share LPVA positions between partitions. The varying sizes of the partitions is also a complicating factor. The problem is comparable to creating a common map of Shanghai, Stockholm and Gjøvik. They all share similar attributes (road, buildings, parks, etcetera), but the placement and scale of the attributes differ.

It might be possible to combine partitions of different sizes into a single map by using positions relative to the total sizes of the partitions. However, this concept assumes that the semi-static areas are positioned at the same relative positions, which might not be true. Due to its complex nature the creation of a map of semi-static areas therefore has been left as future work.

6. Summary of Work

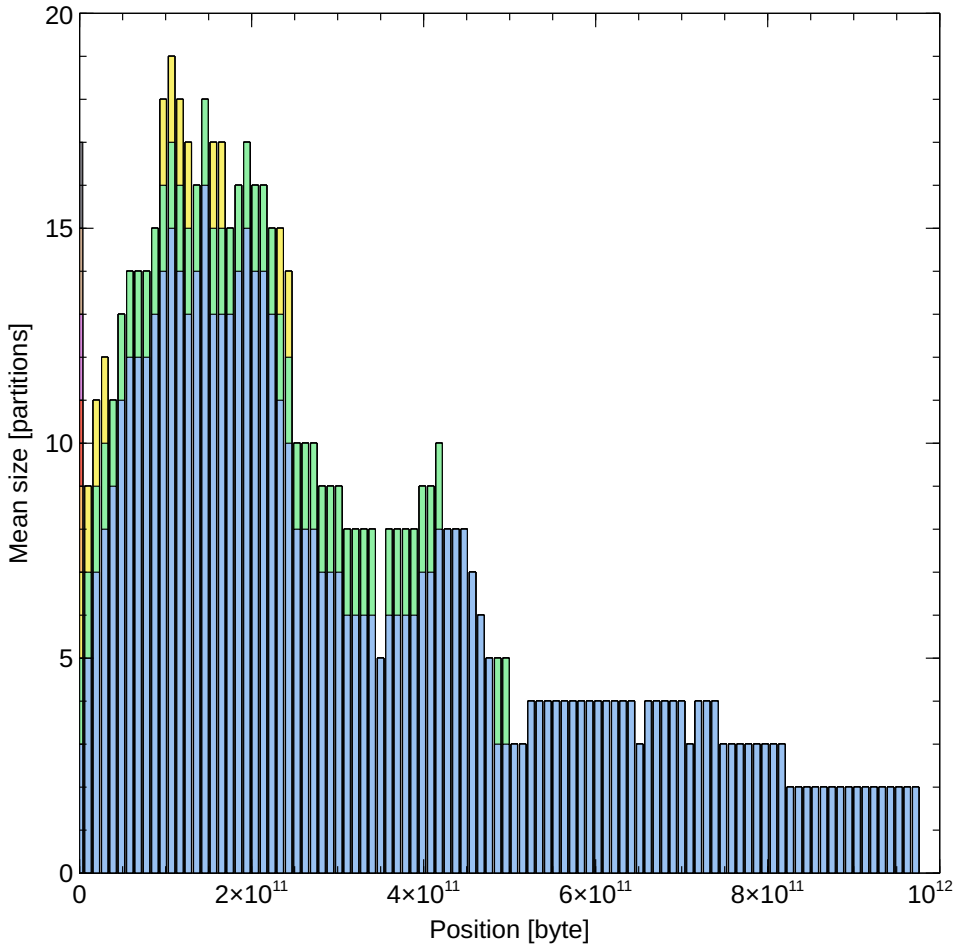


Figure 6.2.: A stacked bar chart of the mean size (in partitions) of the static areas found in NTFS formatted partitions in different areas. Each color represents a static area. The height of a colored bar indicates its size in partitions. The total height of a bar gives the sum of all static areas (in partitions) in an area.

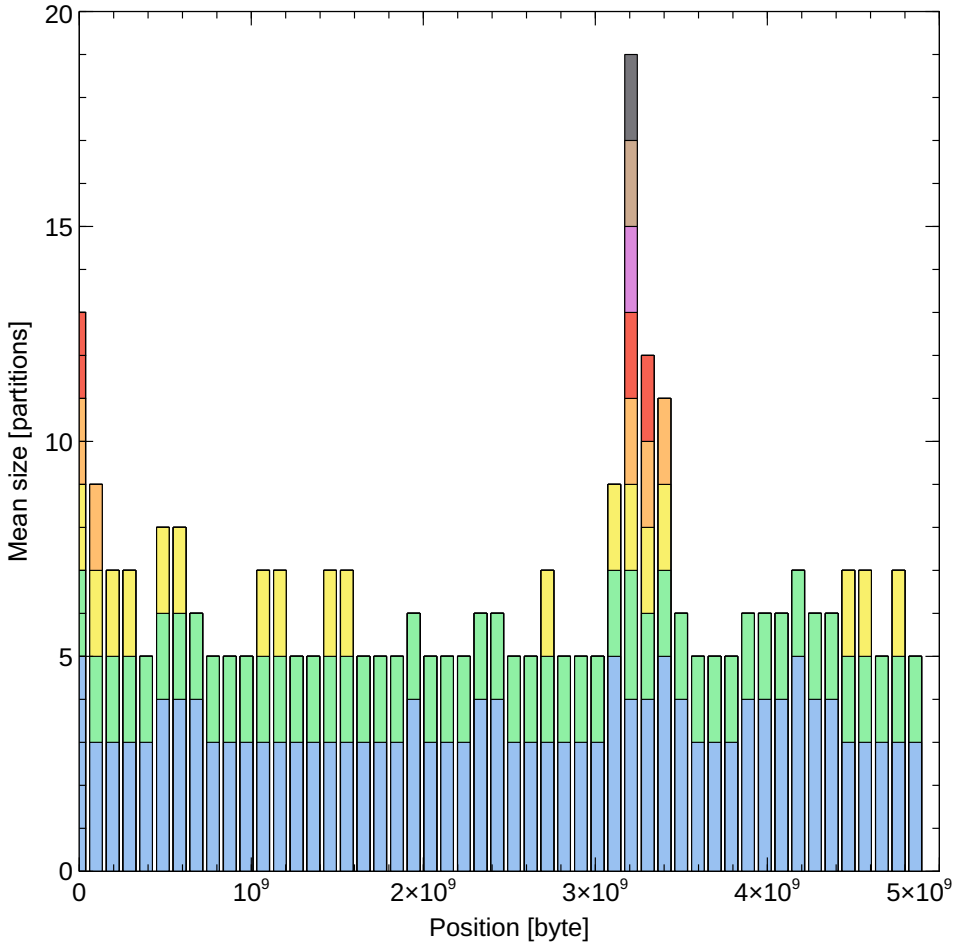


Figure 6.3.: A stacked bar chart of the mean size (in partitions) of the static areas found in NTFS formatted partitions in different areas. The chart shows the first 5 GB of the partitions (the first bar in Figure 6.2). Each color represents a static area. The height of a colored bar indicates its size in partitions. The total height of a bar gives the sum of all static areas (in partitions) in an area.

6.5.2. Allocation Activity

A map can also show the allocation activity at different LPVA positions. The problem is to find a balance between an overview and detailed information. Likewise the activity should be generalized, i. e. not focused on a specific partition.

An allocation activity map can be compared to a topological map with contour lines. However, the allocation activity map only needs one dimension to give the position. Hence it can be depicted as a line diagram (see Figure 6.4).

The line diagram format can also be used to show the general allocation behavior over time. To make the map clearer and less crowded both the positional and temporal dimensions have to be sampled and averaged. Such a map is shown in Figure 6.5.

Using an interactive map the scale of the line diagram can be adjusted dynamically and allow zooming. However, the map is a generalization of a real world situation and therefore have a limited resolution. The situation can be compared to a map of a standard hill. It should have at least one peak from which it is sloping downwards in all directions. A map of a specific hill on the other hand might show several peaks, different angles of the slopes, boulders and pits scattered over the area, etcetera.

6.5.3. Advanced Map Attributes

If several OSs are shown in the same map they can be shown as multiple line diagrams. This is the case shown in Article B. It is also possible to add a dimension and show them as a three dimensional heat map. Such a map would resemble a topological map with contour lines.

The heat map format is generally applicable and can also be used to give a detailed view of the aging of a file system. This can be done using histograms of the allocated LPVA position for each file operation. Hence the y-axis shows position, the x-axis shows time and the z-axis shows the height of the histograms. The heat map format can also be used to show the allocation behavior differences between various use cases.

A map can also show the fragmentation pattern in different situations, for example in connection with longer sequences of deletions or additions of files. Such a map should also contain information on the type of file operation (create, delete, increase or decrease) for each allocation. The map can be used to explain specific patterns found in (working) file systems. Typically the map should show a limited time frame, for example a few allocations before and after the sequence. Longer time frames are also possible, covering larger parts of the life of a partition.

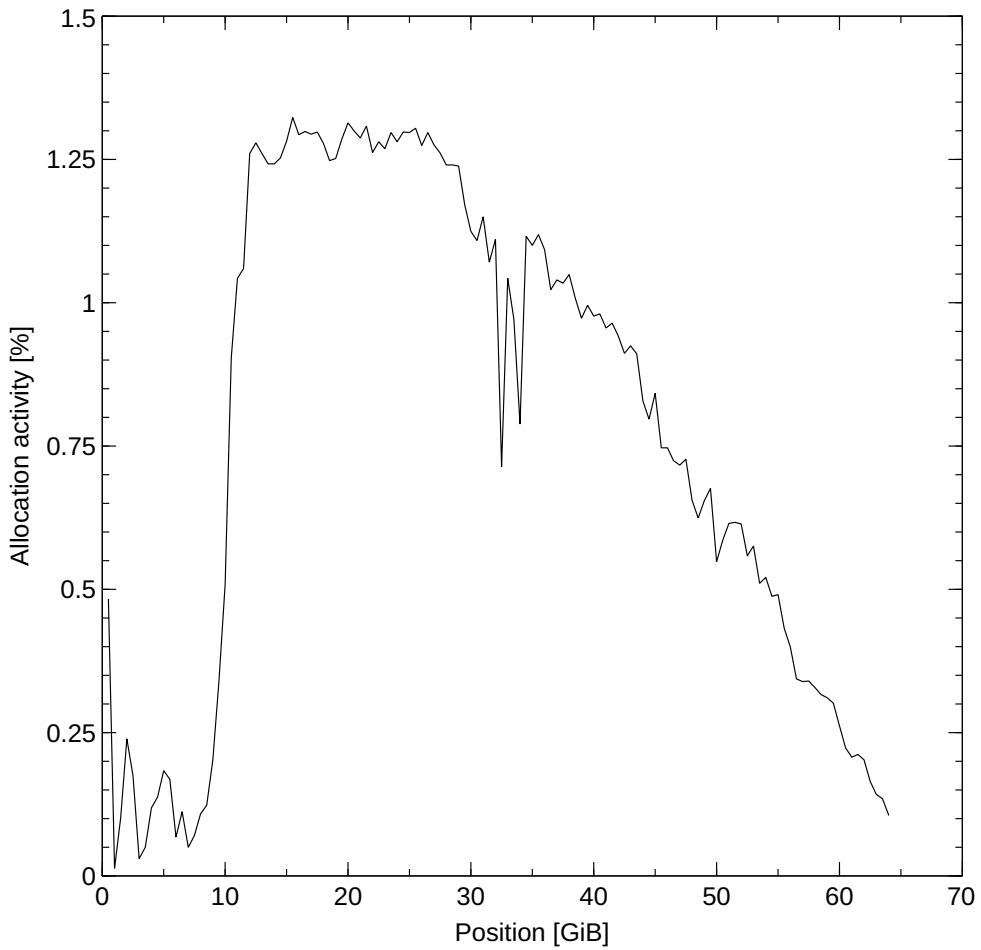


Figure 6.4.: A map of the mean allocation activity at different positions in a generic NTFS 64 GiB partition covering Microsoft Windows 7, 8, 8.1 and 10. The map shows the mean frequency of 512 MiB blocks relative the total number of allocations.

6. Summary of Work

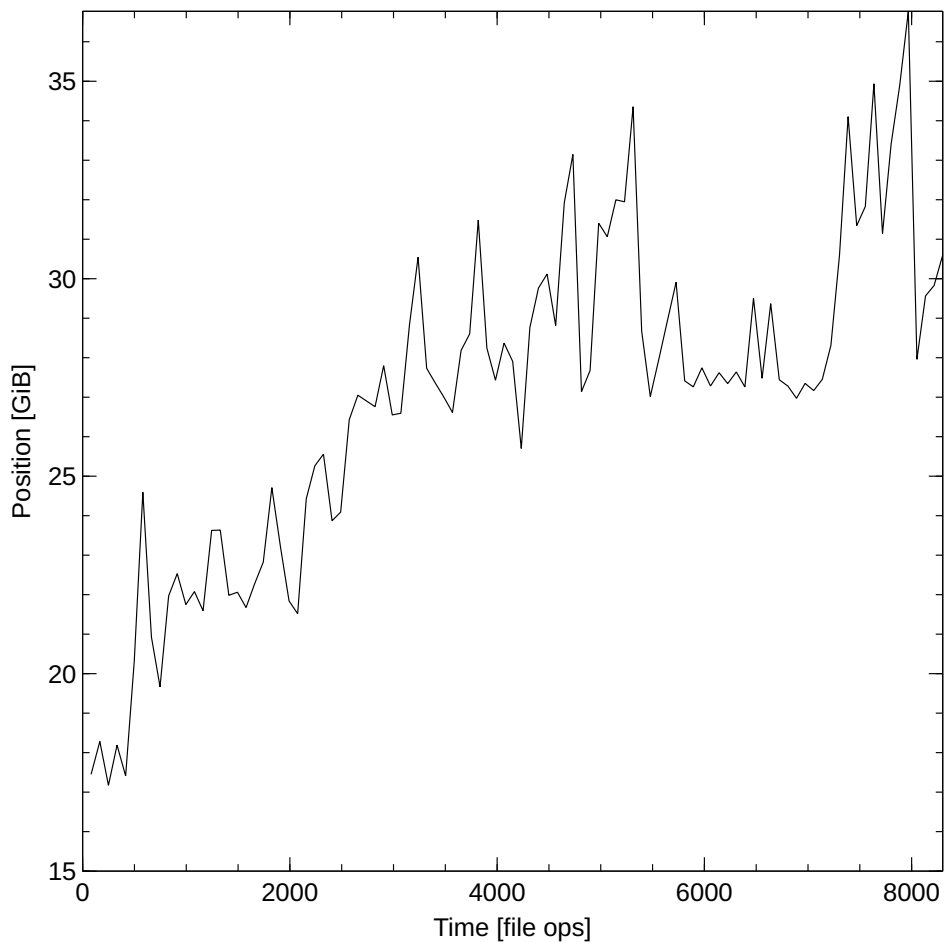


Figure 6.5.: A map of the mean allocation activity over time (in file operations) at different positions. The mean is calculated for groups of 83 allocations, giving 100 groups in total for the given time span. The high activity starts approximately 17 GiB into the partition, at the end of the area used during installation.

7. Contributions

The PhD project work contributes to different areas within and outside of the digital forensics field. The main contribution is the mapping concept. The more significant contributions fall within the file fragment and hash-based carving fields. We have used NTFS to create proof-of-concept maps, but the concept is adaptable to any file system. We have also successfully evaluated the efficiency increase during sampled hash-based carving when using a map compared to a uniform distribution.

Since the mapping concept is new there is no existing research field to build upon and improve. Instead a selection of research and application fields are used to give examples of contributions of the PhD project. The selection is based on the perceived correlation between the PhD project and the chosen areas. There are still many areas we have not yet thought of. We leave these to the reader to suggest.

7.1. File Fragment Carving

File fragment carving uses different means to identify and reassemble fragments of data. The carving is done using only information taken from the data in the fragments themselves. First the fragments of a specific file have to be identified. Then they have to be reassembled into (parts of) the original file again. Files can consist of several types of data, which complicates the process [123]. The behavior of the allocation algorithm and the degree of fragmentation also affects the carving.

The PhD project contributes through the mapping concept and the information on the allocation behavior of NTFS. The information is data type agnostic and therefore can be used for any file type. Searches for file fragments can be directed to the most relevant areas of a partition with the help of the map. Article D contributes with information on file system aging. By knowing how the relevant search areas move over time an investigator can fine tune the searches.

When the relevant file fragments have been found the fragment classification phase will benefit from the new knowledge on the allocation behavior during different writing types presented in Article C. The information will for example help optimizing the block size during scanning. In this way both the speed and detection rate of the process can be improved.

The information from Article C will also indicate the correct order of the found blocks. It can also be used to mark probable fragment sequences in a disk image. That

7. Contributions

will then allow the file fragments to be collected in the most probable order already during the extraction phase.

The most probable starting fragment of a stream written file can be identified using the information in Article C. First all fragments ≤ 20 KiB are located and marked as possible starting fragments. Then the information on the increasing fragment sizes during stream writing is used to reduce the number of fragments to work with. The most probable order of the fragments will also be indicated. Consequently the PhD project directly improve all steps in the file fragment carving process.

7.2. Hash-Based Carving

Hash-based carving searches for fragments of known files in a partition. The method works without access to any metadata and compares cryptographic hashes of file fragments to known fragment hashes.

The biggest challenge to the field is the amount of comparisons to do. Sampling is therefore used to increase the speed of the search. This is however a balance between speed and detection rate. The lower the sampling frequency, the higher the speed. However, that increases the risk of missing a relevant fragment.

The mapping concept enables the use of a dynamic sampling frequency in hash-based carving. The frequency then follows the allocation activity as presented in Article B. Also the age of the file system can be taken into account using the research presented in Article D. Furthermore, the hash-based carving will benefit by starting at the most relevant region of a partition. This gives an immediate increase in the efficiency of the carving, sampling or not.

A simple test (see Article A) with live data shows a 2% increase in efficiency, with a maximum of 9%, using dynamic sampling. The test was conducted using HDDs from four office computers, with a map primarily based home computers. We expect to get better results using a map based on a larger set of data. If the data used for the map represent a similar use case as the evaluation media the results will be even better.

We have also theoretically evaluated the possible efficiency increase during sampled hash-based carving (see Section 1.2). We then get a possible speed increase of more than 25%, combined with an improvement of more than 25% of the detection rate. This is calculated on a case where the number of samples (the speed of the scan) is fixed. The actual speed and detection rate increases depending on the prerequisites in each specific case.

The research on hash collisions and misattributions will benefit from the work presented in Article A. The semi-static areas found during the PhD project can be further explored to extend the hash-based carving field. The presence of the semi-static areas show that there are large areas of equal content shared between unrelated NTFS

formatted partitions. Therefore these areas can be avoided when collecting (sampling) blocks to hash. This is done by detecting the content of a semi-static area and then skipping to its end.

7.3. Digital Stratigraphy

The digital stratigraphy research can be used in a broad range of application areas. Consequently any contribution to the research field will give a leveraging effect at the application level.

The PhD project contributes with the mapping concept. It can be used to visualize digital strata. The higher the allocation activity at an LPVA position, the higher the number of strata.

Article D contributes through the knowledge on how the allocation behavior is affected by the aging of NTFS partitions. It also contributes through the differences found in allocation behavior between Microsoft Windows versions. Furthermore, Article B contributes through the knowledge on how larger file system events will be manifested in the allocation pattern. All these features directly affect the stratigraphy of the file system.

7.4. NTFS Fragmentation

The NTFS fragmentation research is focused on fragmentation differences between file types. The results show that certain file types are often heavily fragmented, while other types almost never fragment. The goal is a more efficient resource utilization by predicting the difficulty of carving different file types. The research field would for example benefit from a better understanding of the allocation algorithm of Microsoft Windows and NTFS.

The PhD project contributes by providing new information on how, when and why fragmentation occurs. This allows the research field to better understand the reasons behind fragmentation. It will also show the connection between the allocation behavior, the type of file writing strategy (as in Article C), and the actual file fragmentation. Our contribution is therefore also relevant at a higher level within the NTFS fragmentation field.

7.5. NTFS Data Allocation Process

The research within the NTFS data allocation process is focused on studies of the possibility to reverse the allocation process, i. e. to backtrack an allocation sequence. The

7. Contributions

research is also trying to provide the ability to time stamp or causally order files and data based on the allocation pattern. Some progress has been made, but the research field would benefit from more information and knowledge.

The mapping concept contributes to the NTFS data allocation process field by supplying updated and detailed information on the allocation behavior of Microsoft Windows NTFS. For example the information on the change in behavior due to aging of the file system will provide the allocation process researchers with the possibility to measure the distribution of fragments (and accompanying statistical metrics). Based on that they can then get high level information on the age of a file allocation operation.

Likewise the extended knowledge on the differences in allocation behavior based on the file writing type will help the research on the causal ordering of files. Newer block written files will (with high probability) have smaller fragments sizes than older files. Stream written files will fill the smallest unallocated areas first, leaving larger areas to newer files. Consequently also the span of available unallocated areas at a given moment will be manifested in the allocation pattern and in that way give information on the causal order of files.

7.6. Data Recovery

File carving and data recovery overlap, but data recovery is also used outside of the digital forensics field. It is typically used by professions like archivists and librarians. Within the data recovery field the requirement for forensically sound extractions is left out in favor for recovering larger amounts of data.

The PhD project contributes through the added ability to directly focus any limited read operations to where it is most likely to find valuable data. In this way the disk access is kept to a minimum. This is important when the storage media is almost worn out or has been physically damaged. In that case each read operation will degrade the status of the storage media and effectively limit the available read operations.

The static areas presented in Article A can be used in data recovery situations. First of all the presence of NTFS in a partition can be identified without access to any meta-data. The static areas can also be used to align a tool with the start of a NTFS partition. The idea is similar to the `testdisk` [177] tool in Linux. Since the area is part of the MFT it can also be used to find the start of a missing MFT, which should be placed exactly 3 GiB into a partition.

7.7. Data Mapping

The data mapping research field is studying how to map a working file system. This is done per case and helps the investigator to get an overview of the specific storage

media. If there is no working file system the situation becomes more dire. There is an Enscript module that is used to map single files using the EnCase digital forensic tool, but it is unclear whether the module can handle also broken file systems or just files.

The mapping concept contributes to the data mapping research field by offering a possibility to expand the field by adding maps of broken file systems and of a generic storage media or partition. Both the new research field opened by the PhD project and the data mapping field will benefit from an exchange of information.

8. Conclusion and Future Work

This chapter concludes the thesis and presents research left as future work. We can conclude that all research questions have been answered and that we have created a proof-of-concept map of a generic NTFS partition. However, a deeper study of the relevant parameters needed to improve the quality of the mapping concept has been left as future work.

8.1. Conclusion

We have studied the inherent structures in the allocation pattern in NTFS. The research has been conducted using storage media from both live and virtual computers. The aim has been to improve the efficiency of the digital forensic process. This has been shown both theoretically, as well as empirically. The goal was to create a map of the allocation activity at different positions within a generic NTFS partition, which has been done.

The main conclusion to be drawn from the results of the PhD project is that there are enough inherent structures within NTFS to create a map of the allocation activity at different positions. A prototype map has been used to show an increase in the efficiency of a simple hash-based carving operation. The improvement is self-evident, because it is more efficient to look for something lost at the place where it is most likely to be found.

The writing type allocation differences can be used to show the type of a file on a high level. Files written as a stream of data are allocated fragments of increasing size. Files written as a block of data are allocated fragments of decreasing size. The starting fragment of a set of file fragments can be found based on the same information. If the set contains stream written data the smallest fragments are possible starting candidates. If the set contains block written data the largest fragments are starting candidates.

The writing type result in combination with the best fit allocation strategy can also be used to indicate the causal ordering of files. A file with larger fragments is probably older than a file with smaller fragments.

The experiment on the differences between HDD and SSD disks showed that the physical differences were hidden by the disk controller. The allocation pattern of the files written to the two disk partitions during the experiment were identical. The only difference was that an extra cluster was allocated in the HDD. It contained a fragment of the SYMEFA.DB file from the Symantec Endpoint Protection product.

8. Conclusion and Future Work

The BitLocker experiment studied the changes to the file allocation pattern induced by the BitLocker encryption process. A storage media partition was populated with files, encrypted and decrypted again. Disk images were created after each step and their contents compared. No changes to the allocation pattern of the written files were found, but seven new files and a directory were created during the encryption process. All these files were placed in the new directory and were parts of the BitLocker mechanism. The files were removed when the partition was decrypted again. Only the directory remained.

In Section 6.5 we have presented and discussed a number of maps covering different parts of the PhD project. However, these are proof-of-concept maps and should only be used as examples. Furthermore, we discuss some of the challenges involved when creating a map. To be usable in a real-life situation it must for example be based on a large set of data. However, the use of a black box principle makes the mapping concept versatile and able to handle both present and future storage media technologies, OSs and file system combinations.

8.2. Future Work

The mapping concept is generic and applicable to any file system. Since the PhD project has been focused on the allocation behavior of NTFS, an obvious future work is to expand the research to other file systems and OSs. This also includes mobile phones and other small digital devices.

The experiments should be enlarged to enable more detailed information to be retrieved. Also the sequences of file operations used can be expanded to further strengthen the results. Preferably the sequence of operations should be based on real-life data. This would allow the experiments to better imitate the natural development of a file system in use. However, this requires access to a computer cluster for an extended period of time.

The efficiency gain within different application areas should be properly measured. The simple efficiency experiment presented in Article A should only be regarded as a proof-of-concept and not be used for comparisons to other fields. The same is true for the theoretical improvement estimation presented in Section 1.2. However, a more thorough experimental evaluation should be done, which has to be properly adapted to each specific field and designed to deliver comparable results.

The relationship between the writing type, allocation pattern and causal ordering of files must be studied further. Similar ideas have been found in the literature [77, 162, 163, 165]. We have found indications showing that it might be possible to order files causally, but this has been in an experimental setting. To extend and deepen the research real world disks should be used.

New application areas of the mapping concept have to be further explored. The concept might also have to be adopted to each field of application. This is left as future work, because we cannot anticipate all current and future application areas. New areas will be added over time, which requires the mapping concept to evolve with them.

The complex nature of semi-static areas has forced us to leave the mapping of them as future work. However, several different solutions have been proposed. One is to use relative positions enabling differently sized partitions to be used in the same map. However, this is complicated by the fact that the semi-static areas need not be found at the same relative positions in different partitions. There are also other problems on the road towards a map that can be used in real investigations. Due to the complexity of the mapping process it is recommended to involve a cartographer to ensure the usability of the maps.

The problem of properly scaling the map, as well as combine different partition sizes into one generic map, also has to be studied further. This includes studies of usage patterns, where the number of files, the size of the files and the file system housekeeping activity (for example deletion of files and defragmentation) will affect the allocation frequency at different positions. The usage patterns might then allow differently sized partitions to be combined, since they all have their data stored in approximately the same LPVA areas. Consequently the study of relevant parameters to improve the stability, applicability, precision and quality of the mapping concept is an important future work.

Bibliography

- [1] M. Scanlon. “Battling the digital forensic backlog through data deduplication.” In: *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*. 2016, pp. 10–14. DOI: 10.1109/INTECH.2016.7845139.
- [2] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, and R. Perez-Aleman. “Automated mapping of large binary objects using primitive fragment type classification.” In: *Digital Investigation 7*. Supplement (2010). The Proceedings of the Tenth Annual DFRWS Conference, S3–S12. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.002.
- [3] S. Tzu. *Sun Tzu on the Art of War*. Last accessed 18-11-2017. Allandale Online Publishing, 2000. URL: https://sites.ualberta.ca/~enoch/Readings/The_Art_Of_War.pdf.
- [4] “About the Content.” In: ed. by B. Rystedt and F. Ormeling. Stockholm, Sweden: International Cartographic Association, 2014. Chap. Foreword.
- [5] T. Owens and T. Padilla. “Digital sources and digital archives: historical evidence in the digital age.” In: *International Journal of Digital Humanities* 1.3 (July 2021), pp. 325–341. DOI: 10.1007/s42803-020-00028-7.
- [6] E. Casey. “Digital Stratigraphy: Contextual Analysis of File System Traces in Forensic Science.” In: *Journal of Forensic Sciences* 63.5 (2018), pp. 1383–1391. DOI: 10.1111/1556-4029.13722.
- [7] M. Broz, M. Patocka, and V. Matyas. “Practical Cryptographic Data Integrity Protection with Full Disk Encryption Extended Version.” In: *IFIP International Information Security Conference*. 2018. DOI: 10.1007/978-3-319-99828-2_6.
- [8] T. Müller and F. Freiling. “A Systematic Assessment of the Security of Full Disk Encryption.” In: *IEEE Transactions on Dependable and Secure Computing* 12.05 (Sept. 2015), pp. 491–503. ISSN: 1941-0018. DOI: 10.1109/TDSC.2014.2369041.
- [9] R. Montasari and R. Hill. “Next-Generation Digital Forensics: Challenges and Future Paradigms.” In: *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*. 2019, pp. 205–212. DOI: 10.1109/ICGS3.2019.8688020.

Bibliography

- [10] N. Hassan. *Digital Forensics Basics: A Practical Guide Using Windows OS*. New York, New York, USA: APress, 2019. DOI: 10.1007/978-1-4842-3838-7.
- [11] P. Gladyshev and J. James. “Decision-theoretic file carving.” In: *Digital Investigation* 22.Supplement C (2017), pp. 46–61. ISSN: 1742-2876. DOI: 10.1016/j.diin.2017.08.001.
- [12] European Police Office (Europol). *Internet Organised Crime Threat Assessment (IOCTA) 2016*. Tech. rep. European Cybercrime Centre (EC3), 2016.
- [13] B. Schatz. “Wirespeed: Extending the AFF4 forensic container format for scalable acquisition and live analysis.” In: *Digital Investigation* 14 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S45–S54. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.016.
- [14] D. Quick and K. Choo. “Data reduction and data mining framework for digital forensic evidence: Storage, intelligence, review and archive.” In: *Trends & Issues in Crime and Criminal Justice* 480 (Sept. 2014), pp. 1–11. ISSN: 1836-2206.
- [15] V. Roussev, C. Quates, and R. Martell. “Real-time digital forensics and triage.” In: *Digital Investigation* 10.2 (2013). Triage in Digital Forensics, pp. 158–167. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.02.001.
- [16] F. Breitinger, G. Stivaktakis, and H. Baier. “FRASH: A framework to test algorithms of similarity hashing.” In: *Digital Investigation* 10.Supplement (2013). The Proceedings of the Thirteenth Annual DFRWS Conference, S50–S58. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.06.006.
- [17] V. Roussev. “Managing Terabyte-Scale Investigations with Similarity Digests.” In: *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*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 19–34. ISBN: 978-3-642-33962-2. DOI: 10.1007/978-3-642-33962-2_2.
- [18] M. Cohen and B. Schatz. “Hash based disk imaging using AFF4.” In: *Digital Investigation* 7 (2010). The Proceedings of the Tenth Annual DFRWS Conference, S121–S128. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.015.
- [19] P. Turner. “Unification of Digital Evidence from Disparate Sources (Digital Evidence Bags).” In: *Digital Investigation* 2.3 (Sept. 2005), pp. 223–228. ISSN: 1742-2876. DOI: 10.1016/j.diin.2005.07.001.

- [20] M. Geiger. "Evaluating Commercial Counter-Forensic Tools." In: *In Proceedings of the 5th Annual Digital Forensic Research Workshop*. 2005, pp. 39–41.
- [21] D. Lillis, B. Becker, T. O'Sullivan, and M. Scanlon. "Current Challenges and Future Research Areas for Digital Forensic Investigation." In: *Annual ADFSL Conference on Digital Forensics*. Last accessed 22-04-2022. May 2016, pp. 9–20.
- [22] E. Casey, M. Ferraro, and L. Nguyen. "Investigation Delayed Is Justice Denied: Proposals for Expediting Forensic Examinations of Digital Evidence*." In: *Journal of Forensic Sciences* 54.6 (2009), pp. 1353–1364. ISSN: 1556-4029. DOI: 10.1111/j.1556-4029.2009.01150.x.
- [23] Statista. *Seagate's average capacity of hard disk drives (HDDs) worldwide from FY2015 to FY2021, by quarter*. Last accessed 16-10-2021. July 2021. URL: <https://www.statista.com/statistics/795748/worldwide-seagate-average-hard-disk-drive-capacity/>.
- [24] SATA-IO. *20th Anniversary*. Last accessed 18-10-2021. 2021. URL: <https://sata-io.org/20th-anniversary>.
- [25] Anson. *PCIe vs SATA vs USB – Storage Interfaces Explained*. Last accessed 18-10-2021. Mar. 2019. URL: <https://www.unbxtech.com/2019/03/pcie-sata-usb-interfaces-explained.html>.
- [26] Western Digital Corporation. *Difference between Sequential and Random Access operations*. 2014. URL: https://kb.sandisk.com/app/answers/detail/a_id/8150/~~/difference-between-sequential-and-random-access-operations.
- [27] ConduSiv. *I/Os Are Not Created Equal – Random I/O Versus Sequential I/O*. Last accessed 14-04-2022. URL: <https://condusiv.com/sequential-io-always-outperforms-random-io-on-hard-disk-drives-or-ssds/>.
- [28] I. Al Awadhi, J. Read, A. Marrington, and V. Franqueira. "Factors Influencing Digital Forensic Investigations: Empirical Evaluation of 12 Years of Dubai Police Cases." In: *Journal of Digital Forensics, Security and Law* 10.4 (2015), pp. 7–16. DOI: 10.15394/jdfsl.2015.1207.
- [29] R. Hranický, F. Breitingner, O. Ryšavý, J. Sheppard, F. Schaedler, H. Morgenstern, and S. Malik. "What do incident response practitioners need to know? A skillmap for the years ahead." In: *Forensic Science International: Digital Investigation* 37 (2021), p. 301184. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2021.301184.

Bibliography

- [30] N. Ramli, S. Hisham, and G. Badshah. “Analysis of File Carving Approaches: A Literature Review.” In: *Advances in Cyber Security*. Ed. by N. Abdullah, S. Manickam, and M. Anbar. Singapore: Springer Singapore, 2021, pp. 277–287. ISBN: 978-981-16-8059-5. DOI: 10.1007/978-981-16-8059-5_16.
- [31] D. Dietrich and F. Adelstein. “Archival science, digital forensics, and new media art.” In: *Digital Investigation* 14 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S137–S145. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.004.
- [32] S. Garfinkel and M. McCarrin. “Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb.” In: *Digital Investigation* 14.Supplement 1 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S95–S105. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.001.
- [33] F. Breitingner, C. Rathgeb, and H. Baier. “An Efficient Similarity Digests Database Lookup - A Logarithmic Divide & Conquer Approach.” In: *Journal of Digital Forensics, Security and Law* 9.2 (2014), pp. 155–166. DOI: 10.15394/jdfs1.2014.1178.
- [34] F. Breitingner and K. Petrov. “Reducing the Time Required for Hashing Operations.” In: *Advances in Digital Forensics IX - 9th IFIP WG 11.9 International Conference on Digital Forensics, Orlando, FL, USA, January 28-30, 2013, Revised Selected Papers*. Ed. by G. Peterson and S. Sheno. Vol. 410. IFIP Advances in Information and Communication Technology. Springer, 2013, pp. 101–117. DOI: 10.1007/978-3-642-41148-9_7.
- [35] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks. “Distinct Sector Hashes for Target File Detection.” In: *Computer* 45.12 (Dec. 2012), pp. 28–35. ISSN: 0018-9162. DOI: 10.1109/MC.2012.327.
- [36] K. Foster. “Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus.” MA thesis. Monterey, California, USA: Naval Postgraduate School, Sept. 2012.
- [37] S. Garfinkel, A. Nelson, D. White, and V. Roussev. “Using purpose-built functions and block hashes to enable small block and sub-file forensics.” In: *Digital Investigation* 7.Supplement (2010). The Proceedings of the Tenth Annual DFRWS Conference, S13–S23. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.003.

- [38] S. Collange, Y. S. Dandass, M. Daumas, and D. Defour. “Using Graphics Processors for Parallelizing Hash-Based Data Carving.” In: *2009 42nd Hawaii International Conference on System Sciences*. Jan. 2009, pp. 1–10. DOI: 10.1109/HICSS.2009.494.
- [39] M. Karresand. “Completing the Picture — Fragments and Back Again.” Licentiate thesis. Linköping Institute of Technology, Linköping University, Sweden, May 2008.
- [40] E. Casey, G. Fellows, M. Geiger, and G. Stellatos. “The growing impact of full disk encryption on digital forensics.” In: *Digital Investigation* 8.2 (2011), pp. 129–134. ISSN: 1742-2876. DOI: 10.1016/j.diin.2011.09.005.
- [41] E. Casey and G. Stellatos. “The Impact of Full Disk Encryption on Digital Forensics.” In: *SIGOPS Oper. Syst. Rev.* 42.3 (Apr. 2008), pp. 93–98. ISSN: 0163-5980. DOI: 10.1145/1368506.1368519.
- [42] L. Luciano, I. Baggili, M. Topor, P. Casey, and F. Breitingner. “Digital Forensics in the Next Five Years.” In: *Proceedings of International Conference on Availability, Reliability and Security, Hamburg, Germany, August 27–30, 2018 (ARES 2018)*. 2018, Article 46. DOI: 10.1145/3230833.3232813.
- [43] LLC SysDev Laboratories. *The basics of file systems*. Last accessed 15-10-2021. Sept. 2021. URL: <https://www.ufsexplorer.com/articles/file-systems-basics.php>.
- [44] OS Today. *Which file system is the default for hard drives in Windows 10?* Last accessed 15-10-2021. 2021. URL: <https://ostoday.org/windows/which-file-system-is-the-default-for-hard-drives-in-windows-10.html>.
- [45] Microsoft. *Overview of FAT, HPFS, and NTFS File Systems*. Last accessed 15-10-2021. Sept. 2021. URL: <https://docs.microsoft.com/en-us/troubleshoot/windows-client/backup-and-storage/fat-hpfs-and-ntfs-file-systems>.
- [46] StatCounter. *Desktop Operating System Market Share Worldwide - January 2023*. Last accessed 01-02-2023. Jan. 2023. URL: <https://gs.statcounter.com/os-market-share/desktop/worldwide>.
- [47] Microsoft. *Definitions for system volume and boot volume*. Oct. 2009. URL: <https://support.microsoft.com/en-us/help/314470/definitions-for-system-volume-and-boot-volume>.
- [48] A. Allievi, A. Ionescu, M. Russinovich, and D. Solomon. *Windows Internals part 2*. seventh ed. Pearson Education, Inc., 2021. ISBN: 978-0-13-546233-1.

Bibliography

- [49] Microsoft. *New Capabilities and Features of the NTFS 3.1 File System*. Last accessed 15-11-2021. 2007. URL: https://www.betaarchive.com/wiki/index.php/Microsoft_KB_Archive/310749.
- [50] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005. ISBN: 0321268172.
- [51] Microsoft. *Windows and GPT FAQ*. Last accessed 04-05-2022. 2022. URL: <https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/windows-and-gpt-faq?view=windows-11>.
- [52] M. Kumar. "Solid state drive forensics analysis – Challenges and recommendations." In: *Concurrency and Computation: Practice and Experience* 33 (2021). DOI: 10.1002/cpe.6442.
- [53] V. van der Meer, H. Jonker, and J. van den Bos. "A contemporary investigation of NTFS file fragmentation." In: *Forensic Science International: Digital Investigation* 38 (2021), p. 301125. DOI: 10.1016/j.fsidi.2021.301125.
- [54] C. Buckel. *Understanding Flash: The Flash Translation Layer*. Last accessed 08-10-2018. Sept. 2014. URL: <https://flashdba.com/2014/09/17/understanding-flash-the-flash-translation-layer/>.
- [55] R. Reiter, T. Swatosh, P. Hempstead, and M. Hicken. *Accessing logical-to-physical address translation data for solid state disks*. Last accessed 08-10-2018. Nov. 2014. URL: <http://www.freepatentsonline.com/8898371.html>.
- [56] J. Barbara. *Solid State Drives: Part 5*. Last accessed 08-10-2018. Apr. 2014. URL: <https://www.forensicmag.com/article/2014/04/solid-state-drives-part-5>.
- [57] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. "A Survey of Flash Translation Layer." In: *J. Syst. Archit.* 55.5-6 (May 2009), pp. 332–343. DOI: 10.1016/j.sysarc.2009.03.005.
- [58] A. Silberschatz, P. Galvin, and G. Gagne. *Operating System Concepts*. 9th ed. Wiley, Dec. 2012.
- [59] IDEMA. *Advanced Format Definitions, Abbreviations, and Conventions*. 2017. URL: http://idema.org/?page_id=2153.
- [60] Seagate. *The Transition to Advanced Format 4K Sector Hard Drives*. Tech. rep. Last accessed 04-05-2022. Seagate LLC, Apr. 2010. URL: https://web.archive.org/web/20110902031330/http://seagate.com/docs/pdf/whitepaper/tp613_transition_to_4k_sectors.pdf.

- [61] G. Rodrigues. *Linux and 4K disk sectors*. Last accessed 04-05-2022. Mar. 2009. URL: <https://lwn.net/Articles/322777/>.
- [62] M. Karls. *Hard Drive Error Correcting Code (ECC)*. Last accessed 17-09-2022. Nov. 2018. URL: <https://www.karlstechnology.com/blog/hard-drive-error-correcting-code-ecc/>.
- [63] Data Clinic Ltd. *Hard drive defects table – P & G Lists*. Last accessed 17-09-2022. 2020. URL: <https://www.dataclinic.co.uk/hard-drive-defects-table/>.
- [64] Datarecovery.com. *What are P-Lists and G-Lists?* Last accessed 30-11-2021. Jan. 2016. URL: <https://datarecovery.com/rd/what-are-p-lists-and-g-lists/>.
- [65] R. Sheldon. *overprovisioning (SSD overprovisioning)*. Last accessed 17-09-2022. Apr. 2022. URL: <https://www.techtarget.com/searchstorage/definition/overprovisioning-SSD-overprovisioning>.
- [66] A. Tanenbaum and H. Bos. *Modern Operating Systems*. 4th. Upper Saddle River, New Jersey, USA: Pearson Education Inc., 2015.
- [67] U. Inc. *Unicode® Version 15.0 Character Counts*. Last accessed 03-05-2021. 2022. URL: <https://www.unicode.org/versions/stats/>.
- [68] U. Inc. *Unicode provides a unique number for every character, no matter what the platform, program, or language is*. Last accessed 03-05-2022. 2021. URL: <https://home.unicode.org/basic-info/overview/>.
- [69] Microsoft. *How NTFS Works*. Last accessed 30-09-2018. 2018. URL: [https://technet.microsoft.com/pt-pt/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx).
- [70] D. Poirier. *The Second Extended File System: Internal Layout*. Last accessed 23-03-2023. Dave Poirier, 2019. URL: <https://www.nongnu.org/ext2-doc/ext2.pdf>.
- [71] P. Cobbaut. *Linux Storage*. Last accessed 23-03-2023. Paul Cobbaut, 2015. URL: <https://linux-training.be/linuxsto.pdf>.
- [72] Micron Technology Inc. *Should You Defrag an SSD?* Last accessed 06-05-2022. 2020. URL: <https://www.crucial.com/articles/about-ssd/should-you-defrag-an-ssd>.
- [73] H. Sencar and N. Memon, eds. *Digital Image Forensics — There is More to a Picture than Meets the Eye*. Springer, New York, NY, 2013.

Bibliography

- [74] A. Pal and N. Memon. “The evolution of file carving.” In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [75] A. Pal, H. T. Sencar, and N. Memon. “Detecting file fragmentation point using sequential hypothesis testing.” In: *Digital Investigation* 5.Supplement (2008). The Proceedings of the Eighth Annual DFRWS Conference, S2–S13. ISSN: 1742-2876. DOI: 10.1016/j.diin.2008.05.015.
- [76] W. Stallings. *Operating Systems — Internals and Design Principles*. 7th. Upper Saddle River, New Jersey, USA: Pearson Education Inc., 2012.
- [77] F. Darnowski and A. Chojnacki. “Writing and Deleting files on hard drives with NTFS.” In: *Computer Science and Mathematical Modelling* 8 (2018), pp. 5–15. DOI: 10.5604/01.3001.0013.1457.
- [78] M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Disk Cluster Allocation Behavior in Windows and NTFS.” In: *Mobile Networks and Applications* 25.1 (Feb. 2020), pp. 248–258. ISSN: 1572-8153. DOI: 10.1007/s11036-019-01441-1.
- [79] H. Liang and S. Xu. *Locate and correct disk space problems on NTFS volumes*. Last accessed 14-01-2023. Sept. 2021. URL: <https://learn.microsoft.com/en-us/troubleshoot/windows-server/backup-and-storage/disk-space-problems-on-ntfs-volumes>.
- [80] J. Hughes. *The Four Stages of NTFS File Growth*. Last accessed 24-10-2018. Oct. 2009. URL: <https://blogs.technet.microsoft.com/askcore/2009/10/16/the-four-stages-of-ntfs-file-growth/>.
- [81] Jeff, phuclv, user3685427, GabrielB, and A. Lazzarotto. *Maximum size of file that can be stored entirely in NTFS Master File Table (MFT)*. Last accessed 28-08-2022. May 2022. URL: <https://superuser.com/questions/1185461/maximum-size-of-file-that-can-be-stored-entirely-in-ntfs-master-file-table-mft>.
- [82] D. Sedory. *An Introduction to NTFS*. Last accessed 28-08-2022. 2021. URL: <https://thestarman.pcministry.com/asm/mbr/IntNTFSfs.htm>.
- [83] jaclaz, Patrick4n6, keydet89, joakims, randomaccess, Chris_Ed, and pbobby. *\$MFT Resident data*. Last accessed 28-08-2022. Mar. 2013. URL: <https://www.forensicfocus.com/forums/general/mft-resident-data/>.
- [84] A. A. Bahjat and J. Jones. “Deleted file fragment dating by analysis of allocated neighbors.” In: *Digital Investigation* 28 (2019), S60–S67. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.01.015.

- [85] maaartinus, L. Osterman, and PC Guru. *What block allocation algorithm does NTFS use?* Last accessed 24-01-2019. Mar. 2017. URL: <https://superuser.com/questions/274855/what-block-allocation-algorithm-does-ntfs-use>.
- [86] R. Benadjila, L. Khati, and D. Vergnaud. “Secure storage – Confidentiality and authentication.” In: *Computer Science Review* 44 (2022), p. 100465. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2022.100465.
- [87] M. Halcrow. “eCryptfs: a Stacked Cryptographic Filesystem.” In: *Linux Journal* (Apr. 2007). Last accessed 08-03-2023. URL: <https://www.linuxjournal.com/article/9400>.
- [88] R. Bragg. *The Encrypting File System*. Last accessed 10-03-2023. June 2009. URL: [https://learn.microsoft.com/en-us/previous-versions/tn-archive/cc700811\(v=technet.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/tn-archive/cc700811(v=technet.10)?redirectedfrom=MSDN).
- [89] P. Matarazzo, A. Czechowski, L. Long, V. Raju, J. Mathew, W. Bjorn, S. Mandalika, M. Athavale, D. Vangel, T. McNaboe, D. Coulter, D. Simpson, T. Jiuding, N. Schonning, A. Gorzelany, L. Poggemeyer, M. Avedon, J. Hall, E. Gallagher, A. Bichsel, and D. Mackenzie. *Trusted Platform Module Technology Overview*. Last accessed 28-02-2023. Feb. 2023. URL: <https://learn.microsoft.com/en-us/windows/security/information-protection/tpm/trusted-platform-module-overview>.
- [90] L. Khati. “Full Disk Encryption and Beyond.” PhD thesis. Université PSL; Ecole Normale Supérieure de Paris, Oct. 2019.
- [91] D. Harnik, O. Naor, E. Ofer, and O. Ozery. “Rethinking block storage encryption with virtual disks.” In: *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems* (2022). DOI: 10.1145/3538643.3539748.
- [92] L. Khati, N. Mouha, and D. Vergnaud. “Full Disk Encryption: Bridging Theory and Practice.” In: *Topics in Cryptology - CT-RSA 2017 - The Cryptographers’ Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*. Ed. by H. Handschuh. Vol. 10159. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 241–257. ISBN: 978-3-319-52153-4. DOI: 10.1007/978-3-319-52153-4_14.
- [93] L. Khati, N. Mouha, and D. Vergnaud. “Full Disk Encryption: Bridging Theory and Practice.” In: *IACR Cryptol. ePrint Arch.* (2016), p. 1114. URL: <http://eprint.iacr.org/2016/1114>.

Bibliography

- [94] F. Rojas, P. Matarazzo, L. Long, M. Ohlinger, S. Mandalika, J. Mathew, D. Simpson, M. Athavale, B. Shilpa, D. Vangel, M. Mardahl, A. Czechowski, B. Dharmanayagam, D. Coulter, M. Mussabekov, L. Keller, Onur, F. Reichmann, A. Gorzelany, J. Hall, and L. Poggemeyer. *Overview of BitLocker Device Encryption in Windows*. Last accessed 02-03-2023. Feb. 2023. URL: <https://learn.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-device-encryption-overview-windows-10>.
- [95] Microsoft. *Feature-specific requirements for Windows 10*. Last accessed 02-03-2023. 2023. URL: <https://www.microsoft.com/en-gb/windows/windows-10-specifications#primaryR5>.
- [96] JdeBP and T. Zych. *Why does Windows7 create two partitions?* Last accessed 20-02-2023. July 2018. URL: <https://superuser.com/questions/330178/why-does-windows7-create-two-partitions>.
- [97] IBM. *What is virtualization?* Last accessed 15-03-2023. URL: <https://www.ibm.com/topics/virtualization>.
- [98] IBM. *What are hypervisors?* Last accessed 02-04-2023. URL: <https://www.ibm.com/topics/hypervisors>.
- [99] Oracle Corporation. *VirtualBox – Changelog for VirtualBox 7.0*. Last accessed 19-02-2023. Jan. 2023. URL: <https://www.virtualbox.org/wiki/Changelog>.
- [100] Oracle Corporation. *Oracle® VM VirtualBox® User Manual*. Version 7.0.6. Last accessed 28-03-2023. 2023. URL: <https://download.virtualbox.org/virtualbox/7.0.6/UserManual.pdf>.
- [101] W. McMillen. *Trim Command – General Benefits for Hard Disk Drives*. White paper. Last accessed 12-03-2023. Western Digital Corporation, Dec. 2021.
- [102] Oracle Corporation. *8.26. VBoxManage storageattach*. Last accessed 11-02-2023. URL: <https://www.virtualbox.org/manual/ch08.html#vboxmanage-storageattach>.
- [103] K. Szynter, glglgl, J. Rowe, D. SMogor, bobpaul, F. Labs, Omnifarious, J. Larsen, davidbaumann, davidtgq, RobM, and xpusostomos. *VirtualBox and SSD's TRIM command support*. Last accessed 13-03-2023. May 2021. URL: <https://superuser.com/questions/646559/virtualbox-and-ssds-trim-command-support>.
- [104] E. Wramner. *Shrink VirtualBox VDI files with TRIM*. Last accessed 13-03-2023. Oct. 2017. URL: <https://erikwramner.wordpress.com/2017/10/17/shrink-virtualbox-vdi-files-with-trim/>.

- [105] F. Ormeling. “Map Use and Map Reading.” In: Stockholm, Sweden: International Cartographic Association, 2014. Chap. 2.
- [106] B. Rystedt and F. Ormeling, eds. *The World of Maps*. Stockholm, Sweden: International Cartographic Association, 2014.
- [107] B. Markoski. *Basic Principles of Topography*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Nature Switzerland AG, 2018.
- [108] K. Kriz, W. Cartwright, and L. Hurni, eds. *Mapping Different Geographies*. Springer-Verlag Berlin Heidelberg, 2010.
- [109] A. Miall. *Stratigraphy: A Modern Synthesis*. 2nd. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Nature Switzerland AG, 2022.
- [110] M. Karresand and N. Shahmehri. “Reassembly of fragmented JPEG images containing restart markers.” In: *Proceedings - 4th Annual European Conference on Computer Network Defense, EC2ND 2008*. 2008, pp. 25–32. DOI: 10.1109/EC2ND.2008.10.
- [111] M. Karresand and N. Shahmehri. “Oscar — Using Byte Pairs to Find File Type and Camera Make of Data Fragments.” In: *Proceedings of the 2nd European Conference on Computer Network Defence, in conjunction with the First Workshop on Digital Forensics and Incident Analysis (EC2ND 2006)*. Ed. by A. Blyth and I. Sutherland. Springer Verlag, 2007, pp. 85–94. DOI: 10.1007/978-1-84628-750-3_9.
- [112] M. Karresand and N. Shahmehri. “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*. Piscataway, NJ, USA: IEEE, 2006, pp. 140–147. DOI: 10.1109/IAW.2006.1652088.
- [113] M. Karresand and N. Shahmehri. “Oscar — File Type and Camera Identification Using the Structure of Binary Data Fragments.” In: *Proceedings of the 1st Conference on Advances in Computer Security and Forensics, ACSF*. Ed. by J. Haggerty and M. Merabti. Liverpool, UK: The School of Computing and Mathematical Sciences, John Moores University, July 2006, pp. 11–20.
- [114] M. Karresand and N. Shahmehri. “Oscar — File Type Identification of Binary Data in Disk Clusters and RAM Pages.” In: *Security and Privacy in Dynamic Environments, Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22-24 May 2006, Karlstad, Sweden*. Vol. 201. Lecture Notes in Computer Science. Springer, 2006, pp. 413–424. DOI: 10.1007/0-387-33406-8_35.

- [115] R. Poisel, M. Rybnicek, and S. Tjoa. "Taxonomy of Data Fragment Classification Techniques." In: *Digital Forensics and Cyber Crime: Fifth International Conference, ICDf2C 2013, Moscow, Russia, September 26-27, 2013, Revised Selected Papers*. Ed. by P. Gladyshev, A. Marrington, and I. Baggili. Springer International Publishing, 2014, pp. 67–85. DOI: 10.1007/978-3-319-14289-0_6.
- [116] A. Tridgell. *Spamsum README*. Last accessed 27-04-2018. 2002. URL: <https://www.samba.org/ftp/unpacked/junkcode/spamsum/README>.
- [117] J. Kornblum. "Identifying almost identical files using context triggered piecewise hashing." In: *Digital Investigation 3*. Supplement (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), pp. 91–97. ISSN: 1742-2876. DOI: 10.1016/j.diin.2006.06.015.
- [118] Y. Dandass, N. Necaïse, and S. Thomas. "An Empirical Analysis of Disk Sector Hashes for Data Carving." In: *J. Digit. Forensic Pract.* 2.2 (Apr. 2008), pp. 95–104. ISSN: 1556-7281. DOI: 10.1080/15567280802050436.
- [119] N. Canceill. "Random Sampling applied to Rapid Disk Analysis." MA thesis. University of Amsterdam, The Netherlands, July 2013.
- [120] J. Taguchi. "Optimal Sector Sampling for Drive Triage." MA thesis. Monterey, CA 93943, USA: Naval Postgraduate School, June 2013.
- [121] M. Hirano, H. Takase, and K. Yoshida. "Evaluation of a Sector-Hash Based Rapid File Detection Method for Monitoring Infrastructure-as-a-Service Cloud Platforms." In: *2015 10th International Conference on Availability, Reliability and Security*. 2015, pp. 584–591. DOI: 10.1109/ARES.2015.15.
- [122] F. Gutierrez-Villarreal. "Improving sector hash carving with rule-based and entropy-based non-probative block filters." MA thesis. Monterey, California, USA: Naval Postgraduate School, Mar. 2015.
- [123] V. Roussev and S. Garfinkel. "File Fragment Classification-The Case for Specialized Approaches." In: *2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*. May 2009, pp. 3–14. DOI: 10.1109/SADFE.2009.21.
- [124] J. Garcia. "Duplications and Misattributions of File Fragment Hashes in Image and Compressed Files." In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 2018, pp. 1–5. DOI: 10.1109/NTMS.2018.8328690.

- [125] C. Veenman. “Statistical Disk Cluster Classification for File Carving.” In: *Proceedings of the Third International Symposium on Information Assurance and Security, 2007 (IAS 2007)*. Ed. by N. Zhang, A. Abraham, Q. Shi, and J. Thomas. IEEE Computer Society, 2007, pp. 393–398. DOI: 10.1109/ISIAS.2007.4299805.
- [126] W. Calhoun and D. Coles. “Predicting the types of file fragments.” In: *Digital Investigation* 5. Supplement 1 (Sept. 2008), S14–S20. DOI: 10.1016/j.diin.2008.05.005.
- [127] I. Ahmed, K.-s. Lhee, H. Shin, and M. Hong. “On Improving the Accuracy and Performance of Content-Based File Type Identification.” In: *Proc. ACISP 2009*. Ed. by C. Boyd and G. Nieto. Vol. 5594/2009. LNCS. Springer-Verlag Berlin Heidelberg, 2009, pp. 44–59. DOI: 10.1007/978-3-642-02620-1_4.
- [128] Q. Li, A. Ong, P. Suganthan, and V. Thing. “A Novel Support Vector Machine Approach to High Entropy Data Fragment Classification.” In: *South African Information Security Multi-Conference, SAISMC 2010, Port Elizabeth, South Africa, May 17-18, 2010. Proceedings*. Ed. by N. Clarke, S. Furnell, and R. Solms. University of Plymouth, 2010, pp. 236–247.
- [129] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn. “Using NLP techniques for file fragment classification.” In: *Digital Investigation* 9 (2012). The Proceedings of the Twelfth Annual DFRWS Conference, S44–S49. ISSN: 1742-2876. DOI: 10.1016/j.diin.2012.05.008.
- [130] M. Bhatt, A. Mishra, M. Kabir, S. Blake-Gatto, R. Rajendra, M. Hoque, and I. Ahmed. “Hierarchy-Based File Fragment Classification.” In: *Machine Learning and Knowledge Extraction* 2.3 (2020), pp. 216–232. ISSN: 2504-4990. DOI: 10.3390/make2030012. URL: <https://www.mdpi.com/2504-4990/2/3/12>.
- [131] A. Bhat, A. Likhite, S. Chavan, and L. Ragha. “File Fragment Classification using Content Based Analysis.” In: *ITM Web Conf.* 40 (2021), p. 03025. DOI: 10.1051/itmconf/20214003025.
- [132] R. Ali and K. Mohamad. “RX_myKarve carving framework for reassembling complex fragmentations of JPEG images.” In: *Journal of King Saud University — Computer and Information Sciences* 33.1 (2021), pp. 21–32. ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2018.12.007. URL: <https://www.sciencedirect.com/science/article/pii/S131915781831070X>.

Bibliography

- [133] DFRWS. *DFRWS 2006 Forensics Challenge Results*. Last accessed 21-08-2022. 2016. URL: <http://old.dfrws.org/2006/challenge/results.shtml>.
- [134] Microsoft. *Default cluster size for NTFS, FAT, and exFAT*. Aug. 2015. URL: <https://support.microsoft.com/en-us/help/140365/default-cluster-size-for-ntfs-fat-and-exfat>.
- [135] E. Casey. "Arson, Archaeology, and Computer Crime Investigation." In: *Computer Fraud & Security* 2003.7 (2003), pp. 12–15. ISSN: 1361-3723. DOI: 10.1016/S1361-3723(03)07011-8.
- [136] P. Gladyshev and A. Patel. "Finite state machine approach to digital event reconstruction." In: *Digital Investigation* 1.2 (2004), pp. 130–149. ISSN: 1742-2876. DOI: 10.1016/j.diin.2004.03.001.
- [137] L. Jun and Z. Guo. "Time Bounding Event Reasoning in Computer Forensic." In: *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)*. 2007, pp. 946–952. DOI: 10.1109/CISW.2007.4425652.
- [138] E. Casey. *Digital Evidence and Computer Crime*. 3rd ed. Elsevier Inc. (Academic Press), 2011.
- [139] E. Casey. "Reconstructing Digital Evidence." In: *Crime Reconstruction*. Ed. by W. J. Chisum and B. E. Turvey. 2nd ed. San Diego: Academic Press, 2011. Chap. 17, pp. 531–548. ISBN: 978-0-12-386460-4. DOI: 10.1016/B978-0-12-386460-4.00017-5.
- [140] F. Darnowski and A. Chojnacki. "Selected Methods of File Carving and Analysis of Digital Storage Media in Computer Forensics." In: *Teleinformatics Review* 1–2 (2015), pp. 25–40.
- [141] C. Harfield and J. Schofield. "(Im)material culture: towards an archaeology of cybercrime." In: *World Archaeology* 52.4 (2020), pp. 607–618. DOI: 10.1080/00438243.2021.1882333.
- [142] S. Ross and A. Gow. *Digital Archaeology: Rescuing Neglected and Damaged Data Resources*. Tech. rep. Last accessed 15-05-2022. Humanities Advanced Technology and Information Institute (HATII), University of Glasgow, UK, Feb. 1999. URL: <https://purehost.bath.ac.uk/ws/portalfiles/portal/11350174/p2.pdf>.
- [143] D. Farmer and W. Venema. *Forensic Discovery*. Pearson Education, Inc., 2005.
- [144] M. Graves. *Digital Archaeology — The Art and Science of Digital Forensics*. Pearson Education, Inc., 2013.

- [145] M. Pollitt. “History, Historiography and the Hermeneutics of the Hard Drive.” In: *Advances in Digital Forensics IX*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 3–17. ISBN: 978-3-642-41148-9. DOI: 10.1007/978-3-642-41148-9_1.
- [146] K. Smith and M. Seltzer. “File System Aging—Increasing the Relevance of File System Benchmarks.” In: *SIGMETRICS Perform. Eval. Rev.* 25.1 (June 1997), pp. 203–213. ISSN: 0163-5999. DOI: 10.1145/258623.258689.
- [147] J. Douceur and W. Bolosky. “A Large-Scale Study of File-System Contents.” In: *SIGMETRICS Perform. Eval. Rev.* 27.1 (May 1999), pp. 59–70. ISSN: 0163-5999. DOI: 10.1145/301464.301480.
- [148] S. Garfinkel. “Carving contiguous and fragmented files with fast object validation.” In: *Digital Investigation* 4.Supplement (2007), S2–S12. ISSN: 1742-2876. DOI: 10.1016/j.diin.2007.06.017.
- [149] S. L. Garfinkel. “Forensic feature extraction and cross-drive analysis.” In: *Digital Investigation* 3.Supplement (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS ’06), pp. 71–81. ISSN: 1742-2876. DOI: 10.1016/j.diin.2006.06.007.
- [150] M. Cohen. “Advanced carving techniques.” In: *Digital Investigation* 4.3 (2007), pp. 119–128. ISSN: 1742-2876. DOI: 10.1016/j.diin.2007.10.001.
- [151] PassMark Software. *File Fragmentation Tool*. Last accessed 05-01-2022. Nov. 2009. URL: <https://www.passmark.com/products/fragger/>.
- [152] D. Meyer and W. Bolosky. “A Study of Practical Deduplication.” In: *ACM Trans. Storage* 7.4 (Feb. 2012). ISSN: 1553-3077. DOI: 10.1145/2078861.2078864.
- [153] R. Pahade, B. Singh, and U. Singh. “A Survey on Multimedia File Carving.” In: *International Journal of Computer Science and Engineering Survey (IJCSSES)* 6.6 (Dec. 2015), pp. 27–46. DOI: 10.5121/ijcses.2015.6603.
- [154] V. van der Meer, H. Jonker, G. Dols, H. van Beek, J. van den Bos, and M. van Eekelen. “File Fragmentation in the Wild: a Privacy-Friendly Approach.” In: *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*. 2019, pp. 1–6. DOI: 10.1109/WIFS47025.2019.9034981.
- [155] G.-S. Cho. “An Arbitrary Disk Cluster Manipulating Method for Allocating Disk Fragmentation of Filesystem.” In: *Journal of Korea Society of Digital Industry and Information Management* 16.2 (June 2020). The article is in Korean, pp. 11–25. DOI: 10.17662/ksdim.2020.16.2.011.

Bibliography

- [156] S. Willassen. “Finding Evidence of Antedating in Digital Investigations.” In: *2008 Third International Conference on Availability, Reliability and Security*. Mar. 2008, pp. 26–32. DOI: 10.1109/ARES.2008.149.
- [157] F. Crispino. “Le principe de Locard est-il scientifique? Ou analyse de la scientificité des principes fondamentaux de la criminalistique.” PhD thesis. Institut de Police Scientifique, Ecole des Sciences Criminelles, Faculte de Droit, Université de Lausanne, May 2006.
- [158] S. Dobrowski. *Forensic Fact of the Day - Quotations - Edmond Locard*. Jan. 2013. URL: <https://castleviewuk.com/blog/index.php?forensic-fact-of-the-day---quotations---edmond-locard>.
- [159] S. Wilding. *Locard's Exchange Principle*. Last accessed 04-04-2023. 2012. URL: <http://www.forensichandbook.com/locards-exchange-principle/>.
- [160] M. Andrew. “Defining a Process Model for Forensic Analysis of Digital Devices and Storage Media.” In: *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*. 2007, pp. 16–30. DOI: 10.1109/SADFE.2007.8.
- [161] K. Zatyko and J. Bay. “The Digital Forensics Cyber Exchange Principle.” In: *Forensic Magazine* 8.6 (2011).
- [162] S. Willassen. “Methods for Enhancement of Timestamp Evidence in Digital Investigations.” PhD thesis. Norwegian University of Science, Technology, Faculty of Information Technology, Mathematics, and Electrical Engineering, Department of Telematics, Jan. 2008.
- [163] A. Chojnacki and F. Darnowski. “A model of the process of writing and deleting file information on a disk with NTFS.” In: *Computer Science and Mathematical Modelling* 9 (2019), pp. 19–25. DOI: 10.5604/01.3001.0013.6602.
- [164] J. Bouma. “Interpreting NTFS Time-stamps.” MA thesis. Open University of the Netherlands, Aug. 2019.
- [165] V. Ghotge and P. Nema. *Description of the Cluster Preallocation Algorithm in the NTFS File System*. Tech. rep. KB 841551. No first hand version available. Microsoft, 2004.
- [166] S. Bader. *What is data recovery and how does it work?* Last accessed 09-08-2022. May 2022. URL: <https://rewind.com/blog/what-is-data-recovery/>.

- [167] SysDev Laboratories LLC. *What is data recovery?* Last accessed 09-08-2022. Sept. 2021. URL: <https://www.ufsexplorer.com/articles/what-is-data-recovery.php>.
- [168] R. McKemmish. “When is Digital Evidence Forensically Sound?” In: *Advances in Digital Forensics IV*. Ed. by I. Ray and S. Sheno. Boston, MA: Springer US, 2008, pp. 3–15. ISBN: 978-0-387-84927-0. DOI: 10.1007/978-0-387-84927-0_1.
- [169] ninad. *Top 20 Best Linux Data Recovery Tools to Recover Deleted/Corrupted Files*. Last accessed 06-08-2022. Aug. 2022. URL: <https://www.digitalocean.com/community/tutorials/top-best-linux-data-recovery-tools>.
- [170] T. Fisher. *21 Best Free Data Recovery Software Tools*. Last accessed 06-08-2022. Aug. 2022. URL: <https://www.lifewire.com/free-data-recovery-software-tools-2622893>.
- [171] I. Haynes. *Top 5 Best Open-Source Data Recovery Software in 2022 (For Your Business or Personal Purposes!)* Last accessed 06-08-2022. Feb. 2022. URL: <https://www.handyrecovery.com/best-open-source-data-recovery-software/>.
- [172] B. Cornec. *Mondo Rescue Home Page*. Last accessed 06-08-2022. Jan. 2020. URL: <http://www.mondorescue.org/>.
- [173] antonio. *Ddrescue - Data recovery tool*. Last accessed 06-08-2022. Jan. 2022. URL: <https://www.gnu.org/software/ddrescue/>.
- [174] *Foremost*. Last accessed 06-08-2022. URL: <http://foremost.sourceforge.net/>.
- [175] C. Corax. *safecopy*. Last accessed 06-08-2022. Mar. 2012. URL: <http://safecopy.sourceforge.net/>.
- [176] N. Case. *About extundelete*. Last accessed 06-08-2022. 2013. URL: <http://extundelete.sourceforge.net/>.
- [177] C. Grenier. *TestDisk Documentation — Release 7.1*. Last accessed 14-05-2022. Mar. 2022. URL: <https://www.cgsecurity.org/testdisk.pdf>.
- [178] C. Grenier. *TestDisk, Data Recovery*. Last accessed 06-08-2022. Oct. 2019. URL: <https://www.cgsecurity.org/wiki/TestDisk>.
- [179] C. Grenier. *PhotoRec, Digital Picture and File Recovery*. Last accessed 06-08-2022. July 2019. URL: <https://www.cgsecurity.org/wiki/PhotoRec>.

Bibliography

- [180] G. Richard III and L. Marziale. *sleuthkit/scalpel*. Last accessed 06-08-2022. Mar. 2021. URL: <https://github.com/sleuthkit/scalpel>.
- [181] G. Richard III and V. Roussev. “Scalpel: a frugal, high performance file carver.” In: *Proceedings of the Fifth Annual DFRWS Conference*. Last accessed 29-11-2017. Aug. 2005, pp. 1–10. URL: https://www.dfrws.org/sites/default/files/session-files/paper-scalpel_-_a_frugal_high_performance_file_carver.pdf.
- [182] EaseUS. *EaseUS Data Recovery Wizard*. Last accessed 06-08-2022. 2022. URL: <https://www.easeus.com/datarecoverywizard/free-data-recovery-software.htm>.
- [183] 508 Software LLC. *Disk Drill Data Recovery Software*. Last accessed 06-08-2022. 2022. URL: <https://www.cleverfiles.com/data-recovery-software.html>.
- [184] Piriform Software Ltd. *Recuva*. Last accessed 06-08-2022. 2022. URL: <https://www.ccleaner.com/recuva>.
- [185] Atea Ataroa Limited. *Distrowatch.com — Put the fun back in computing*. Last accessed 07-08-2022. 2022. URL: <https://distrowatch.com/>.
- [186] OffSec Services Limited. *The most advanced Penetration Testing Distribution*. Last accessed 07-08-2022. 2022. URL: <https://www.kali.org/>.
- [187] A. Linux. *Rescue*. Last accessed 07-08-2022. Dec. 2021. URL: <https://en.altlinux.org/Rescue>.
- [188] R. Finnie. *Finnix*. Last accessed 07-08-2022. Mar. 2022. URL: <https://www.finnix.org/>.
- [189] The Grml Team. *Grml Live Linux*. Last accessed 07-08-2022. 2022. URL: <https://grml.org/>.
- [190] K. Chevreuil and T. Bequet. *Welcome to Kaisen Linux — The distribution for professional IT*. Last accessed 07-08-2022. 2022. URL: <https://kaisenlinux.org/>.
- [191] F. Dupoux, G. Egidy, and M. Mello. *System Rescue Homepage*. Last accessed 07-08-2022. May 2022. URL: <https://www.system-rescue.org/>.
- [192] Clonezilla. *Clonezilla — The Free and Open Source Software for Disk Imaging and Cloning*. Last accessed 07-08-2022. 2022. URL: <https://clonezilla.org/>.
- [193] GNOME Partition Editor. *Live CD/USB/PXE/HD*. Last accessed 07-08-2022. July 2022. URL: <https://gparted.org/livecd.php>.

- [194] Parted Magic LLC. *Powerful Tools for Home or Office! — Parted Magic is a complete hard disk management solution*. Last accessed 07-08-2022. 2022. URL: <https://partedmagic.com/>.
- [195] E. Hanlhofer. *Latest news*. Last accessed 07-08-2022. July 2022. URL: <https://www.plop.at/en/home.html>.
- [196] S. Key. *File Block Hash Map Analysis*. Last accessed 28-04-2018. 2012. URL: <https://www.guidancesoftware.com/app/File-Block-Hash-Map-Analysis>.
- [197] Open Text Corporation. *OpenText EnCase Forensic*. Last accessed 06-02-2022. 2021. URL: <https://security.opentext.com/encase-forensic>.
- [198] K. Woods and C. Lee. “Acquisition and processing of disk images to further archival goals.” In: *Proceedings of Archiving 2012*. Society for Imaging Science and Technology, 2012, pp. 147–152.
- [199] H. van Beek, E. van Eijk, R. van Baar, M. Ugen, J. Bodde, and A. Siemelink. “Digital forensics as a service: Game on.” In: *Digital Investigation* 15 (2015). Special Issue: Big Data and Intelligent Data Analysis, pp. 20–38. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.07.004.
- [200] R. van Baar, H. van Beek, and E. van Eijk. “Digital Forensics as a Service: A game changer.” In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S54–S62. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.03.007.
- [201] B. Schatz. *System and Method for Simultaneous Forensic, Acquisition, Examination and Analysis of a Computer Readable Medium at Wirespeed*. United States Patent No. 10,354,062 B2. July 2019.
- [202] M. Russinovich. *DiskView v2.41*. Last accessed 27-02-2023. Oct. 2020. URL: <https://learn.microsoft.com/en-us/sysinternals/downloads/diskview>.
- [203] markruss, A. Mihaiuc, J. Stephens, A. Buck, P. Yosifovich, L. Kim, analyze-v, M. Russinovich, M. Polla, and chadmando. *Sysinternals*. Last accessed 05-03-2023. Dec. 2022. URL: <https://learn.microsoft.com/en-us/sysinternals/>.
- [204] B. Carrier. *TSK Tool Overview*. 2014. URL: http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.
- [205] D. Kandakatla and J. Danyow. *Windows 10 release information*. Last accessed 18-03-2023. Mar. 2023. URL: <https://learn.microsoft.com/en-us/windows/release-health/release-information>.

Bibliography

- [206] S. White, jun-yo, B. Schofield, J. Walker, D. Coulter, drew batchelor, M. Jacobs, and M. Satran. *Operating System Version*. Last accessed 18-03-2023. Nov. 2021. URL: <https://learn.microsoft.com/en-us/windows/win32/sysinfo/operating-system-version>.
- [207] V. Vozenilek. "MAP DESIGN." In: Stockholm, Sweden: International Cartographic Association, 2014. Chap. 4.
- [208] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. "The first collision for full SHA-1." In: *Advances in Cryptology — CRYPTO 2017*. Vol. 10401. Lecture Notes in Computer Science. 2017, pp. 570–596. DOI: 10.1007/978-3-319-63688-7_19.
- [209] Cryptology Group at Centrum Wiskunde & Informatica (CWI) and Google Research Security, Privacy and Anti-abuse Group. *Shattered — We have broken SHA-1 in practice*. Last accessed 28-04-2018. URL: <https://shattered.io/>.
- [210] Real Data Corpus. *Real Data Corpus*. Last accessed 29-09-2018. July 2018. URL: <https://digitalcorpora.org/corpora/disk-images/real-data-corpus>.
- [211] S. Gibbs. *From Windows 1 to Windows 10: 29 years of Windows evolution*. Last accessed 29-09-2018. Oct. 2014. URL: <https://www.theguardian.com/technology/2014/oct/02/from-windows-1-to-windows-10-29-years-of-windows-evolution>.
- [212] C. Buckel. *Understanding Flash: Blocks, Pages and Program Erases*. Last accessed 03-10-2018. 2014. URL: <https://flashdba.com/2014/06/20/understanding-flash-blocks-pages-and-program-erases/>.
- [213] Y. Gubanov and O. Afonin. *SSD and eMMC Forensics 2016, part 1*. Last accessed 07-10-2018. Apr. 2016. URL: <https://articles.forensicfocus.com/2016/04/20/ssd-and-emmc-forensics-2016/>.
- [214] Y. Gubanov and O. Afonin. *SSD and eMMC Forensics 2016, part 2*. Last accessed 07-10-2018. May 2016. URL: <https://articles.forensicfocus.com/2016/05/04/ssd-and-emmc-forensics-2016-part-2/>.
- [215] Y. Gubanov and O. Afonin. *SSD and eMMC Forensics 2016, part 3*. Last accessed 07-10-2018. June 2016. URL: <https://articles.forensicfocus.com/2016/06/07/ssd-and-emmc-forensics-2016-part-3/>.

- [216] Y. Gubanov and O. Afonin. *Recovering Evidence from SSD Drives in 2014: Understanding TRIM, Garbage Collection and Exclusions*. Last accessed 07-10-2018. 2014. URL: <https://articles.forensicfocus.com/2014/09/23/recovering-evidence-from-ssd-drives-in-2014-understanding-trim-garbage-collection-and-exclusions/>.
- [217] Y. Gubanov and O. Afonin. *Why SSD Drives Destroy Court Evidence, and What Can Be Done About It*. Last accessed 08-10-2018. 2012. URL: <https://belkasoft.com/download/info/SSD%20Forensics%202012.pdf>.
- [218] Swedish Defence Research Agency. *CRATE – Sweden’s national cyber training facility*. Last accessed 26-03-2023. Sept. 2022. URL: <https://www.foi.se/en/foi/research/information-security/crate---swedens-national-cyber-training-facility.html>.
- [219] Oracle Corporation. *Chapter 5. Virtual Storage*. Last accessed 18-03-2023. 2022. URL: <https://www.virtualbox.org/manual/ch05.html>.
- [220] TerryE and mpack. *All about VDIs*. Last accessed 30-12-2018. Feb. 2018. URL: <https://forums.virtualbox.org/viewtopic.php?t=8046>.
- [221] Oracle Corporation. *Oracle® VM VirtualBox® User Manual*. Version 5.2.20. Last accessed 26-03-2023. 2018. URL: <https://download.virtualbox.org/virtualbox/5.2.20/UserManual.pdf>.
- [222] Arch Linux. *Disk encryption*. Last accessed 30-12-2018. Nov. 2018. URL: https://wiki.archlinux.org/index.php/disk_encryption.
- [223] M. Gattol. *Block-layer Encryption*. Last accessed 24-01-2019. Jan. 2015. URL: https://web.archive.org/web/20150917051251/http://www.markus-gattol.name/ws/dm-crypt_luks.html.
- [224] K. Scarfone, M. Souppaya, and M. Sexton. *Guide to Storage Encryption Technologies for End User Devices*. Tech. rep. 800-111. National Institute of Standards and Technology, Nov. 2007.
- [225] Aorimn. *dislocker*. Sept. 2021. URL: <https://github.com/Aorimn/dislocker>.
- [226] Der_Meister, That Brazilian Guy, and31415, Techie007, and Pacerier. *Is it safe to delete the System Volume Information folder?* Last accessed 29-03-2023. May 2015. URL: <https://superuser.com/questions/763165/is-it-safe-to-delete-the-system-volume-information-folder>.
- [227] Guest and Pondus. *Avast detected file SYMEFA.DB as a virus on my computer*. Last accessed 29-03-2023. Jan. 2015. URL: <https://forum.avast.com/index.php?topic=164221.0>.

Bibliography

- [228] Anonymous users and Brian. *Prevent EfaData/SYMEFA.DB from being created?* Last accessed 29-03-2023. Apr. 2013. URL: <https://community.broadcom.com/symantecenterprise/communities/community-home/digestviewer/viewthread?MessageKey=9d963f34-0db3-41e9-b5bd-dd6c066e59ae&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=digestviewer>.
- [229] J. Metz. *BitLocker Drive Encryption (BDE) format specification*. Last accessed 29-03-2023. Feb. 2022. URL: [https://github.com/libyal/libbde/blob/main/documentation/BitLocker%20Drive%20Encryption%20\(BDE\)%20format.asciidoc](https://github.com/libyal/libbde/blob/main/documentation/BitLocker%20Drive%20Encryption%20(BDE)%20format.asciidoc).
- [230] M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8.
- [231] M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Using NTFS Cluster Allocation Behavior to Find the Location of User Data.” In: *Digital Investigation 29* (2019), S51–S60. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.018.
- [232] M. Karresand, G. Dyrkolbotn, and S. Axelsson. “An Empirical Study of the NTFS Cluster Allocation Behavior Over Time.” In: *Forensic Science International: Digital Investigation 33* Supplement (July 2020), p. 301008. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2020.301008.

Part II.

Included Publications

A. Creating a Map of User Data in NTFS to Improve File Carving

The layout of the article has been lightly edited to fit the overall layout of the thesis. This text and a full citation of the article has been added below the title. Since the work is focused on disk partitions (Windows volumes) the term LBA in the article has been corrected to LPVA where applicable and one erroneous LBA erased from the second paragraph in Appendix A.3. The content is otherwise unchanged and corresponds to the original, published, version.

M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8

Abstract

Digital forensics, and especially file carving, is burdened by the large and increasing amount of data that needs to be processed. Attempts to solve the problem are being made by introducing for example more efficient carving algorithms, parallel processing in the cloud, and the reduction of data by filtering out uninteresting files.

We propose to use the principle of searching where it is more probable to find what you are looking for. This is done by creating a map of the probability of finding unique data at different Logical Partition Volume Address (LPVA) positions of a collection of storage media. We use Secure Hash Algorithm 1 (SHA-1) hashes of 512 B sectors to represent the data. Our results show that the mean probability of finding unique hash values at different LPVA positions vary between 12% to 41% over a New Technology File System (NTFS) partition.

The map 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 random sampling frequency, which we show. Our method 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.

A. Creating a Map of User Data in NTFS to Improve File Carving

Our work is based on a collection of 30 NTFS partitions from computers running Microsoft Windows 7 and newer.

A.1. Introduction

The ever-increasing amount of data to be handled in digital forensics is a major challenge to digital forensics case work [1] and has been discussed for many years [2–6]. The field of *file carving* is especially affected by the increasing amount of data. File carving is used in situations where there is no file system available, instead only the the properties of the stored data itself [7, 8] are used. 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 [9–14]. We have also used the compressibility of data for type identification [15–17]. As before we now use small blocks of data (512 B sectors in this article) and their statistical properties to improve file carving, but now we apply the principle of finding patterns in unknown data to full hard disk partitions. However, this time we determine the most probable position of user data, not the exact type of it.

Being able to carve files without the help of a working file system is difficult, but highly valuable to the digital forensic investigator. The research community is therefore focused on solving the problem of the increasing amount of data by different means. In a survey from 2014 Quick and Choo [1] lists the following concepts; data mining, data reduction and subsets, triage, intelligence analysis and digital intelligence, distributed and parallel processing, visualization, digital forensics as a service (DFaaS) and different artificial intelligence techniques.

In the file carving sub-field of *hash-based carving*, hashes of blocks of unknown data from the storage media is compared to known equally sized blocks of suspicious material. The large amount of comparisons of hashes made by a hash-based carving algorithm puts extra burden on the forensic process. Therefore different strategies, techniques and algorithms for hash-based carving have been developed [18–24].

What has not yet been tested is to utilize the principle of searching for something where it is more probable to find it. Since the allocation algorithm of the operating system (OS) will place new data in the file system according to a set of rules, not randomly, the principle can be used in the digital forensic field too. The allocation process is however too complex to fully understand and thus commonly regarded as random. Therefore the current principle is to linearly search the storage media from beginning to end, regardless of the most probable position of the sought-after data.

Most of the data of interest in an investigation is related to user activity, i. e. system logs and files created by the user. Such data is often unique to the specific computer and cannot be found elsewhere, because the probability of two users independently

creating exactly the same data is negligible. Of course also shared data downloaded (from the internet or elsewhere) by the user are of interest, for example child abuse material. Such data will be stored intertwined with the unique user data according to the allocation algorithm's rules. Hence it makes sense to use the Logical Partition Volume Address (LPVA) position of unique user data to also find where the user's activity has stored shared data.

We have therefore performed an experiment using Secure Hash Algorithm 1 (SHA-1) hashes of the content of non-related computers running Windows 7 and later, to find the probability of unique hashes (unique user data) at different positions in the 30 largest New Technology File System (NTFS) formatted partitions of 26 hard disks. The data was chosen to be as realistic as possible to increase the applicability of our results, and we therefore used real world computers in the data collection. The unique data are unique in our data set, there is no guarantee that they are unique world wide.

The rest of this paper is organized as follows: The remaining parts of Section A.1 presents related work and our contributions. In Section A.2 we describe our data set and how it was collected, together with a description of how the experiments were implemented. Section A.3 presents the results of the study. In Section A.4 we discuss the effects and implications of our results to the research field of hash-based carving and also to other areas within and related to digital forensics. Section A.5 concludes the work and presents ideas of future work to be done.

A.1.1. Related work

Although we have not found any work that directly relates to our work, there are several research sub-fields that have bearing on our work; The main field being file carving, and especially its sub-field of hash-based carving.

File fragment carving

Apart from our work within the file fragment carving area there are also work done by others using different means to identify the type of data fragments. Veenman [25] use the entropy, histogram and Kolmogorov complexity of 4 KiB file fragments to determine their type. The result show that histograms have the highest detection rate versus false positives of the chosen algorithms. Calhoun and Coles [26] experiment with different statistical measures, for example frequency of ASCII codes, entropy, modes, mean, standard deviation and correlation between adjacent bytes. They also look at using the longest common sub-strings and sub-sequences between file fragments for data classification. Ahmed et al. [27] use the byte frequency distribution with a new method of measuring the distance between the statistical properties of a data fragment and a model. Instead of using the Mahalanobis distance measure they use cosine sim-

ilarity with improved results. Li et al. [28] also use the byte frequency distribution (histogram) of different data fragments, but in conjunction with a support vector machine as a discriminant between different data types. They explain that the best results are achieved using the byte frequency distribution alone. Fitzgerald et al. [29] combine several statistical measures of data fragments (among them histograms of one and two byte sequences, entropy and Kolmogorov complexity) to get feature vectors that are fed into a support vector machine for classification. They notice that their method outperform many method presented in previous work. However, they do not evaluate the contribution of each of the chosen feature vectors, but instead leave it as future work. There is also a taxonomy of data fragment classification techniques by Poisel et al. [30] describing the research area.

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 [31]. According to Garfinkel one of the first times hashes are used for file carving is during the Digital Forensic Research Workshop (DFRWS) 2006 Carving Challenge [18]. Later the spamsum tool is used as a basis for an article by Kornblum [32] on piecewise hashing and what is now known as *approximate matching*. The concept of using hashes for file carving is further studied by Dandass [33] 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. [24] exploring the possibility of using a Graphics Processing Unit (GPU) for comparing hashes of 512 byte sections of known files with hashes of 512 byte sectors from disk images.

When Garfinkel use hashes for file carving in the DFRWS 2006 Carving Challenge [18] parts of files found on the internet are hashed and used to find equal hashes in the challenge image. These experiences lead to the development of the `frag_find` tool [23]. In connection to the `frag_find` article the authors discuss the optimal size of the data blocks to hash. They conclude that the size shall be equal to the sector size, without stating if they mean 512 B or 4 KiB sectors. Garfinkel et al. [18] 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 [34].

Foster [22] discusses the problem of data shared across files, stating that “the block of NULs is the most common block in our corpus” [22, 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. [21] continues the work further developing the 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 [18, 22, 23]. 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.

Data persistence

The concept of data persistence is interesting 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 map of a generic storage media.

Jones and Khan [35] 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 [36] present 12 factors affecting data persistence in storage media. Fairbanks [37, 38] also describes the low-level functions of ext4 and their effect on digital forensics.

Data reduction

Quick and Choo propose different methods to reduce the amount of data needed to be analyzed in digital forensic investigations. Their approach [4, 39] builds on extracting specific files using a list of key files and then work 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 [40] 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.” [40, p. 86]. However, the methods studied by Rowe all require a working file system, which is not consistent with the foundation of file carving.

Data mapping

Key [41] has developed an EnScript module to the EnCase software which creates a map of the recoverable sectors of a file found in a file system. It can handle situations where other tools does not work, for example partially damaged files, although it is very processor intensive and therefore can only create maps of a few files at a time.

Gladyshev and James [2] 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

A. Creating a Map of User Data in NTFS to Improve File Carving

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 mention using the distribution of data on disk, but do not seem to relate that to the probability of finding user data at different LPVA positions in storage media.

In two articles by van Baar et al. [42] and van Beek et al. [43] outlining the DFaaS system Hansken [43] and its predecessor Xiraf [42] the concept of non-linear extraction of data from images is discussed. Both van Baar and van Beek suggests that the Master File Table (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 in 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.

A.1.2. Contribution

As can be seen from the review of related work, there is a need to improve the efficiency of the tools and algorithms used in digital forensics, and especially in file carving. There are many different proposed solutions to the problem, but no one has yet utilized the inherent structures of the allocation algorithms to address the problem. We therefore present the novel idea of using the probability of finding user data at different locations in storage media to govern the digital forensic process and hence enabling an immediate increase of the efficiency of existing file carving algorithms and tools. In hash-based carving the concept can be used to increase the efficiency when doing random sampling by varying the sampling rate in accordance with the probability of finding user data at different LPVA positions. The principle can also be used during triage and other situations where speed and detection rate has to be balanced.

Unlike many of the methods presented in related work our method works without a file system. The map we create can be used directly to further improve the speed of any of the file carving algorithms presented as related work by showing the most probable position of unique data in a general NTFS formatted storage media. It can either be used for starting the forensic process at the position with the highest probability of containing data of interest (for example user data) or varying the sampling rate in accordance with the probability of finding user data. In the latter case the sampling frequency will be higher where it matters most and lower in other areas, increasing the probability of getting a hit while maintaining the same amount of samples as with equally distributed sampling.

Our work also benefits the digital forensic investigator, because our map introduces the possibility to plan the forensic process in a similar way to how a map is used when planning operations in the physical world. Currently storage media are treated as black boxes, forcing the forensic investigators to spend valuable time scanning them from

start to end before the analysis. This is especially useful in general file carving situations when there is no file system to govern the search. With our method the forensic investigators can focus on relevant areas of the storage media and postpone, or even skip, less relevant areas.

The map can also be used in storage media imaging situations. By starting the imaging process at the most probable position of user data, continuing in decreasing order of relevance, the analysis process can be run almost in parallel to the imaging since the most relevant data for analysis will be immediately available. In that way the analysis process can be started earlier, even before the imaging is finalized, saving valuable time and effort for the forensic investigators. Of course the reliability of the analysis will increase as more data are analyzed, but a preliminary result to guide the progressing work will be available at an earlier stage. This concept is also supported by the Hansken project [42, 43]. By implementing our concept in Hansken its ability to also handle media with broken file systems will be higher, possibly close to the performance of the standard process.

To enable handling of any storage media, regardless of its file system cluster size, our method use 512 byte sectors when hashing the data. Since our map is created once and can be reused there is no performance penalty in using it, just like a physical map. Since we have divided the map into a small number of equally sized areas (currently 128) any random seek penalty will only occur between these areas, not within, and thus can be ignored. Also the only situation where the use of 512 byte hashes are required is when the map is created. There is no need to use 512 byte hashes when performing case work on a suspect's hard drive. Likewise any hashing algorithm can be used for case work because the hashes of the map are only used to calculate the probability of user data at different positions and never meant to be compared to hashes from a specific case. If a hash algorithm is broken it can simply be exchanged for a new and better algorithm, our map will still work.

During our work we found a total of nine sectors having the same hash value at the same LPVA position in all 30 partitions. These sectors can for example be used to identify an NTFS file system, find the start of a NTFS partition and locate the \$MFT file for further processing. This can be done regardless of the state of the file system.

To the best of our knowledge this specific research field has not yet been explored, a field with the possibility to bring improvements to a number of related research fields in digital forensics. This new approach therefore has a high impact factor and relevance to most, if not all, digital forensic cases.

A.2. Experimental Setup

To determine the distribution of unique data in the major NTFS formatted partition of a common Microsoft Windows computer we first collect live data from real computers. Then we calculate the probability of finding unique hash values at different LPVA positions. Finally we create a map by calculating the mean probability of a number of (128 in our case) equally sized partition areas based on LPVA position. The mean probability calculation is done to generalize and scale the map into a usable format.

To lower the size of data to be stored for the experiment and also to protect the privacy of the user we use the SHA-1 algorithm to hash each 512 byte sector of all 30 NTFS formatted main partitions included in the experiment. We use SHA-1 because it currently offers the best balance between speed, collision risk and hash size among the hash algorithms we choose from (Message-Digest algorithm 5 (MD5) and the SHA family). The choice is based on a practical evaluation using available hardware. The hashing of data is done locally at each source computer and thus only the resulting hashes leave the computers.

SHA-1 maps 512 bytes of data onto a 20 byte long hash and thus there is a theoretical risk of collisions. If we apply the Birthday Paradox to our situation, the risk of a collision is approximately $1.1 \cdot 10^{-28}$ and hence negligible¹. We therefore assume a unique SHA-1 hash to represent a unique piece of data.

Even though the SHA-1 algorithm is broken [44, 45] from a cryptographic point of view the risk of an intentional collision is also negligible, because the amount of computing power required to create a collision is out of reach for the common user [44, 45]. Also such an attack would require an attacker to create a large amount of collisions for a majority of the storage media in the map source data. It would be much simpler to fill the disks with shared and unique data in an intentional pattern. This is however

¹The theoretical risk of collisions come from the fact that 512 bytes of data are compressed into a 20 byte long hash and therefore the results might contain false positives. The problem can be viewed as a Birthday Paradox, where N is the number of possible hashes, n is the number of hashes, i. e. the total amount of sectors we have hashed (as a worst case scenario), and $P(Collision)$ the probability of a collision, which can be calculated as

$$P(Collision) = 1 - \frac{N!}{N^n \cdot (N - n)!}$$

and with $N = 2^{160}$ and $n = 18\,210\,308\,798$ the probability of at least one collision is approximately

$$P(Collision) \approx 1 - e^{-n^2/2N} \approx n^2/2N \approx 1.1 \cdot 10^{-28}.$$

Our approximation is based on Stirling's approximation of factorials, which gives acceptable results when dealing with very large numbers. Since the SHAttered [44, 45] attack is 100 000 times faster than a brute force attack using the birthday paradox the risk of an intentional collision is higher, but the attack is unfeasible in our situation.

mitigated by collecting the source data from non-related sources. Finally the mapping process is not limited to the use of SHA-1, any hashing algorithm will do, as long as all mapping data is hashed using the same algorithm.

A.2.1. Data Collection

To get hold of data representing real life situations we chose to use a convenience sample collecting data from computers owned by people in our acquaintanceship. We did not use the Real Data Corpus (RDC) because the time stamps on the RDC web site [46] indicate that the last update of the data set was made in 2011. Therefore our data set is more up to date containing also versions 8 and 10 of Windows².

We have collected data from 30 partitions of 26 computers (23 consumer grade and 3 office grade). The data was collected by hashing every 512 byte sector of the entire hard disks using the `dcflddd` disk imaging tool set to use the SHA-1 cryptographic hash algorithm. The OS installations represent three different language packs and range from Microsoft Windows 7 to Windows 10, both Enterprise, Professional, Ultimate, Home and Educational versions. Some of the computers have been upgraded from an earlier Windows version to Windows 10. Five of the computers are in our possession and we therefore have access to their raw content.

The reason for using real computers and not a simulation in a laboratory environment is to avoid any bias from the simulation of user behavior. By using real computers our results will be as close to the forensic investigators case work as possible. The drawback is a lower degree of control of the material. For example we lack information on whether a hard disk is mechanical or solid-state drive (SSD) in some cases. This lack of information does not affect our results since we collect the data at the LPVA level from the hard drive controller. The lower levels of physical storage formats are therefore hidden from us [48–51].

From our point of view the only difference between a mechanical hard disk and an SSD hard disk is their filling of unused areas, which can be either old data, 0x00 or 0xFF depending on how the TRIM command is implemented in the SSDs [52–57]. Hence a mechanical drive will more often give us old data from currently unallocated clusters than an SSD. Since we only use the LPVA positions of unique data any 0x00 and 0xFF filling is automatically filtered out. In the case of old data from unallocated clusters a very unbalanced erase/write cycle is required to leave a large amount of old data, i. e. first a large amount of data should be erased, followed by a small amount of (or no) writing of new data. This will be the case if a hard disk is erased using a random pattern and then reformatted and reused. If a large amount of the unallocated sectors contain old data, which are unique, they will have an effect on our results. To affect the

²Windows 8 was introduced at the end of 2012 [47] and therefore cannot exist in the RDC, nor can Windows 10.

A. *Creating a Map of User Data in NTFS to Improve File Carving*

map creation process to any greater extent the scenario shall be true for a significant part of the partitions in our data set. Before we collect the data we therefore check with the users if they have done any large file system cleaning close to our data collection.

The hard disks in our data set differ in size, ranging from 64 GB to 1 TB. We extract the largest NTFS formatted partition (in four cases there were an extra storage partition present which was extracted too) from each hard disk, based on the assumption that it contains the OS and user files. As can be seen in Table A.1 the total size of the partitions in the experiment is 8 638.4 GiB, corresponding to 18 210 308 798 hashes. Of those 3 809 786 792 hashes are unique. The percentage of unique hashes for each partition is also shown in Table A.1. A low number of unique hashes is an indication of the partition not being used, or at least not for storing user data. A low amount of unique hashes and a high amount of 0x00 or 0xFF filling can be seen in Table A.1 for the bigger partitions (those ending in “b”) of the hard drives where we used more than one partition to collect data.

The life time and hence amount of data stored on the hard disks vary. Most of the hard disks are filled with 0x00 to some extent. That can be remnants of the production process, but of the smaller hard disks (≤ 256 GiB) some are SSD, which are filled with 0xFF from the factory [52]. To determine whether any of the partitions in our data set has been completely filled with data at any time during its life time we studied the last 20 GB of each partition. The size of 20 GB was chosen to be a suitable trade-off between a large enough amount of data and the risk of including the OS area for the smaller partitions. In Table A.1 the partitions sizes and the amount of 0x00 and 0xFF filling are shown. A low amount of both 0x00 and 0xFF filling is an indication of the partition being (almost) completely filled with data during some stage of its life time. This could either be user data or a random data from a disk wiping tool.

We are only using partitions formatted as NTFS, because that is currently the most common file system among desktop systems having an approximate market share of 90% [58]. The partition names in Table A.1 are given based on the order of hashing, i. e. partition “A” was hashed before “B” and so on. Four computers contain two partitions each that are included based on size (the computers were installed with an extra partition for user data). These partitions are indicated by a second lowercase letter in the name in Table A.1. Although lacking an OS these partitions still contain an NTFS file system and therefore can be included in our data set.

The unique hash values we have found also include an amount of 1 KiB³ MFT records. These records will result in up to two unique hash values each when hashing due to their highly varying content (time stamps, file names, file content etc). We therefore performed a survey on 27 computers not included in our data set estimating

³The size of an MFT record is defined in the boot sector of an NTFS partition. The de facto standard size is 1 KiB [59].

Table A.1.: The sizes in GiB of the partitions in the experiment, their amount of unique hashes and 0x00 and 0xFF filling in the last 20 GB of the partitions. A low amount of filling is an indicator of the partition being completely filled or wiped with at random pattern at some stage during its life time.

Name	Size [GiB]	Unique hashes [%]	0x00 fill [%]	0xFF fill [%]
F	59.5	0.08	100.00	0
E	59.5	22.36	2.01	0.12
AC	111.3	7.63	100.00	0
I	111.6	61.83	20.12	1.67
A	118.4	23.17	75.24	0.00
W	118.6	59.80	26.18	0
K	146.4	5.82	100.00	0
Qa	150	43.33	45.78	0.07
N	177.6	38.32	48.48	0.00
Ra	185.9	31.89	56.89	0.14
Sa	200	86.85	0.02	0.02
Oa	209	13.68	100.00	0
Y	217.1	14.77	100.00	0
P	232.7	53.97	0.83	0.07
H	237.3	16.68	100.00	0
AA	237.3	12.60	100.00	0
D	237.9	20.59	0	100.00
G	238.1	7.03	25.56	0.16
M	238.1	23.06	79.47	0.01
Rb	258.4	1.68	100.00	0
Sb	265.6	36.22	100.00	0
T	297.9	9.35	48.12	0.28
C	421.7	34.98	0.86	1.17
Z	423.9	4.05	100.00	0
X	443.8	6.48	100.00	0
U	448	10.60	100.00	0
V	465.6	48.43	100.00	0
Ob	699	0.15	98.72	0.00
Qb	766.5	1.47	100.00	0
B	905.2	29.74	100.00	0
Sum	8683.4	20.92	67.61	3.35

A. Creating a Map of User Data in NTFS to Improve File Carving

the mean number of MFT records by counting the total number of files and folders in the computers (since each file and folder in a computer is represented by, at least, one MFT record⁴). The result of the survey showed that the average total amount of files and folders in these computers were 363 630. Due to the uncertainty involved in the counting (we counted via the file explorer) the value includes an extra 25% added to cover for hidden files, files requiring more than one MFT record and any MFT records that are internal to the file system. The extra 25% also cover for any network storage of user data of the office grade computers in our file counting data set. In consumer grade computers all user files would probably have been stored locally and therefore included in our counting.

A.2.2. Implementation

To prepare the data for the experiment we extract and merge the hash data from the largest partitions into a single file, which is then sorted in ascending order of hash value. We then extract the unique hashes from the file, thus any 0x00 and 0xFF filled sectors are automatically filtered out. After the extraction of unique hashes we sort them in order of ascending LPVA position and separated them in individual files based on partition identity. The data for each partition are then divided into 128 equally sized areas, each being $\frac{1}{128}$ of the size of the partition. Then we calculate the probability of finding unique hashes in each area through counting the number of unique hashes divided by the size of the areas in sectors for each partition.

After the probability calculation step we calculate the mean, median and standard deviation of the probability of unique hashes for each area of the partitions regardless of the differing partition sizes. The mean values are used as a map of a general storage media, showing where it is more probable to find user data (unique data) in a generic NTFS formatted partition.

A.2.3. Evaluation

To evaluate our map we run an experiment simulating a hash-based carving scenario comparing the performance of sampling according to our map to a uniform sampling distribution. As ground truth we use four real partitions not included in our data set. We use the distribution of unique data in the four partitions to pick a random integer *target*. Then we use our map to pick a random integer *map* and the uniform distribution to pick a random integer *uni*. All random integers are selected within the same total range representing the LPVA positions of a fictive partition, although with bias for *target* and *map*. If *map* = *target* our map gets one hit, if *uni* = *target* the uniform

⁴If a file has many attributes, for example alternate streams or is heavily fragmented, the file system creates a new MFT record to hold the extra information.

distribution gets one hit. The predefined range is set to 16 MiB and divided into 128 equally sized areas using the mapping process. The small partition size was chosen to increase the number of hits.

We iterate the random sampling process 10^9 times for each of the four partitions to stabilize the result. The low number of partitions used to create the map does however affect the evaluation since it is a small population to build a model from. Likewise our set of partitions forming the ground truth is small and the result is therefore affected by any individual variations of the partition content. Another factor affecting the result is the fact that the four partitions used as ground truth were taken from computers that should be scrapped and therefore had well used hard drives. They therefore contained a lower amount of 0x00 and 0xFF at the end.

The experiment was executed using Python 2.7 and the random library in a Debian Stretch (v. 9) computer.

A.3. Result

As can be seen in Figure A.1 showing a map of our results the probabilities of unique hashes at different positions are varying between approximately 12% to 41%. The low median values in the second half of the partitions are due to the presence of 0x00 and 0xFF filling in a significant number of the partitions. If more than 50% of the partitions have no or a very low amount of unique data in that area the median value will be (close to) zero, which it is. The plot is based on splitting each partition into 128 blocks corresponding to $\frac{1}{128}$ of the partitions size.

When formatting a hard disk with NTFS 12.5% of the volume space is reserved for the MFT [60] as default. In all 30 partitions in our data set the MFT area starts exactly 3 GiB into the partition. Hence the start of the area where non-resident file data are allocated can be found at position $P = 3 * 2^{30} + 0.125 \cdot \text{partition size in bytes}$, if not the user changes the MFT reserved space at the time of formatting. If the partition is very small the non-resident data allocation point is probably changed. Based on our data set the non-resident data allocation start is valid for partitions ≥ 60 GiB.

At the non-resident data allocation point the bulk of the OS, first user files and different software from the initial installation reside. The minimum space requirement for a Windows 7, 8, 8.1 and 10 installation is 20 GiB for 64-bit systems according to Microsoft [61–63]. The most probable start of storage of the day-to-day usage of a partition containing Windows is consequently at $(3+20) \cdot 2^{30} + 0.125 \cdot \text{partition size in bytes}$ bytes into a partition. Transferring this to a percentage of the partition length gives in our case (see Figure A.1) a value approximately between 14 and 43%. The highest amount of OS files is found in the beginning of the area and it decreases towards the end. This can explain the overall sharp negative trend of the plot between 20% and

A. Creating a Map of User Data in NTFS to Improve File Carving

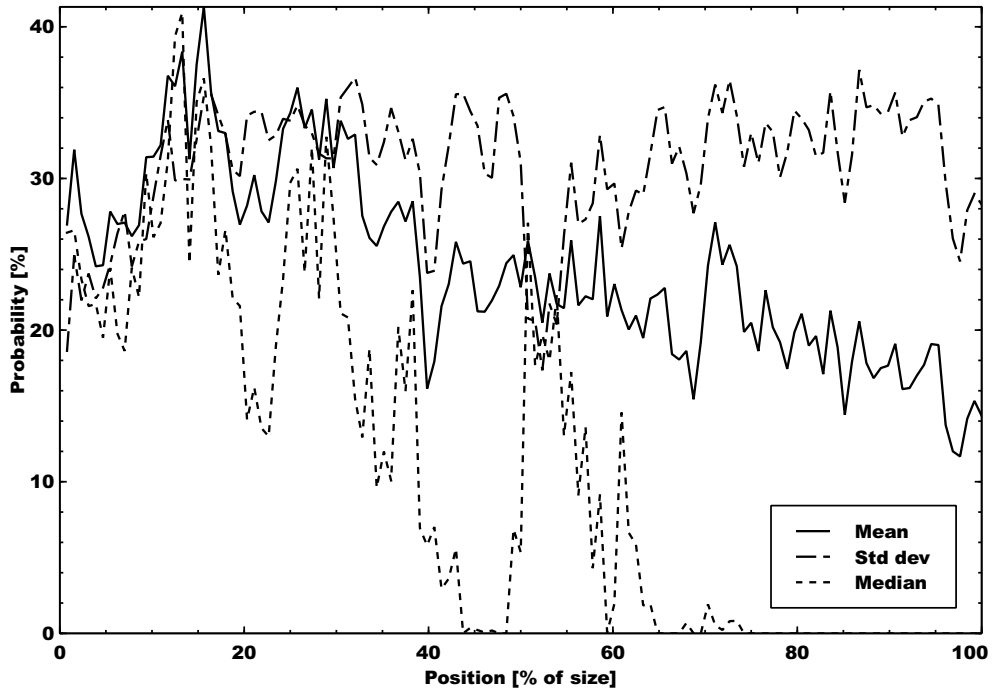


Figure A.1.: A plot of the mean, median and standard deviation of the probability of unique hashes (in percent) at different positions of the 30 partitions in our data set. The position is given as percent of the partition size. Each of the partitions is split into 128 equally sized areas based on the specific partition's total size. The behavior of the median plot in the second half is due to the low number of unique hashes in these parts of most of the partitions.

40%, together with the peaks around 30%. Looking at the behaviour of the mean plot from 40% and upwards the values are slowly decreasing and the standard deviation is increasing. This is the effect of the differing usage patterns of the partitions. Some have been storing more data, been more utilized, than other partitions in the data set.

We found 3 809 786 792 unique hashes in our data set, which correspond to data created locally by the user or the OS, such as logs. There are however unique parts of MFT records too in the data. Each file and directory is represented by at least one MFT record in NTFS⁵. The MFT records might affect the result by increasing the number of unique non-user data hashes. To estimate the effect from the MFT records we studied the number of files and folders in 27 typical computers (both office and home). We found the mean value to be 363 630 files, which corresponds to approximately 0.7% of the unique hashes in our 30 computers.

The `pagefile.sys` and `hiberfil.sys` might also generate a large amount of unique hashes depending on to what extent they are used. These files will certainly affect the map and our results, but since they are of high value to a digital forensic investigation they should be included in our data.

During the work with the mapping process we found four sectors containing the same hash value at the same LPVA position in all partitions included in our data set. The sectors are found in file system cluster 786 435. They all contain the second half of MFT records which has only been used once according to their signature values [59, p. 352]. The first part of these MFT records contain similar, but not equal information. The `istat` tool [64] shows that the sectors belong to the `$MFT` file, i. e. the file system itself. The `$DATA` attribute of the `$MFT` files in the five computers we have raw access to all allocate the same eight clusters at the beginning of the run length (see Table A.2). Combining this with the static content of the four sectors in cluster 786 435 the NTFS formatting seems to place the start of the MFT at the same position exactly 3 GiB into the partition. If this is true the first and last sectors of an NTFS partition should contain the hexadecimal string `00 00 0C 00 00 00 00 00` starting at position `0x30` [65] (little endian). We have verified this for the five computers we have raw access to.

According to Carrier the “`$DATA` attribute of the `$MFTMirr` file allocates clusters in the middle of the file system” [59, p. 303]. This implies that the middle sector, based on the size of the volume (the partition), is actually where the mirror should be kept. However, this is not always true. In four of the five computers we have full access to the `$MFTMirr` file allocates file system cluster 2 and in the last partition file system cluster 8 912 895 is allocated. However, the latter partition is 59 919 808 clusters in size, hence none of the `$MFTMirr` files are located near the middle of any of the

⁵Depending on the number of attributes connected to a file more than one MFT record might be needed to store them. A typical example is a file with a lot of alternate data streams, or a highly fragmented file.

A. Creating a Map of User Data in NTFS to Improve File Carving

```
Type: $DATA (128-12)   Name: N/A   Non-Resident [...]
786432 786433 786434 786435 786436 786437 786438 786439
[...]
Type: $DATA (128-1)   Name: N/A   Non-Resident [...]
786432 786433 786434 786435 786436 786437 786438 786439
[...]
Type: $DATA (128-6)   Name: N/A   Non-Resident [...]
786432 786433 786434 786435 786436 786437 786438 786439
[...]
Type: $DATA (128-1)   Name: N/A   Non-Resident [...]
786432 786433 786434 786435 786436 786437 786438 786439
[...]
Type: $DATA (128-6)   Name: N/A   Non-Resident [...]
786432 786433 786434 786435 786436 786437 786438 786439
[...]
```

Figure A.2.: Part of the \$DATA attribute of the \$MFT file for five computers in our data set. The eight numbers on every third row indicate the file system clusters allocated to the file. File system cluster 786 435 contains the four static sectors (at positions 6 291 481, 6 291 483, 6 291 485 and 6 291 487) we have found in all 30 partitions.

partitions. Consequently the allocation strategy of NTFS seem to have changed since Carrier wrote his book.

To evaluate the efficiency of our map in random sampling situations we tested it against 4 NTFS partitions not included in the 30 used to create the map. Due to the low number of partitions used in the evaluation, the distribution of unique data in the individual partitions have a high impact on the result. We therefore regard the result as a first indicator of the performance of future maps, not the final answer. We are awaiting access to more data to be able to run a new evaluation. The result of the evaluation can be seen in Table A.2.

The best result (when the map most resembles one of the evaluation partitions) is almost 10% better than using a uniformly distributed sampling rate. Varying the number of equally sized areas do not change the results in any significant way, neither do varying the fictive (16 MiB) partition size.

Table A.2.: The result of the evaluation of the map against four unrelated NTFS partitions, which are not included in the 30 partitions in the mapping data set. The table shows the number of hits using the map relative to using a uniformly distributed sampling rate. We used 16 MiB partitions divided into 128 equally sized areas sampled 10^9 times for the evaluation.

Map	Uniform	Map/uniform [%]
28635	30279	94.6
29881	30363	98.4
32556	30836	105.6
33257	30461	109.2
124329	121939	102.0

A.4. Discussion

Although the validity of the idea of looking for data where the probability of finding it is higher than randomly searching for data in a uniform pattern is based on common sense we have also performed an empirical evaluation to test our specific implementation. The result shows an improvement of 2% when using our map compared to a uniformly distributed sampling rate. This might not create a paradigm shift, but it still is a positive indicator of the relevance of our idea. The reason for the seemingly poor result is the low number of partitions used to create the map. To reveal the underlying deterministic allocation pattern the amount of data needs to be much larger. Using a more solid statistical foundation will then improve the strength of the result. Having a big enough data set also allows us to divide it into several use cases, each one rendering its own map. The idea is to be able to diversify between for example web surfers, office administrators, file sharers. This however requires a much larger data collection effort, while maintaining a high level of control of the collected material to filter out unique data not created by the user or system, such as data written during disk wiping.

When the mapping foundation is stable there are several ways 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 using hash based carving to find parts of files in a hard disk. Then three different scenarios are possible:

Speed is prioritized. The total amount of samples is lowered compared to the uniformly distributed sampling case without any significant loss in detection ability. This scenario can for example be used in triage situations or when there is a need to get a preliminary answer quickly.

A. Creating a Map of User Data in NTFS to Improve File Carving

Speed is maintained. The same amount of samples are maintained compared to the uniformly distributed sampling case, which gives a higher detection ability at the same execution speed. This is the standard case, which can be used without changing the digital forensic process.

Detection rate is prioritized. A larger amount of samples are used compared to the uniformly distributed sampling case, giving a much higher detection rate at a lower cost in execution speed. For example used in situations where the suspects hard disk has an unusual usage pattern. In this way the standard amount of hashes can be maintained in low priority areas and at the same time use a higher sampling rate for better detection ability in high priority areas of the hard drive.

When the area reserved for the MFT is used up a new area equalling 12.5% of the volume size is added. If possible that area is to be contiguous, but need not be. Hence as the file system grows new MFT records are added and allocated where suitable [59]. Thus an old and well used NTFS partition might very well have MFT records spread all over the storage space. This would possibly affect the creation of the map, adding noise to the unique data. According to our empirical study of the number of files and directories (usually represented by a single MFT record each) in an NTFS partition the amount of MFT records corresponds to approximately 0.7% of the total amount of unique hashes in each partition. The actual amount of unique hashes belonging to an MFT record is probably less than 0.7% because the second part of an MFT record often contain 510 zero bytes followed by a two byte long *signature value*⁶ at the end of the sector. The worst case scenario is a partition filled with files less than approximately 700 bytes⁷ in size, which would result in a partition filled with MFT records storing the data internally. If all files contained the same data only the MFT meta data (time stamps etcetera) would differ, thus the partition would still seem to be filled with random data. The maximum number of files in an NTFS partition is $2^{32} - 1$ [60], hence the partition would be approximately 4 TiB in size.

To estimate the amount of unique MFT record hashes in our data set in another way we generate SHA-1 hash values for all possible combinations of 510 zeros and a two byte signature value, which correspond to the second half of a standard MFT

⁶Signature values [59] are used by NTFS to verify the integrity of data structures (but not sectors containing file content) spanning two or more sectors. The last two bytes of every sector in such a data structure are called a *fixup value* and are moved to an array in the beginning of the structure during the process of writing to disk. These last two bytes are then replaced by the signature value. When the data structure is read the signature values are used to check that all sectors that are read have the same signature value, and thus belong to the same data structure. Every time a data structure is updated on disk the signature value is incremented by one [59].

⁷The 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 to 700 bytes. Microsoft reports a 900 byte limit [60].

record. The first such hash being unique in our data set represent a signature value of 3613 (0x1D0E, little endian). Many of the lower signature values generate several thousands of hits. There is however no guarantee that all the generated hashes belong to MFT records, but at least four do and consequently the amount of possibly unique MFT hash values polluting our data set is most probably less than 0.7%. Hence the unique hashes of the MFT records do not affect the precision of the map to any larger extent.

We have chosen to limit our experiment to computers running Microsoft Windows 7 and later and having NTFS formatted main partitions. To protect the privacy of the computer owners we use the cryptographic SHA-1 hash to obscure the real data. This limits our ability to trace the original data of each hash, but since we are only interested in the LPVA position of unique hashes we do not need to know what data the hashes represent to be able to create a map.

Our work can also be used to find shared data. Of special interest is what we call static data, shared data that are found at the same LPVA position in several unrelated storage media. Knowledge of the LPVA position of static data will be of great use for a wide range of digital forensics applications. Together with for example forensic imaging and analysis prioritizing such knowledge can also provide an investigator with the means to break the encryption of a hard drive through a plain text attack [66], depending on the encryption algorithm used.

The LPVA position of static data can be used to handle corrupt storage media. In many cases large parts of the corrupt media are readable, but there are no indications of the forensic value of the lost parts. Having access to a map of static content in storage media will help the digital forensic investigator to improve the evaluative reporting during case work by indicating the forensic value of any lost areas. This will in the end lead to a higher confidence in the collected evidence.

Furthermore, a map can be used to create signatures to identify the correct file system in partially recovered partitions. Since the meta data layout and allocation process during installation are differing between OSs such signatures are feasible.

Finally areas that should have a high probability of static content, but do not, will work as an indicator of the presence of malware or any other suspicious activity in a file system, since deviations are unlikely in such areas. Instead of having to hash every file in a file system in search of deviations, the search can start at the most probable place in the file system. The partition is then scanned in descending order of probability of static content.

A.5. Conclusion and Future Work

Our work is based on the principle that it is better to search for something where the probability of finding it is higher. We therefore have developed a method to create a map of the probability of finding unique data at different LPVA positions of storage media. The term unique data is defined as data that are created locally on a computer and not (yet) shared. This includes both data created by the system, such as log files and data created locally by the user (not downloaded from the internet). Such uniquely created data are often more valuable to a forensic investigation than shared data, even though shared data of course can be valuable too.

The map provides the digital forensic investigator with a pre-calculated view of a generic storage media, which can be used to concentrate the forensic process on the relevant parts of the disputed material, instead of spending valuable time on first scanning the complete storage media from end to end. The concept of unique data is only used when creating the map, which is done once (apart from regular updates). When the map is finished it can be used repetitively for any data, method, tool or investigation process and without the need to recreate it for each new case.

The concept of creating a map of the probability of unique (or static) data at different positions of storage media opens up a new world of applications. It can for example be used in triage situations, when planning the order of analysis of large amounts of seized storage media, estimate the value of partially analyzed data due to corruption and for breaking encryption of storage media. We therefore plan to extend our data set to stabilize the map creation and make the map more reliable. We will also explore other methods to be used for creating maps, as well as the possibility to create maps for different use cases.

The four sectors with equal hash values that we found at approximately 3 GiB into all 30 partitions in our data set show that there might be specific areas of NTFS partitions that are static. We aim to search for and study the origin of any such areas as future work. We also plan to extend our approach to other file systems, especially ext4 and Apple File System (APFS), with the goal of creating a general mapping process for any storage media, regardless of type and file system.

We are releasing our current hash data set to the public, but due to its size the optimal transfer option will need to be agreed upon in each specific case. Please contact the first author to arrange for a transfer.

A.6. Bibliography

- [1] D. Quick and K. Choo. “Impacts of increasing volume of digital forensic data: A survey and future research challenges.” In: *Digital Investigation* 11.4 (2014), pp. 273–294. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.09.002.
- [2] P. Gladyshev and J. James. “Decision-theoretic file carving.” In: *Digital Investigation* 22.Supplement C (2017), pp. 46–61. ISSN: 1742-2876. DOI: 10.1016/j.diin.2017.08.001.
- [3] European Police Office (Europol). *Internet Organised Crime Threat Assessment (IOCTA) 2016*. Tech. rep. European Cybercrime Centre (EC3), 2016.
- [4] D. Quick and K. Choo. “Data reduction and data mining framework for digital forensic evidence: Storage, intelligence, review and archive.” In: *Trends & Issues in Crime and Criminal Justice* 480 (Sept. 2014), pp. 1–11. ISSN: 1836-2206.
- [5] F. Breitingner, G. Stivaktakis, and H. Baier. “FRASH: A framework to test algorithms of similarity hashing.” In: *Digital Investigation* 10.Supplement (2013). The Proceedings of the Thirteenth Annual DFRWS Conference, S50–S58. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.06.006.
- [6] V. Roussev. “Managing Terabyte-Scale Investigations with Similarity Digests.” In: *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*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 19–34. ISBN: 978-3-642-33962-2. DOI: 10.1007/978-3-642-33962-2_2.
- [7] R. Poisel and S. Tjoa. “A Comprehensive Literature Review of File Carving.” In: *2013 International Conference on Availability, Reliability and Security*. Sept. 2013, pp. 475–484. DOI: 10.1109/ARES.2013.62.
- [8] A. Pal and N. Memon. “The evolution of file carving.” In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [9] M. Karresand. “Completing the Picture — Fragments and Back Again.” Licentiate thesis. Linköping Institute of Technology, Linköping University, Sweden, May 2008.
- [10] M. Karresand and N. Shahmehri. “Reassembly of fragmented JPEG images containing restart markers.” In: *Proceedings - 4th Annual European Conference on Computer Network Defense, EC2ND 2008*. 2008, pp. 25–32. DOI: 10.1109/EC2ND.2008.10.

A. Creating a Map of User Data in NTFS to Improve File Carving

- [11] M. Karresand and N. Shahmehri. “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*. Piscataway, NJ, USA: IEEE, 2006, pp. 140–147. DOI: 10.1109/IAW.2006.1652088.
- [12] M. Karresand and N. Shahmehri. “Oscar — Using Byte Pairs to Find File Type and Camera Make of Data Fragments.” In: *Proceedings of the 2nd European Conference on Computer Network Defence, in conjunction with the First Workshop on Digital Forensics and Incident Analysis (EC2ND 2006)*. Ed. by A. Blyth and I. Sutherland. Springer Verlag, 2007, pp. 85–94. DOI: 10.1007/978-1-84628-750-3_9.
- [13] M. Karresand and N. Shahmehri. “Oscar — File Type and Camera Identification Using the Structure of Binary Data Fragments.” In: *Proceedings of the 1st Conference on Advances in Computer Security and Forensics, ACSF*. Ed. by J. Haggerty and M. Merabti. Liverpool, UK: The School of Computing and Mathematical Sciences, John Moores University, July 2006, pp. 11–20.
- [14] M. Karresand and N. Shahmehri. “Oscar — File Type Identification of Binary Data in Disk Clusters and RAM Pages.” In: *Security and Privacy in Dynamic Environments, Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22-24 May 2006, Karlstad, Sweden*. Vol. 201. Lecture Notes in Computer Science. Springer, 2006, pp. 413–424. DOI: 10.1007/0-387-33406-8_35.
- [15] S. Axelsson. “Using Normalized Compression Distance for Classifying File Fragments.” In: *2010 International Conference on Availability, Reliability and Security*. Feb. 2010, pp. 641–646. DOI: 10.1109/ARES.2010.100.
- [16] S. Axelsson. “The Normalised Compression Distance as a file fragment classifier.” In: *Digital Investigation 7*. Supplement (2010). The Proceedings of the Tenth Annual DFRWS Conference, S24–S31. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.004.
- [17] S. Axelsson, K. Bajwa, and M. Srikanth. “File Fragment Analysis Using Normalized Compression Distance.” In: *Advances in Digital Forensics IX: 9th IFIP WG 11.9 International Conference on Digital Forensics, Orlando, FL, USA, January 28-30, 2013, Revised Selected Papers*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 171–182. ISBN: 978-3-642-41148-9. DOI: 10.1007/978-3-642-41148-9_12.

- [18] S. Garfinkel and M. McCarrin. “Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb.” In: *Digital Investigation* 14.Supplement 1 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S95–S105. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.001.
- [19] F. Breitingner, C. Rathgeb, and H. Baier. “An Efficient Similarity Digests Database Lookup - A Logarithmic Divide & Conquer Approach.” In: *Journal of Digital Forensics, Security and Law* 9.2 (2014), pp. 155–166. DOI: 10.15394/jdfs1.2014.1178.
- [20] F. Breitingner and K. Petrov. “Reducing the Time Required for Hashing Operations.” In: *Advances in Digital Forensics IX - 9th IFIP WG 11.9 International Conference on Digital Forensics, Orlando, FL, USA, January 28-30, 2013, Revised Selected Papers*. Ed. by G. Peterson and S. Sheno. Vol. 410. IFIP Advances in Information and Communication Technology. Springer, 2013, pp. 101–117. DOI: 10.1007/978-3-642-41148-9_7.
- [21] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks. “Distinct Sector Hashes for Target File Detection.” In: *Computer* 45.12 (Dec. 2012), pp. 28–35. ISSN: 0018-9162. DOI: 10.1109/MC.2012.327.
- [22] K. Foster. “Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus.” MA thesis. Monterey, California, USA: Naval Postgraduate School, Sept. 2012.
- [23] S. Garfinkel, A. Nelson, D. White, and V. Roussev. “Using purpose-built functions and block hashes to enable small block and sub-file forensics.” In: *Digital Investigation* 7.Supplement (2010). The Proceedings of the Tenth Annual DFRWS Conference, S13–S23. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.003.
- [24] S. Collange, Y. S. Dandass, M. Daumas, and D. Defour. “Using Graphics Processors for Parallelizing Hash-Based Data Carving.” In: *2009 42nd Hawaii International Conference on System Sciences*. Jan. 2009, pp. 1–10. DOI: 10.1109/HICSS.2009.494.
- [25] C. Veenman. “Statistical Disk Cluster Classification for File Carving.” In: *Proceedings of the Third International Symposium on Information Assurance and Security, 2007 (IAS 2007)*. Ed. by N. Zhang, A. Abraham, Q. Shi, and J. Thomas. IEEE Computer Society, 2007, pp. 393–398. DOI: 10.1109/ISIAS.2007.4299805.

A. Creating a Map of User Data in NTFS to Improve File Carving

- [26] W. Calhoun and D. Coles. "Predicting the types of file fragments." In: *Digital Investigation* 5.Supplement 1 (Sept. 2008), S14–S20. DOI: 10.1016/j.diin.2008.05.005.
- [27] I. Ahmed, K.-s. Lhee, H. Shin, and M. Hong. "On Improving the Accuracy and Performance of Content-Based File Type Identification." In: *Proc. ACISP 2009*. Ed. by C. Boyd and G. Nieto. Vol. 5594/2009. LNCS. Springer-Verlag Berlin Heidelberg, 2009, pp. 44–59. DOI: 10.1007/978-3-642-02620-1_4.
- [28] Q. Li, A. Ong, P. Suganthan, and V. Thing. "A Novel Support Vector Machine Approach to High Entropy Data Fragment Classification." In: *South African Information Security Multi-Conference, SAISMC 2010, Port Elizabeth, South Africa, May 17-18, 2010. Proceedings*. Ed. by N. Clarke, S. Furnell, and R. Solms. University of Plymouth, 2010, pp. 236–247.
- [29] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn. "Using NLP techniques for file fragment classification." In: *Digital Investigation* 9 (2012). The Proceedings of the Twelfth Annual DFRWS Conference, S44–S49. ISSN: 1742-2876. DOI: 10.1016/j.diin.2012.05.008.
- [30] R. Poisel, M. Rybnicek, and S. Tjoa. "Taxonomy of Data Fragment Classification Techniques." In: *Digital Forensics and Cyber Crime: Fifth International Conference, ICDF2C 2013, Moscow, Russia, September 26-27, 2013, Revised Selected Papers*. Ed. by P. Gladyshev, A. Marrington, and I. Baggili. Springer International Publishing, 2014, pp. 67–85. DOI: 10.1007/978-3-319-14289-0_6.
- [31] A. Tridgell. *Spamsum README*. Last accessed 27-04-2018. 2002. URL: <https://www.samba.org/ftp/unpacked/junkcode/spamsum/README>.
- [32] J. Kornblum. "Identifying almost identical files using context triggered piecewise hashing." In: *Digital Investigation* 3.Supplement (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), pp. 91–97. ISSN: 1742-2876. DOI: 10.1016/j.diin.2006.06.015.
- [33] Y. Dandass, N. Necaie, and S. Thomas. "An Empirical Analysis of Disk Sector Hashes for Data Carving." In: *J. Digit. Forensic Pract.* 2.2 (Apr. 2008), pp. 95–104. ISSN: 1556-7281. DOI: 10.1080/15567280802050436.
- [34] Microsoft. *Default cluster size for NTFS, FAT, and exFAT*. Aug. 2015. URL: <https://support.microsoft.com/en-us/help/140365/default-cluster-size-for-ntfs--fat--and-exfat>.

- [35] J. Jones, T. Khan, K. Laskey, A. Nelson, M. Laamanen, and D. White. “Inferring Previously Uninstalled Applications from Residual Partial Artifacts.” In: *Annual ADFSL Conference on Digital Forensics, Security and Law*. 2016, pp. 113–130.
- [36] K. Fairbanks and S. Garfinkel. “Column: Factors Affecting Data Decay.” In: *Journal of Digital Forensics, Security and Law* 7 (2012). DOI: 10.15394/jdfs1.2012.1116.
- [37] K. Fairbanks. “A Technique for Measuring Data Persistence Using the Ext4 File System Journal.” In: *2015 IEEE 39th Annual Computer Software and Applications Conference*. Vol. 3. July 2015, pp. 18–23. DOI: 10.1109/COMPSAC.2015.164.
- [38] K. Fairbanks. “An analysis of Ext4 for digital forensics.” In: *Digital Investigation* 9.Supplement (2012). The Proceedings of the Twelfth Annual DFRWS Conference, S118–S130. ISSN: 1742-2876. DOI: 10.1016/j.diin.2012.05.010.
- [39] D. Quick and K.-K. R. Choo. “Big forensic data reduction: digital forensic images and electronic evidence.” In: *Cluster Computing* 19.2 (June 2016), pp. 723–740. ISSN: 1573-7543. DOI: 10.1007/s10586-016-0553-1.
- [40] N. C. Rowe. “Identifying Forensically Uninteresting Files Using a Large Corpus.” In: *Digital Forensics and Cyber Crime: Fifth International Conference, ICDF2C 2013, Moscow, Russia, September 26-27, 2013, Revised Selected Papers*. Ed. by P. Gladyshev, A. Marrington, and I. Baggili. Cham: Springer International Publishing, 2014, pp. 86–101. ISBN: 978-3-319-14289-0. DOI: 10.1007/978-3-319-14289-0_7.
- [41] S. Key. *File Block Hash Map Analysis*. Last accessed 28-04-2018. 2012. URL: <https://www.guidancesoftware.com/app/File-Block-Hash-Map-Analysis>.
- [42] R. van Baar, H. van Beek, and E. van Eijk. “Digital Forensics as a Service: A game changer.” In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S54–S62. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.03.007.
- [43] H. van Beek, E. van Eijk, R. van Baar, M. Ugen, J. Bodde, and A. Siemelink. “Digital forensics as a service: Game on.” In: *Digital Investigation* 15 (2015). Special Issue: Big Data and Intelligent Data Analysis, pp. 20–38. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.07.004.

A. Creating a Map of User Data in NTFS to Improve File Carving

- [44] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. “The first collision for full SHA-1.” In: *Advances in Cryptology — CRYPTO 2017*. Vol. 10401. Lecture Notes in Computer Science. 2017, pp. 570–596. DOI: 10.1007/978-3-319-63688-7_19.
- [45] Cryptology Group at Centrum Wiskunde & Informatica (CWI) and Google Research Security, Privacy and Anti-abuse Group. *Shattered — We have broken SHA-1 in practice*. Last accessed 28-04-2018. URL: <https://shattered.io/>.
- [46] Real Data Corpus. *Real Data Corpus*. Last accessed 29-09-2018. July 2018. URL: <https://digitalcorpora.org/corpora/disk-images/real-data-corpora>.
- [47] S. Gibbs. *From Windows 1 to Windows 10: 29 years of Windows evolution*. Last accessed 29-09-2018. Oct. 2014. URL: <https://www.theguardian.com/technology/2014/oct/02/from-windows-1-to-windows-10-29-years-of-windows-evolution>.
- [48] C. Buckel. *Understanding Flash: The Flash Translation Layer*. Last accessed 08-10-2018. Sept. 2014. URL: <https://flashdba.com/2014/09/17/understanding-flash-the-flash-translation-layer/>.
- [49] R. Reiter, T. Swatosh, P. Hempstead, and M. Hicken. *Accessing logical-to-physical address translation data for solid state disks*. Last accessed 08-10-2018. Nov. 2014. URL: <http://www.freepatentsonline.com/8898371.html>.
- [50] J. Barbara. *Solid State Drives: Part 5*. Last accessed 08-10-2018. Apr. 2014. URL: <https://www.forensicmag.com/article/2014/04/solid-state-drives-part-5>.
- [51] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. “A Survey of Flash Translation Layer.” In: *J. Syst. Archit.* 55.5-6 (May 2009), pp. 332–343. DOI: 10.1016/j.sysarc.2009.03.005.
- [52] C. Buckel. *Understanding Flash: Blocks, Pages and Program Erases*. Last accessed 03-10-2018. 2014. URL: <https://flashdba.com/2014/06/20/understanding-flash-blocks-pages-and-program-erases/>.
- [53] Y. Gubanov and O. Afonin. *SSD and eMMC Forensics 2016, part 1*. Last accessed 07-10-2018. Apr. 2016. URL: <https://articles.forensicfocus.com/2016/04/20/ssd-and-emmc-forensics-2016/>.
- [54] Y. Gubanov and O. Afonin. *SSD and eMMC Forensics 2016, part 2*. Last accessed 07-10-2018. May 2016. URL: <https://articles.forensicfocus.com/2016/05/04/ssd-and-emmc-forensics-2016-part-2/>.

- [55] Y. Gubanov and O. Afonin. *SSD and eMMC Forensics 2016, part 3*. Last accessed 07-10-2018. June 2016. URL: <https://articles.forensicfocus.com/2016/06/07/ssd-and-emmc-forensics-2016-part-3/>.
- [56] Y. Gubanov and O. Afonin. *Recovering Evidence from SSD Drives in 2014: Understanding TRIM, Garbage Collection and Exclusions*. Last accessed 07-10-2018. 2014. URL: <https://articles.forensicfocus.com/2014/09/23/recovering-evidence-from-ssd-drives-in-2014-understanding-trim-garbage-collection-and-exclusions/>.
- [57] Y. Gubanov and O. Afonin. *Why SSD Drives Destroy Court Evidence, and What Can Be Done About It*. Last accessed 08-10-2018. 2012. URL: <https://belkasoft.com/download/info/SSD%20Forensics%202012.pdf>.
- [58] Net Applications.com. *Desktop Operating System Market Share*. Sept. 2017. URL: <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0>.
- [59] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005. ISBN: 0321268172.
- [60] Microsoft. *How NTFS Works*. Last accessed 30-09-2018. 2018. URL: [https://technet.microsoft.com/pt-pt/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx).
- [61] Microsoft. *Windows 7 system requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-us/help/10737/windows-7-system-requirements>.
- [62] Microsoft. *System requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-gb/help/12660/windows-8-system-requirements>.
- [63] Microsoft. *Windows 10 system requirements*. Last accessed 30-04-2018. Nov. 2017. URL: <https://support.microsoft.com/en-us/help/4028142/windows-windows-10-system-requirements>.
- [64] B. Carrier. *TSK Tool Overview*. 2014. URL: http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.
- [65] NTFS.com. *NTFS Partition Boot Sector*. Last accessed 08-10-2018. 2018. URL: <http://www.ntfs.com/ntfs-partition-boot-sector.htm>.
- [66] B. Schneier. *Applied Cryptography — Protocols, Algorithms, and Source Code in C*. 2nd ed. John Wiley & Sons, Inc., 1996.

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

The layout of the article has been lightly edited to fit the overall layout of the thesis. This text and a full citation of the article has been added below the title. Since the work is focused on disk partitions (Windows volumes) the term Logical Block Addressing (LBA) in the article has been corrected to LPVA where applicable. Therefore the X-axis of Figure B.5 has been changed to “Cluster #” and LBA removed from the caption. The content is otherwise unchanged and corresponds to the original, published, version.

M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Using NTFS Cluster Allocation Behavior to Find the Location of User Data.” In: *Digital Investigation* 29 (2019), S51–S60. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.018

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.

B.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 [1–5], 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 [6, 7]. 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 [8–13]. 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 [14–18].

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 [19] the following concepts are listed; data mining, data reduction and subsets, triage, intelligence analysis and digital intelligence, distributed and 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 New Technology File System (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 Partition Volume Address (LPVA) 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 (LPVA) level of the hard drive controller. In an SSDs the flash based physical storage is hidden by the Flash Translation Layer (FTL) [20–24], which also hides any wear leveling or other low level functions of the controller. If an SSD is accessed using Factory Access Mode (FAM) [20] 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 Master File Table (MFT) is extracted externally (via the host). This process is iterated 10000¹ 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 LPVA positions in the partitions.

The work presented in this article complements a previous article [25] 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 (SHA-1) 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 Introduction presents related work and our contributions. In Section Experiment we describe the experimental platform and how the experiment was implemented. Section Result presents the results of the experiment and in Section 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 Conclusion and

¹We 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.

future work concludes the work and presents ideas of future work to be done.

B.1.1. Background

Silberschatz et al. [26] 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. [26] 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 Random Access Memory (RAM) [26].

Based on the information given by Microsoft [27] and Hughes [28] Windows in combination with NTFS is using an 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 B²) in the meta data records themselves. Microsoft [27] states that the meta data in NTFS is held in the MFT, which in turn hold the MFT records associated with the files in the file system. According to Carrier [29] the best fit algorithm is used by Windows XP on NTFS formatted hard disks. Since the book

²The 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 to 700 bytes. Microsoft reports a 900 byte limit [27].

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 Logical Block Addressing (LBA) [30].

When formatting an NTFS partition 12.5% of the space is reserved for the MFT as default [27]. The MFT records 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 LPVA order. If a cluster is allocated the corresponding bit in the \$Bitmap file is set to 1, hence 0 represents an unallocated cluster.

B.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.

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 [31]. According to Garfinkel one of the first times hashes are used for file carving is during the Digital Forensic Research Workshop (DFRWS) 2006 Carving Challenge [14]. Later the spamsum tool is used as a basis for an article by Kornblum [32] 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. [33] 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. [18] 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 [14] 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 [17]. 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

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

and McCarrin [14] 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 [34].

Foster [16] discusses the problem of data shared across files, stating that “the block of NULs is the most common block in our corpus” [16, 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. [15] 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 [14, 16, 17]. 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.

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. [35] 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 [36] present 12 factors affecting data persistence in storage media. Fairbanks [37] and Fairbanks [38] also describes the low-level functions of ext4 and their effect on digital forensics.

Data reduction

Quick and Choo [3, 39] 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 [40] 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.” [40, p. 86]. The methods studied by Rowe all require a working file system, which is not consistent with the foundation of file carving.

Data mapping

Key [41] 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 [1] 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 [1] mention using the distribution of data on disk, but do not seem to relate that to the probability of finding user data at different LPVA positions in storage media.

In two articles by Baar et al. [42] and Beek et al. [43] outlining the DFaaS system Hansken [43] and its predecessor Xiraf [42] 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.

B.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 LPVA 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

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

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 SHA-1 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.

B.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 [44] on the nodes.

We run four virtual machines in each node, one for each version of Windows in our test (see Table D.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.

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 [45]. 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.

Table B.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

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.

³There 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 [45].

B.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 B.2). The selection of file operations is biased (weighted) and the size of the files varied

Table B.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

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 B.2 represents a file operation and the amount of a specific number relative to the total amount of operations gives its bias. The bias value of an operation is calculated as $bias_{op} = \frac{factor_{op}}{\sum factor_s}$. 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 B.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 B.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 B.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 Figure B.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

Table B.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

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 B.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

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

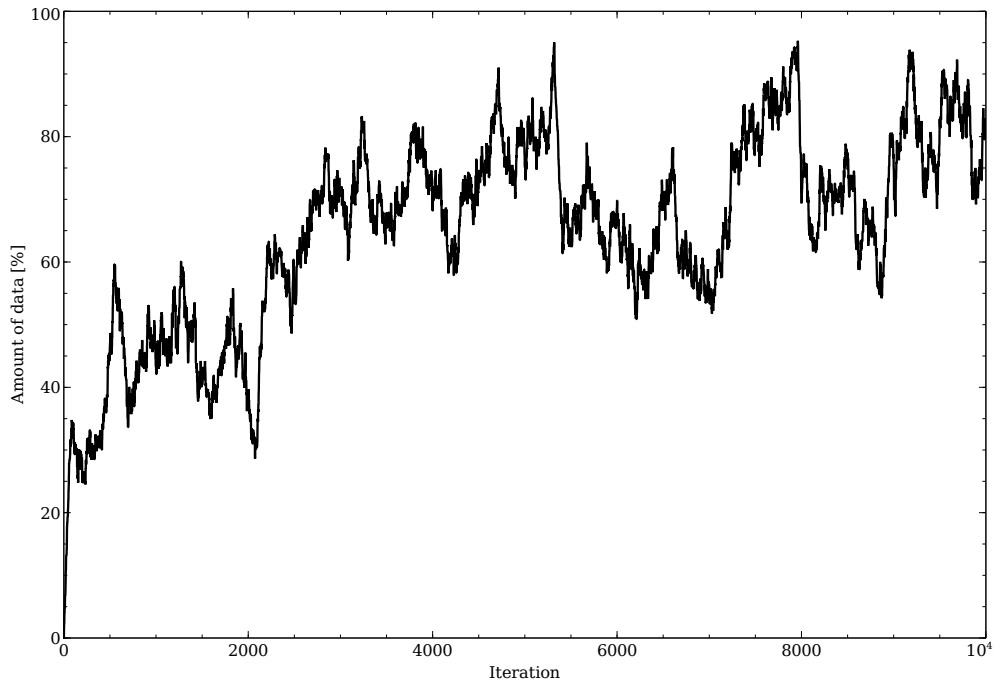


Figure B.1.: The simulated user behavior, i. e. the total amount of data stored in the NTFS partition after each iteration, of the 16 virtual machines that were used for the similarity test. The plot shows 10000 iterations. The Windows 8 machines were run at least 9035 iterations before being shut down,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

transactions if needed.

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 [44] 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 [44] or the `idifference` tool [46] 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` [44] to find what clusters each file allocates and then do a difference calculation. Consequently the `idifference` tool is of no use to us.

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

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! (EFS!)) affect the allocation strategy of the OS or file system, since they act either above or below the file system [47, 48]. 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 include 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.

B.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 LPVA 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 LPVA position and each allocation must be preceded by a deallocation. The remaining posts are then merge sorted in order of LPVA 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.

B.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.

In Figure B.2 a plot of the allocation frequency at different LPVA positions is shown. The plot is divided into 64 equally sized groups based on LPVA 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 [27] as default. We have not found any specification of the exact LPVA 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 Figure B.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 [49–51]. 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 Figure B.2.

Close to the middle of the partitions (see Figure B.2) there are large disturbances in the plots for all versions but Windows 10. According to [29] the 4 KiB `$MFTMirr` file, containing a backup of the first four files in the MFT, is located at the middle of an NTFS partition. This might be true in Windows XP⁴, but not for the newer versions of Windows in our experiment. In these machines we have found the `$MFTMirr` file

⁴We have checked a number of partitions downloaded from the [52]) indicating this.

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

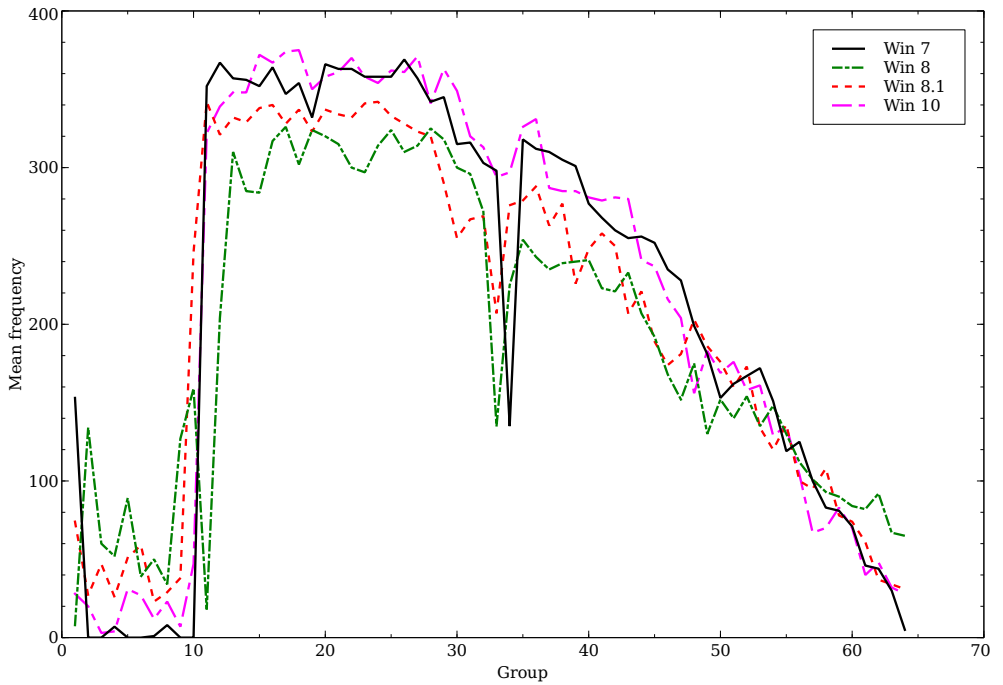


Figure B.2.: A map of the mean allocation frequency for all 32 virtual machines having NTFS formatted partitions in 64 GiB disks running Windows 7, 8, 8.1 and 10. The resolution is set to 64 groups.

to be located at different LPVA 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 Figure B.2.

The allocation frequency of the 16 virtual machines that were using the same file operation pattern can be seen in Figure B.3. Since two Windows 8.1 machines and three 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 Figure B.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 Figure B.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 Figure B.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 Figure B.2 and Figure B.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 [44] 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

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

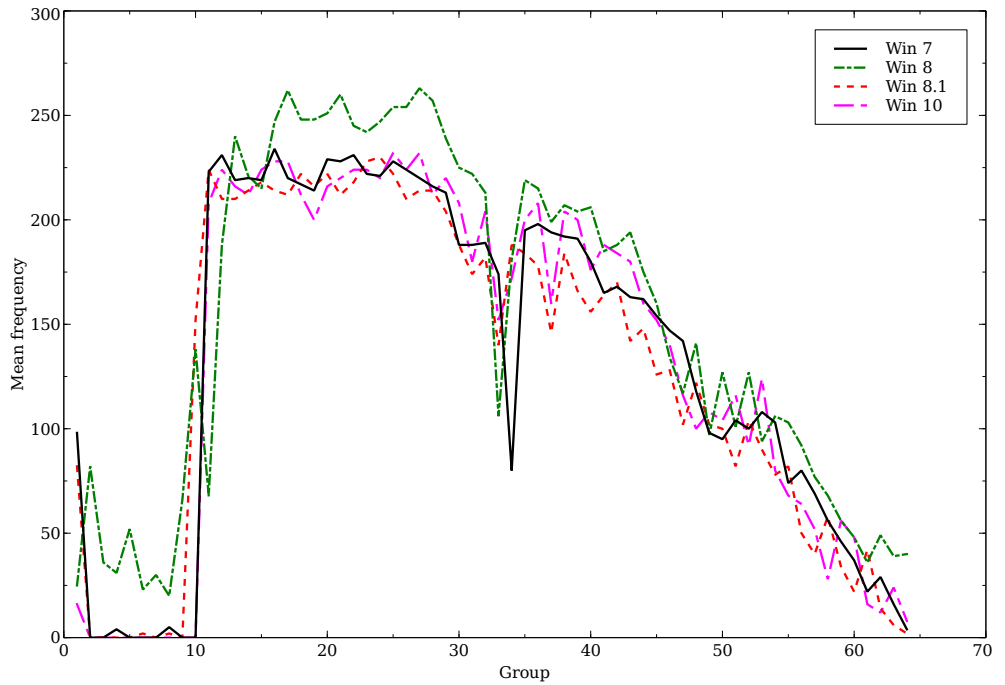


Figure B.3.: A map of the mean allocation frequency using the same file operation pattern of the 16 NTFS formatted partitions in 64 GiB disks running Windows 7, 8, 8.1 and 10. The resolution is set to 64 groups.

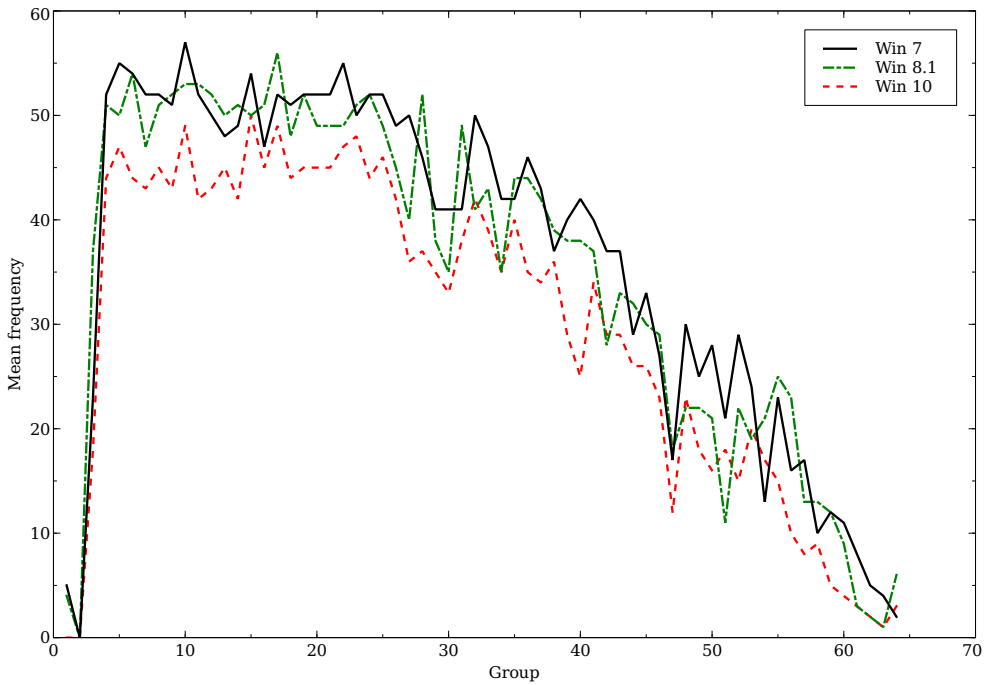


Figure B.4.: A map of the mean allocation frequency in NTFS formatted partitions in 256 GiB hard disks running Windows 7, 8.1 and 10 and using the same file operation pattern. The resolution is set to 64 groups.

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

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 Figure B.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.

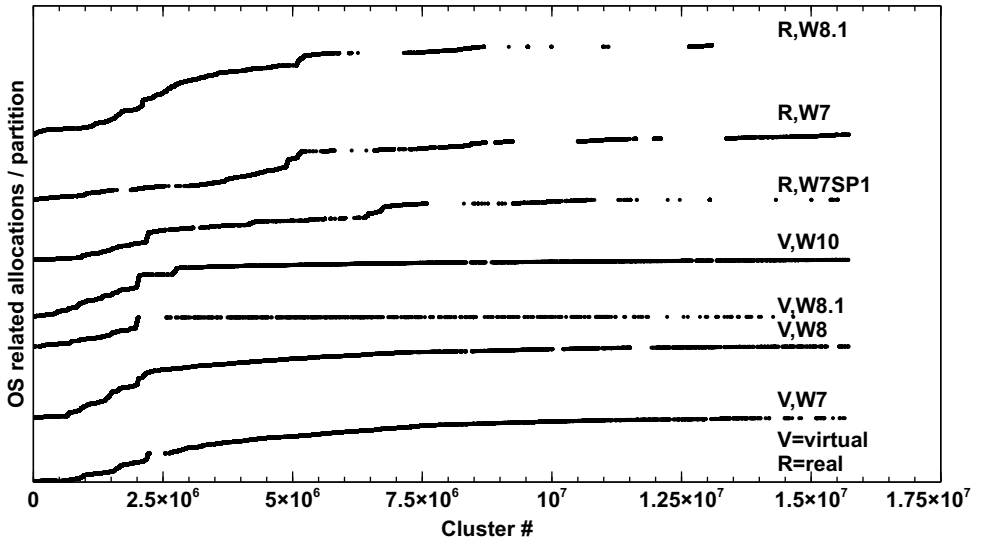


Figure B.5.: The density of OS related files at different positions in four virtual and three real-life hard disks. A steeper slope of a curve means a higher density of OS files and a lower slope represents a lower density. As can be seen the starting areas, up to approximately cluster $2.5 \cdot 10^6$, 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.

The three real-life hard disks in Figure B.5 have their almost vertical bend, which probably represents the end of the OS installation file area, at a higher LPVA 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 simple 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.

B.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 LPVA positions allocated to new or modified files is not optimal, but we had 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 [25].

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.

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

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 Figure B.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 [42, 43] 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.

B.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 LPVA 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 usable 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 LPVA 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 Carrier's book [29].

B.6. 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.

B.7. Bibliography

- [1] P. Gladyshev and J. James. "Decision-theoretic file carving." In: *Digital Investigation* 22.Supplement C (2017), pp. 46–61. ISSN: 1742-2876. DOI: 10.1016/j.diin.2017.08.001.
- [2] European Police Office (Europol). *Internet Organised Crime Threat Assessment (IOCTA) 2016*. Tech. rep. European Cybercrime Centre (EC3), 2016.
- [3] D. Quick and K. Choo. "Data reduction and data mining framework for digital forensic evidence: Storage, intelligence, review and archive." In: *Trends & Issues in Crime and Criminal Justice* 480 (Sept. 2014), pp. 1–11. ISSN: 1836-2206.
- [4] F. Breitingner, G. Stivaktakis, and H. Baier. "FRASH: A framework to test algorithms of similarity hashing." In: *Digital Investigation* 10.Supplement (2013). The Proceedings of the Thirteenth Annual DFRWS Conference, S50–S58. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.06.006.

- [5] V. Roussev. “Managing Terabyte-Scale Investigations with Similarity Digests.” In: *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*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 19–34. ISBN: 978-3-642-33962-2. DOI: 10.1007/978-3-642-33962-2_2.
- [6] R. Poisel and S. Tjoa. “A Comprehensive Literature Review of File Carving.” In: *2013 International Conference on Availability, Reliability and Security*. Sept. 2013, pp. 475–484. DOI: 10.1109/ARES.2013.62.
- [7] A. Pal and N. Memon. “The evolution of file carving.” In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [8] M. Karresand. “Completing the Picture — Fragments and Back Again.” Licentiate thesis. Linköping Institute of Technology, Linköping University, Sweden, May 2008.
- [9] M. Karresand and N. Shahmehri. “Reassembly of fragmented JPEG images containing restart markers.” In: *Proceedings - 4th Annual European Conference on Computer Network Defense, EC2ND 2008*. 2008, pp. 25–32. DOI: 10.1109/EC2ND.2008.10.
- [10] M. Karresand and N. Shahmehri. “Oscar — Using Byte Pairs to Find File Type and Camera Make of Data Fragments.” In: *Proceedings of the 2nd European Conference on Computer Network Defence, in conjunction with the First Workshop on Digital Forensics and Incident Analysis (EC2ND 2006)*. Ed. by A. Blyth and I. Sutherland. Springer Verlag, 2007, pp. 85–94. DOI: 10.1007/978-1-84628-750-3_9.
- [11] M. Karresand and N. Shahmehri. “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*. Piscataway, NJ, USA: IEEE, 2006, pp. 140–147. DOI: 10.1109/IAW.2006.1652088.
- [12] M. Karresand and N. Shahmehri. “Oscar — File Type and Camera Identification Using the Structure of Binary Data Fragments.” In: *Proceedings of the 1st Conference on Advances in Computer Security and Forensics, ACSF*. Ed. by J. Haggerty and M. Merabti. Liverpool, UK: The School of Computing and Mathematical Sciences, John Moores University, July 2006, pp. 11–20.

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

- [13] M. Karresand and N. Shahmehri. “Oscar — File Type Identification of Binary Data in Disk Clusters and RAM Pages.” In: *Security and Privacy in Dynamic Environments, Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22-24 May 2006, Karlstad, Sweden*. Vol. 201. Lecture Notes in Computer Science. Springer, 2006, pp. 413–424. DOI: 10.1007/0-387-33406-8_35.
- [14] S. Garfinkel and M. McCarrin. “Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb.” In: *Digital Investigation* 14.Supplement 1 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S95–S105. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.001.
- [15] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks. “Distinct Sector Hashes for Target File Detection.” In: *Computer* 45.12 (Dec. 2012), pp. 28–35. ISSN: 0018-9162. DOI: 10.1109/MC.2012.327.
- [16] K. Foster. “Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus.” MA thesis. Monterey, California, USA: Naval Postgraduate School, Sept. 2012.
- [17] S. Garfinkel, A. Nelson, D. White, and V. Roussev. “Using purpose-built functions and block hashes to enable small block and sub-file forensics.” In: *Digital Investigation* 7.Supplement (2010). The Proceedings of the Tenth Annual DFRWS Conference, S13–S23. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.003.
- [18] S. Collange, Y. S. Dandass, M. Daumas, and D. Defour. “Using Graphics Processors for Parallelizing Hash-Based Data Carving.” In: *2009 42nd Hawaii International Conference on System Sciences*. Jan. 2009, pp. 1–10. DOI: 10.1109/HICSS.2009.494.
- [19] D. Quick and K. Choo. “Impacts of increasing volume of digital forensic data: A survey and future research challenges.” In: *Digital Investigation* 11.4 (2014), pp. 273–294. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.09.002.
- [20] O. Afonin. *Life after Trim: Using Factory Access Mode for Imaging SSD Drives*. Last accessed 15-03-2019. Jan. 2019. URL: <https://blog.elcomsoft.com/2019/01/life-after-trim-using-factory-access-mode-for-imaging-ssd-drives/>.
- [21] C. Buckel. *Understanding Flash: The Flash Translation Layer*. Last accessed 08-10-2018. Sept. 2014. URL: <https://flashdba.com/2014/09/17/understanding-flash-the-flash-translation-layer/>.

- [22] R. Reiter, T. Swatosh, P. Hempstead, and M. Hicken. *Accessing logical-to-physical address translation data for solid state disks*. Last accessed 08-10-2018. Nov. 2014. URL: <http://www.freepatentsonline.com/8898371.html>.
- [23] J. Barbara. *Solid State Drives: Part 5*. Last accessed 08-10-2018. Apr. 2014. URL: <https://www.forensicmag.com/article/2014/04/solid-state-drives-part-5>.
- [24] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. “A Survey of Flash Translation Layer.” In: *J. Syst. Archit.* 55.5-6 (May 2009), pp. 332–343. DOI: 10.1016/j.sysarc.2009.03.005.
- [25] M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8.
- [26] A. Silberschatz, P. Galvin, and G. Gagne. *Operating System Concepts*. 9th ed. Wiley, Dec. 2012.
- [27] Microsoft. *How NTFS Works*. Last accessed 30-09-2018. 2018. URL: [https://technet.microsoft.com/pt-pt/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx).
- [28] J. Hughes. *The Four Stages of NTFS File Growth*. Last accessed 24-10-2018. Oct. 2009. URL: <https://blogs.technet.microsoft.com/askcore/2009/10/16/the-four-stages-of-ntfs-file-growth/>.
- [29] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005. ISBN: 0321268172.
- [30] maaartinus, L. Osterman, and PC Guru. *What block allocation algorithm does NTFS use?* Last accessed 24-01-2019. Mar. 2017. URL: <https://superuser.com/questions/274855/what-block-allocation-algorithm-does-ntfs-use>.
- [31] A. Tridgell. *Spamsum README*. Last accessed 27-04-2018. 2002. URL: <https://www.samba.org/ftp/unpacked/junkcode/spamsum/README>.
- [32] J. Kornblum. “Identifying almost identical files using context triggered piecewise hashing.” In: *Digital Investigation* 3. Supplement (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS ’06), pp. 91–97. ISSN: 1742-2876. DOI: 10.1016/j.diin.2006.06.015.

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

- [33] Y. Dandass, N. Necaie, and S. Thomas. "An Empirical Analysis of Disk Sector Hashes for Data Carving." In: *J. Digit. Forensic Pract.* 2.2 (Apr. 2008), pp. 95–104. ISSN: 1556-7281. DOI: 10.1080/15567280802050436.
- [34] Microsoft. *Default cluster size for NTFS, FAT, and exFAT*. Aug. 2015. URL: <https://support.microsoft.com/en-us/help/140365/default-cluster-size-for-ntfs-fat-and-exfat>.
- [35] J. Jones, T. Khan, K. Laskey, A. Nelson, M. Laamanen, and D. White. "Inferring Previously Uninstalled Applications from Residual Partial Artifacts." In: *Annual ADFSL Conference on Digital Forensics, Security and Law*. 2016, pp. 113–130.
- [36] K. Fairbanks and S. Garfinkel. "Column: Factors Affecting Data Decay." In: *Journal of Digital Forensics, Security and Law* 7 (2012). DOI: 10.15394/jdfsl.2012.1116.
- [37] K. Fairbanks. "A Technique for Measuring Data Persistence Using the Ext4 File System Journal." In: *2015 IEEE 39th Annual Computer Software and Applications Conference*. Vol. 3. July 2015, pp. 18–23. DOI: 10.1109/COMPASAC.2015.164.
- [38] K. Fairbanks. "An analysis of Ext4 for digital forensics." In: *Digital Investigation* 9. Supplement (2012). The Proceedings of the Twelfth Annual DFRWS Conference, S118–S130. ISSN: 1742-2876. DOI: 10.1016/j.diin.2012.05.010.
- [39] D. Quick and K.-K. R. Choo. "Big forensic data reduction: digital forensic images and electronic evidence." In: *Cluster Computing* 19.2 (June 2016), pp. 723–740. ISSN: 1573-7543. DOI: 10.1007/s10586-016-0553-1.
- [40] N. C. Rowe. "Identifying Forensically Uninteresting Files Using a Large Corpus." In: *Digital Forensics and Cyber Crime: Fifth International Conference, ICDF2C 2013, Moscow, Russia, September 26-27, 2013, Revised Selected Papers*. Ed. by P. Gladyshev, A. Marrington, and I. Baggili. Cham: Springer International Publishing, 2014, pp. 86–101. ISBN: 978-3-319-14289-0. DOI: 10.1007/978-3-319-14289-0_7.
- [41] S. Key. *File Block Hash Map Analysis*. Last accessed 28-04-2018. 2012. URL: <https://www.guidancesoftware.com/app/File-Block-Hash-Map-Analysis>.
- [42] R. van Baar, H. van Beek, and E. van Eijk. "Digital Forensics as a Service: A game changer." In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S54–S62. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.03.007.

- [43] H. van Beek, E. van Eijk, R. van Baar, M. Ugen, J. Bodde, and A. Siemelink. “Digital forensics as a service: Game on.” In: *Digital Investigation* 15 (2015). Special Issue: Big Data and Intelligent Data Analysis, pp. 20–38. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.07.004.
- [44] B. Carrier. *TSK Tool Overview*. 2014. URL: http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.
- [45] TerryE and mpack. *All about VDIs*. Last accessed 30-12-2018. Feb. 2018. URL: <https://forums.virtualbox.org/viewtopic.php?t=8046>.
- [46] S. Garfinkel and K. Fairbanks. *sleuthkit/tools/fiwalk/*. Last accessed 30-12-2018. Apr. 2012. URL: <https://github.com/kfairbanks/sleuthkit/tree/master/tools/fiwalk>.
- [47] Arch Linux. *Disk encryption*. Last accessed 30-12-2018. Nov. 2018. URL: https://wiki.archlinux.org/index.php/disk_encryption.
- [48] M. Gattol. *Block-layer Encryption*. Last accessed 24-01-2019. Jan. 2015. URL: https://web.archive.org/web/20150917051251/http://www.markus-gattol.name/ws/dm-crypt_luks.html.
- [49] Microsoft. *Windows 10 system requirements*. Last accessed 30-04-2018. Nov. 2017. URL: <https://support.microsoft.com/en-us/help/4028142/windows-windows-10-system-requirements>.
- [50] Microsoft. *System requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-gb/help/12660/windows-8-system-requirements>.
- [51] Microsoft. *Windows 7 system requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-us/help/10737/windows-7-system-requirements>.
- [52] Real Data Corpus. *Real Data Corpus*. Last accessed 29-09-2018. July 2018. URL: <https://digitalcorpora.org/corpora/disk-images/real-data-corporus>.

C. Disk Cluster Allocation Behavior in Windows and NTFS

The layout of the article has been lightly edited to fit the overall layout of the thesis. This text and a full citation of the article has been added below the title. Since the work is focused on disk partitions (Windows volumes) the term LBA in the article has been corrected to LPVA where applicable. The content is otherwise unchanged and corresponds to the original, published, version.

The article was originally accepted to CARDS 2019 - 11th EAI International Conference on Cyber Attacks Response and Defense (formerly ICDF2C). For reasons beyond our control the conference was canceled. The organizing committee therefore offered us a publication of the article in the journal *Mobile Networks and Applications*. We accepted the offer, even though the journal had a different scope and aim than the CARDS conference.

M. Karresand, S. Axelsson, and G. Dyrkolbotn. "Disk Cluster Allocation Behavior in Windows and NTFS." in: *Mobile Networks and Applications* 25.1 (Feb. 2020), pp. 248–258. ISSN: 1572-8153. DOI: 10.1007/s11036-019-01441-1

Abstract

The allocation algorithm of a file system has a huge impact on almost all aspects of digital forensics, because it determines where data is placed on storage media. Yet there is only basic information available on the allocation algorithm of the currently most widely spread file system; New Technology File System (NTFS). We have therefore studied the NTFS allocation algorithm and its behavior empirically. To do that we used two virtual machines running Windows 7 and 10 on NTFS formatted fixed size virtual hard disks, the first being 64 GiB and the latter 1 TiB in size. Files of different sizes were written to disk using two writing strategies and the \$Bitmap files were manipulated to emulate file system fragmentation.

Our results show that files written as one large block are allocated areas of decreasing size when the files are fragmented. The decrease in size is seen not only within files, but also between them. Hence a file having smaller fragments than another file is written after the file having larger fragments. We also found that a file written as a stream gets the opposite allocation behavior, i. e. its fragments are increasing in size as the file is

written. The first allocated unit of a stream written file is always very small and hence easy to identify.

The results of the experiment are of importance to the digital forensics field and will help improve the efficiency of for example file carving and timestamp verification.

C.1. Introduction

File carving [1, 2] and timestamps are two key concepts in digital forensics. Unfortunately both concepts are laden with complex operations and a large amount of uncertainty regarding the correctness of the results. In file carving the digital forensic investigator has to find and categorize a large amount of (fragmented) data and then put the fragments back into the original file again without the help of a file system. The same applies to timestamps, where a timeline must be constructed, often only with data from different log files and the file system meta data at hand, two sources that are easy to manipulate. In both cases a digital forensic investigator would be helped by the extra information hidden in the storage structure of data, its allocation layout.

The file carving process and the extraction of timestamps are both dependent on the behavior of the combination of file system and operating system (OS), which govern the placement of data on hard disks through an allocation algorithm. Knowledge of the layout pattern of data on disk is crucial to the forensic investigator when doing file carving. However, the actual behavior of the allocation algorithm is not known. For example the general assumption used as a basis for different forensic file carving tools is that the data to be carved is stored contiguously and is only mildly fragmented.

To improve the timestamp information used in investigations several propositions have been made to also use the behavior of the allocation algorithm as a source [3, 4]. However, we have not found any previous work that study the exact behavior of the allocation algorithm of for example New Technology File System (NTFS). There is a small number of main allocation algorithm concepts used in all modern OSs, but the exact behavior of the different implementations of the algorithms are not known, at least not for the closed source OS variants. Windows using NTFS is for example said to use a best fit allocation strategy, but that information is getting dated and is also based on the Linux implementation of the NTFS driver [5]. We therefore have studied the allocation behavior of two modern versions of Windows (7 and 10) in combination with NTFS to empirically reverse engineer the allocation algorithm(s) used. The data source is based on writing files of different sizes to disk using different writing strategies (writing the file as one large block at once or as a continuous stream of data). The experiment was done using two virtual machines having fixed size virtual hard disks of different sizes, one 64 GiB and one 1 TiB.

The experiment is part of the future work presented in two earlier articles [6, 7]

where we develop a framework to create maps of user data placement on hard disks. The maps give the probability of finding unique user data at different Logical Partition Volume Address (LPVA) positions in Windows NTFS partitions and can for example be used for triage, planning of hard disk investigations and to enable different areas to be prioritized in file carving processes. Knowledge of the actual behavior of the allocation algorithm of different file systems will enhance the precision of the maps.

Our work is based on an empirical evaluation of the behavior of the allocation algorithm of Windows NTFS, using the hypothesis that a best fit allocation strategy is used. Hence we do not reverse engineer any code or expose any secret information. Likewise the reported results are not detailed enough to enable someone to recreate the Microsoft Windows NTFS allocation algorithm or driver. However, the presented results are of great value to the digital forensics community and we therefore make them public as a help in the global fight against digital crime.

The rest of this paper is organized as follows: The remaining parts of Section D.1 presents background on file allocation algorithms and related work. In Section C.2 we describe how the experiment was implemented. Section D.3 presents the results of the experiment and in Section D.4 we discuss the effects and implications of our result to the research field of file carving and other relevant areas within and related to digital forensics. Section D.5 concludes the work and presents ideas of future work to be done.

C.1.1. Background

The theory of file system construction is for example described by Silberschatz et. al [8], Stallings [9] and Tanenbaum and Bos [10]. A file system keeps track of data stored on secondary storage and is organized in different ways. However, all implementations share some common properties: the addressing of the physical storage is abstracted by the file system into logical addresses and the logical storage position of written data is determined by an allocation algorithm.

During the history of computer systems different methods of allocating disk space have been in use. The main methods are contiguous, linked and indexed allocation [8–10]. The indexed allocation strategy, where the addresses of the data blocks are held in an index separated from a file's data, is currently the most popular. The indexed allocation strategy does not suffer from external fragmentation (unallocated holes being too small to be filled with new data), but heavily used storage media can still lead to fragmentation of files and require regular defragmentation. There is also a risk of disk space being wasted when using indexed allocation, 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 [8] presents three algorithms; *first fit*, *best fit* and *worst fit*, Stallings [9] mentions *nearest fit* and Tanenbaum and Bos [10] add *next*

C. Disk Cluster Allocation Behavior in Windows and NTFS

fit and *quick fit*. Together they give the following free space allocation algorithms:

First fit starts the search for free space at the beginning of the file system and has the requirement of the space being large enough to hold the entire file. If there is no space large enough to hold the entire file, a fragment is written at the found position and the search continues.

Next fit (Stallings [9, p. 544] calls this *nearest fit*) uses the same principle as *first fit* of allocating the next free space being large enough to hold the entire file, but the search is done from the current position in the file system. If there is no single free space to hold the entire file the file is fragmented and the available free spaces are used to hold the file.

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

Worst fit uses the free space having the worst fit to the new file. This is the opposite to best fit, i. e. giving the largest remaining free space. As for the best fit algorithm the available free spaces of the entire file system have to be scanned before the worst fit can be chosen.

Quick fit uses several lists of different sizes of free block areas. In this way a fitting area can be found very quickly, but the algorithm suffers from a complex process during deallocation when freed up areas might have to be merged. If this is not done the storage will soon fragment into a large amount of free areas too small to be usable.

These free space allocation algorithms are not specific to storage of data on disk, they are also used in for example memory allocation in Random Access Memory (RAM) [8].

Based on the information given by [11] and [12] Microsoft Windows' NTFS is using an index allocation strategy. The problem of space being wasted when using index allocation is solved in NTFS by storing the data of smaller files (up to approximately 700 bytes¹) in the MFT meta data records themselves [12].

According to Carrier [5] the best fit free space 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. Our previous experiments [6, 7] indicate that the actual behavior of the allocation algorithm in NTFS is not strictly best

¹The maximum size of an internal \$DATA attribute varies depending on the size of other attributes stored in the Master File Table (MFT) record. Most sources give a maximum internal \$DATA attribute size of 600 to 700 bytes. Microsoft reports a 900 byte limit [12].

fit. The results from experiments indicate that groups of free clusters are allocated in descending order of size, which is best fit, but not always. The deviating behavior is mentioned in a Superuser Q&A [13].

When formatting an NTFS partition 12.5% of the space is by default reserved for the MFT [12]. The MFT records are 1 KiB in size and usually the size of the smallest allocatable unit (called cluster) in NTFS is 4 KiB [12]. 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 file system cluster in ascending LPVA order. If a cluster is allocated the corresponding bit in the \$Bitmap file is set to 1, hence 0 represents an unallocated cluster.

Files can be written to disk in two ways; either as a stream, or the entire file as a block. In the first case the OS does not know the final size of the file and therefore cannot optimize the allocation accordingly. This often leads to file fragmentation, but the behavior is partly mitigated by the internal buffering of the OS. When the entire file is written in one piece the OS knows its size in advance and can utilize the standard allocation algorithm, which optimizes the storage. This behavior is probably more common when dealing with smaller files that easily can be held in RAM, than for large files. The specific write behavior is software dependent and might incorporate temporary writing of files to protect the data in case of a power loss or hardware failure.

C.1.2. Related work

We have not found any related work directly addressing the detailed behavior of the allocation algorithm in Microsoft Windows 7 and 10 running in NTFS formatted partitions. There are however some work done on the connection between timestamps, cluster allocation and file creation order.

Willassen studies the formal foundation of timestamps in his PhD dissertation [4]. There he also formulates the criteria needed to use the allocation strategy of a file system for checking timestamps. He also briefly discusses the allocation strategy of NTFS and states that it is best fit in Windows XP and that a first fit allocation strategy is used within the MFT. The PhD dissertation is partially based on an earlier article by Willassen [3] describing a method to use a first fit allocation strategy to detect antedating of files.

Tse [14] intends to use the allocation pattern of files to estimate the causal ordering of events, but concludes that using the actual allocation pattern is complex. Tse instead defines two metrics approximating the allocation pattern causality and then tests the metrics against standard timestamps.

Minnaard [15] studies the inner workings of the Linux implementation of the File Allocation Table (FAT) file system in an article from 2014 by looking at the source code and then uses the result to find the causal order of files from a TomTom Go. The

article mentions that the Windows 7 implementation of FAT is more complex than the Linux variant.

C.2. Experimental setup

The experiment is designed to test whether Windows in combination with NTFS is using the best fit allocation strategy as indicated in the literature [4, 5]. To enable this the behavior of the allocation algorithm has to be studied in different situations regarding disk utilization, file system fragmentation, file size and partition size. We therefore manipulated the \$Bitmap file of the file systems of the virtual machines used to emulate different states of file system fragmentation.

C.2.1. Virtual hardware

The experiments were performed on two virtual machines using VirtualBox running Windows 7 and Windows 10. The Windows 7 machine had a fixed size 64 GiB hard disk and the Windows 10 machine a 1 TiB ditto to cover both older and newer Windows OSs, as well as small and large hard disks. Each virtual machine was given a folder shared with the host. To enable us to use standard digital forensic tools (the Sleuth Kit [16]) on the virtual hard disks and their partitions they were loop mounted with read/write access rights.

The disk of the Windows 7 machine was already used since it was taken from one of our earlier experiments [6], where 10000 file operations were performed in a pseudo-random pattern with bias towards file writing. This had given a heavily fragmented file system corresponding to an old and well used (home) computer. The fragmentation status of the file system was checked before the start of the experiment and one remaining large area of free clusters was found at the end of the partition. To even out the fragmentation pattern we decided to split that area into five smaller areas of approximately 262000 clusters each, corresponding to approximately 1 GiB per area.

The Windows 10 machine was freshly installed and the file system therefore only contained files from the installation, without any significant fragmentation. It represented an office computer using mainly network based storage, without synchronization between local and network storage.

C.2.2. Process description

During the experiment each virtual machine was repeatedly power cycled to ensure that all file operations were flushed to hard disk (both the virtual and the real on the host). The experiment started by copying the \$Bitmap file of the powered down virtual

machine to the host. Then the following steps were iterated over for each file write operation:

1. The virtual machine was powered on
2. A file signaling the power on sequence had finished was written to the shared folder
3. The file write operation was executed
4. A file signaling the file operation had finished was written to the shared folder
5. The virtual machine was powered down
6. The power down status was checked using `vboxmanage`
7. The allocation information of the newly written file was extracted using the `istat` [16] tool
8. The \$Bitmap file of the virtual machine was copied to the host using `icat` [16] tool

To allow the virtual machine, as well as the host, to properly write all files to disk up to 10 second long delays were introduced between the steps 1 to 3 of each iteration. We also used files written to a shared folder to signal when a step was finished.

To cover for both writing a file as one block and writing it as a stream a Python 2.7 script was used to either write a file to an array in RAM and then to disk (block writing) or using Python's own file write operation directly writing one 512 B sector at a time (stream writing). Both types of write operations were buffered by the OS, but in the block writing case the OS got information on the file size before the file was written to disk, which it did not when the file was written as a stream. Each 512 byte block of the files were uniquely marked with a consecutive number together with a file identifier. The marking was used as a backup procedure in case we had to read the raw data from the virtual hard disks due to errors in the process. The marking was also used to verify that the `istat` tool reported the allocation pattern in the same order as the clusters were written to, which it did.

We used files of different sizes to determine if the allocation algorithm behaved differently depending on the size of the written file. In the first round of the experiment we used 4, 128, 511, 512, 513 and 1024 MiB files. The 511 and 513 MiB files originated from the first version of the Python script, where the type of writing was depending on the size of the file. If the file size was ≤ 512 MiB the file was written as one block and if it was larger it was written as a stream. The range of file sizes were later expanded to also include 12, 96, 384, 768 and 1536 MiB files to get a more even coverage of small

and large files and the Python script was updated to enable different writing strategies regardless of file size.

C.2.3. Bitmap manipulation

To cover for possibly different allocation behavior depending on the available amount of storage of the file system we manipulated the \$Bitmap file of the Windows NTFS virtual hard disks. The \$Bitmap file of the Windows 7 machine was manipulated once and the Windows 10 \$Bitmap file three times.

The Windows 7 virtual machine’s main partition was heavily fragmented from the beginning, but still contained a contiguous area of approximately 5.3 GiB. This area was divided into five smaller areas, which can be seen in Table C.1, all other free areas were kept in their original state. The modified layout is called *BM 7:1* throughout the text. The unmodified layout was never used due to its already heavily fragmented file system.

Table C.1.: The unallocated areas in the original 5.3 GiB space after the *BM 7:1* manipulation.

Start position	Size [cluster]
15372418	262000
15634546	262143
15896818	262144
16159090	262145
16421363	262160

Directly after installation of the Windows 10 virtual machine its 1 TiB main partition contained two large unallocated areas of approximately 497 and 511 GiB respectively at the end of the partition. This original, unmodified, layout is called *BM 10:0* throughout the text.

After the *BM 10:0* file writing operations were executed we manipulated the \$Bitmap file to fragment the allocation layout. The smaller of the two large free areas was divided into 501 equally sized unallocated areas of 120000 clusters (468.75 MiB) each and the larger unallocated area was divided into 1026 unallocated areas of increasing size, from 120 clusters (480 KiB) to 123120 clusters (approximately 481 MiB) in steps of 120. The two areas together contained 123342120 (approximately 471 GiB) free clusters after the modification. All other unallocated areas on the partition were unmodified. We refer to this manipulation setting as *BM 10:1*

After testing the *BM 10:1* allocation manipulation we created a new bitmap (referred to as *BM 10:2*) file for the Windows 10 virtual hard disk where we decreased the available space even more by first restoring the virtual machine to its original state and then creating the free areas shown in Table C.2 from the two large unallocated spaces. The *BM 10:2* manipulation was meant to test the block writing allocation behavior by

Table C.2.: The modified areas of *BM 10:2*.

Spaces	Function	Tot. [cluster]
511	$\text{int}(120000/x + 27; x = [512 : -1 : 2])$	711541
23	120000	2760000
512	$7x + 13; x = [0 : 511]$	922368
16	7999	127984
1	17	17
29	$\text{int}(1800000/x + 17; x = [1 : 29])$	7131462

forcing the algorithm to chose between a few very large areas and many small. The seemingly odd values used in Table C.2 were chosen to avoid creating free areas of exact multiples of 2. The total amount of free space in the manipulated area was 11715760 clusters (44.7 GiB) after the modification.

However, the *BM 10:2* \$Bitmap manipulation was not strict enough to generate any significant fragmentation during the block writing. We therefore manually decreased any remaining free areas larger than 120000 consecutive clusters with a factor 10. This left a total of 3361315 clusters (12.8 GiB) of free space in the manipulated area. This last manipulation is referred to as *BM 10:3*.

C.3. Result

The overall result of the experiment shows that Windows 7 and 10 using NTFS formatted partitions are both using a best fit allocation algorithm. This is however not always true, as will be shown in the following sections. In the text we will refer to *lower rest of free clusters* as a term for allocation patterns where the pattern gives a lower number of remaining free clusters compared to a strict best fit pattern. Please observe that the term is only defining a local minimum, i. e. we have only looked at one alternative pattern.

C.3.1. Block writing

The result of the block writing operations are found to be following the best fit allocation strategy in most of the cases in Windows 10, but not for Windows 7. We also present an interesting pattern common to both versions of Windows regarding the sizes of fragments.

Windows 7

The result of the block writing file operations on the 64 GiB hard disk of the Windows 7 virtual machine using *BM 7:1* can be seen in Figure C.1. The file system was already partially fragmented due to 10000 random file operations executed in an earlier experiment and we fragmented the remaining large free area into five smaller ones to put an even higher burden on the allocation algorithm.

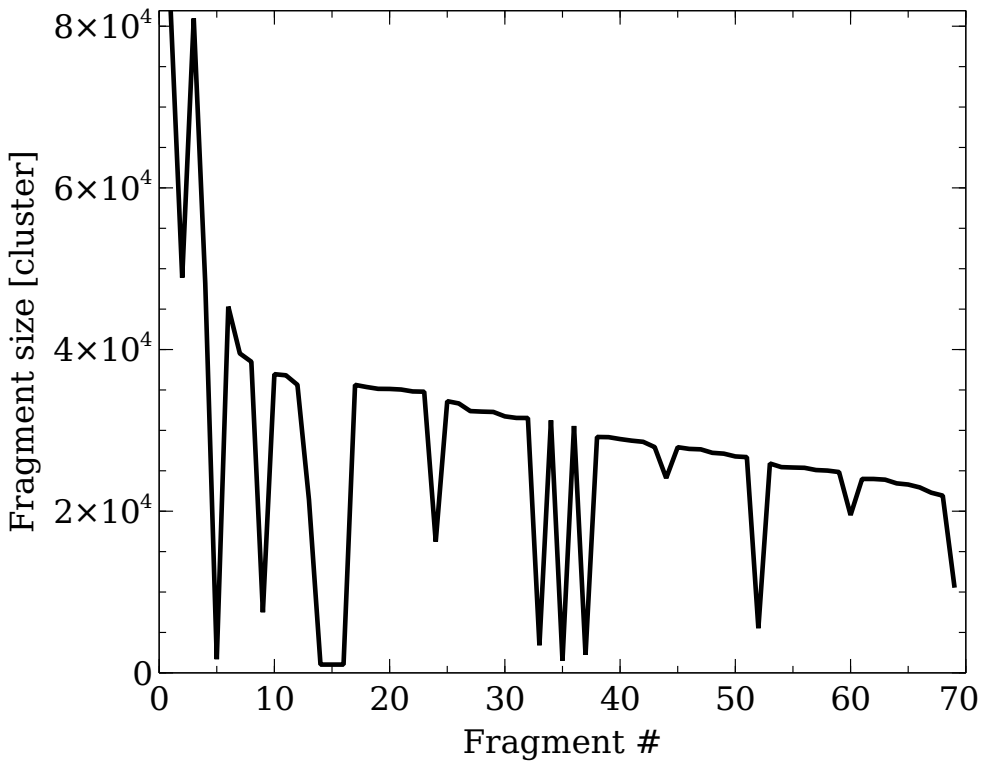


Figure C.1.: The decrease in fragment size for the Windows 7 block writing experiment using the *BM 7:1* layout.

All but three block writing file operations result in between 2 and 9 fragments, which are all allocated in descending order of size within each file operation (apart for file operations 5 to 7 (fragments 14 to 16 in Figure C.1), which are small and unfragmented). None of the fragmented file operations are best fit allocations and neither are they optimized from the point of lower rest of free clusters. The last fragment in every file operation is smaller than the previous fragments in the operation, which is represented by the dips in the curve in Figure C.1. What also can be noticed is the decreasing size of the fragments even between file operations, where the size of the second last fragment in the previous file operation is larger than the starting fragment size in the following file operation. Also the free areas chosen by the algorithm are often slightly larger (< 10 clusters) than the fragment and therefore free space might be wasted.

The continuously decreasing fragment sizes seen in Figure C.1 might seem to indicate a typical best fit behavior, but there were larger free areas that would have been better to use from a best fit point of view. Instead the algorithm seems to balance the sizes of the fragments to be as similar as possible, apart from the last fragment. We did not see any pattern regarding the position of the chosen fragments.

Windows 10

The original installation (*BM 10:0*) of Windows 10 on a 1 TiB virtual hard disk contains two large areas of free clusters at the end separated by a 512 MiB large area holding a number of files related to the System Volume Information directory. The two areas are the largest free areas on disk and over 500 times larger than the third largest area. The 11 largest free areas on the partition can be seen in Table C.3.

All file operations using *BM 10:0* allocated files of one block each, which can be seen in Table C.4. The first and second write operations are best fit allocations, as can be seen when comparing their sizes and positions to the available free areas in Table C.3. The rest of the file write operations all allocate consecutive areas in the second largest free area, which is not a strict best fit behavior. Even the three 4 MiB file write operations (operation 3 to 5 in Table C.4) are allocated in the same area and not in any of the better fitting available areas. Although there are system related and stream writing file operations executed between operations 3 to 5 there are several free areas that would fit the file sizes of these operations better than the actual allocation do.

After the file write operations in *BM 10:0* 19 block write operations are performed using the *BM 10:1* allocation layout. The larger write operations are split into $n \geq 2$ fragments, where the $n - 1$ first fragments all have approximately the same size within each operation. The largest available free area is 123120 clusters in size and as can be seen in Table C.5 that area is the first to be allocated in this part of the experiment.

All positions in Table C.5 ending in “4544” are free areas created by us. The larger

C. Disk Cluster Allocation Behavior in Windows and NTFS

Table C.3.: The 11 largest free areas in the original installation of Windows 10 on a 1 TiB hard disk (*BM 10:0*).

Position [cluster]	Free size [cluster]
150728	869
135305	1259
2009009	1679
2005999	3009
2010944	3742
2570502	4774
56466	14272
809984	27680
2775255	239688
3021071	131132401
134284544	134022399

Table C.4.: The 8 block write operations performed using the original allocation layout *BM 10:0*. The operations are presented in order of execution.

File op.	Position [cluster]	Size [cluster]
1	2775255	131072
2	3148190	131072
3	3280413	1024
4	3283997	1024
5	3282700	1024
6	3291406	262144
7	3815408	32768
8	3882725	32768

areas belonging to the “4544” series are also allocated in descending order, hence they are allocated in best fit order. Also the smaller, remaining parts, of the file write operations belonging to the “4544” series are allocated in best fit order, although some of them do not fully occupy the original free area. The unfragmented file operation 6 is also a best fit allocation due to a number of previous system file deallocations that left a free area of suitable size.

The part of the experiment using the *BM 10:2* allocation layout results in single block allocations and all of them are made in order of best fit. Thus even eight 768 MiB files all result in single block allocations. Worth noticing is the fact that all but one allocation are made within the modified area, i. e. not in any of the original free areas. The best fit allocation approach is proven by the fact that an area outside of the modified area is used in a sequence of equally sized files. That area has been available since the installation of the OS and has a size that fits in between the sizes of the modified free areas.

The execution of the file operations using the *BM 10:3* allocation layout gives files fragmented into 3, 4 and 5 fragments, as can be seen in Table C.7. The allocation strategy is no longer strictly best fit. Fragment 2 in file operation 1 is not best fit, because the free area at position 264466712 would have been better to use, as can be seen in Table C.6 showing the 21 largest free areas available to file allocation 1. The second file allocation in Table C.7 is strictly best fit, fragments 0 to 2 belong to the largest available free areas in that round and fragment 3 to the free area best fitting the remainder. The last file allocation, number 3 in Table C.7 is not best fit at a first glance. There is a fragment of 56801 clusters that would have been better to use. However, that free area would then have been combined with an area leaving 1589 free clusters. The current allocation only leaves 205 clusters unallocated and therefore has a lower rest of free clusters than a potential strict best fit allocation. If this is the actual behavior of the allocation algorithm is however not possible to deduce from only one file operation.

General observations

We have not been able to see any patterns in the allocation algorithm’s behavior regarding how the allocated positions are chosen. Likewise we have not been able to find an algorithm for how the fragment sizes are chosen. In most cases the fragments are somewhat smaller (a few clusters) than the free area, leaving small free areas before and/or after the fragment. Hence the the positioning and size of the fragments within the free areas are probably governed by some rules, but we have to few data to fully deduce them.

All fragmented block writing operations have one feature in common, which is the globally decreasing fragment size. The second last fragment in a file operation is always larger than the first fragment in the following file operation. The decreasing size

C. Disk Cluster Allocation Behavior in Windows and NTFS

Table C.5.: The 19 block write operations performed using the *BM 10:1* allocation layout. The operations are presented in order of execution.

Op.	Frag.	Pos. [cluster]	Size [cluster]
1	0	267534544	123120
1	1	142864544	7952
2	0	267404544	123000
2	1	142994544	8072
3	0	267274544	122880
3	1	267144544	122760
3	2	152094544	16504
4	0	267014544	122640
4	1	266884544	122520
4	2	152614544	16984
5	0	137534544	3072
6	0	3553550	3072
7	0	160804544	24576
8	0	137664544	3072
9	0	160934544	24576
10	0	161064544	24576
11	0	161194544	24576
12	0	240754544	98304
13	0	240884544	98304
14	0	241014544	98304
15	0	241144544	98304
16	0	266754544	122400
16	1	266624544	122280
16	2	266494544	122160
16	3	162754544	26376
17	0	266364544	122040
17	1	266234544	121920
17	2	266104544	121800
17	3	163924544	27456
18	0	265974544	121680
18	1	265844544	121560
18	2	265714544	121440
18	3	165094544	28536
19	0	265584544	121320
19	1	265454544	121200
19	2	265324544	121080
19	3	166264544	29616

Table C.6.: The 21 largest free areas in the (*BM 10:3*) allocation layout. These are also all free areas equal to or bigger than 18017 clusters in the partition.

Position [cluster]	Free size [cluster]
265866802	18017
124324183	24027
265638022	28409
124564183	30027
124804183	40027
263669912	40407
264466712	56801
265380686	60550
267499096	62144
267646484	69247
267574290	72017
267338630	81835
267252722	85731
262618164	85961
267162528	90017
267067598	94753
266967404	100017
265080492	103409
266861328	105899
125536472	107711
266748634	112517

Table C.7.: The 3 block write operations performed using the *BM 10:3* allocation layout. The operations are presented in order of execution.

Op.	Frag.	Pos. [cluster]	Size [cluster]
1	0	266748636	112512
1	1	125536472	107708
1	2	265380688	41924
2	0	267252724	85728
2	1	267338632	81832
2	2	267574292	72012
2	3	124324184	22572
3	0	267646484	69244
3	1	267499096	62140
3	2	267438388	60531
3	3	263669912	40407
3	4	124564184	29822

feature is also valid within each file operation. The feature can be seen in Figure C.1 for the Windows 7 virtual machine and in Tables C.4, C.5 and C.7 for the Windows 10 virtual machine. This pattern is valid even if there are system file or stream writing operations interleaved with the block writing operations.

C.3.2. Stream writing

The result of the stream writing file operations consists of 15 operations done in Windows 7 and 23 operations in Windows 10. Since the files are written as a stream the OS cannot implement a proper best fit allocation strategy and we therefore do not check for it.

Windows 7

There is a weak correlation between file size, the file writing order and number of fragments in the results, which can be seen in Table C.8. The correlation is however very weak and seems to be increasing for small files and decreasing for larger files.

The allocation patterns of the different file operations always begin with very small fragment sizes, often there is a single cluster allocated first. The size of the fragments increases as more data is written to disk. In some cases the fragment size is doubled in each of the first 5 consecutive allocations. This pattern diminishes as even more data is written, but the size of the fragments is constantly increasing, occasionally with a large deviation either up or down. The deviations are sometimes as frequent as every second allocation. In some cases the fragment size is suddenly increased with one or two orders of magnitude, a few times even more.

Windows 10

As for the Windows 7 case we show the number of fragments per file and their sizes in clusters (see Table C.9). Windows 10 does not show any correlation between file size, writing order and number of fragments.

The stream writing file allocation for Windows 10 is smoother than for Windows 7, with fewer and lower deviations. The general increase of the fragment's size seen in Windows 7, as well as the small fragments at the beginning of the file operations, are present also in the Windows 10 results. The first few file fragments often reach sizes of approximately 40 clusters in two or three steps, which is faster than in Windows 7. The first stream writing file operations using the *BM 10:0* allocation layout ends with one very large allocation, especially when the files are bigger. In Table C.9 this is manifested in the low number of fragments connected to some of the files containing 131072 and 262144 clusters.

Table C.8.: The number of fragments per file and their sizes in clusters for the Windows 7 stream writing experiment using *BM 7:1*. There is a weak correlation between the number of fragments, the writing order and the file size, which is most clearly seen for the largest files. The result is presented in order of size and time of writing.

No. of frag.	File size [cluster]
26	1024
35	1024
40	1024
70	32768
44	32768
1131	131328
468	131328
195	131328
242	131328
339	262144
226	262144
1657	393216
340	393216
328	393216
206	393216

Table C.9.: The number of fragments per file and their sizes in clusters for the Windows 10 stream writing experiment using *BM 10:0*. No real correlation between the number of fragments, the writing order and the file size can be seen. The result is presented in order of size and time of writing.

No. of frag.	File size [cluster]
7	1024
17	1024
27	1024
13	1024
48	1536
70	12288
18	32768
20	32768
174	49152
21	131072
39	131072
64	131072
43	131072
326	196608
165	196608
417	262144
1024	262144
103	262144
47	262144
23	262144
53	262144
48	262144
152	393216

General observations

Both Windows 7 and 10 show a general increase, although not without deviations, regarding the size of the allocated fragments. The allocation always starts with a small fragment, the maximum size of the first fragment for both versions of Windows is 5. The distribution of starting fragment sizes can be seen in Table C.10.

Table C.10.: The accumulated amount of first fragment sizes in both Windows 7 and 10 using stream writing. All Windows 7 starting fragments are single clusters.

Frag. size [cluster]	Amount [%]
1	71
2	16
3	8
4	0
5	5

All (100%) of the Windows 7 starting fragments are of size 1 and for Windows 10 the amount of size 1 starting fragments is 52%.

C.4. Discussion

The main contribution of the experiment on the behavior of the NTFS allocation algorithm in Windows 7 and 10 is the result showing a globally decreasing fragment size of block writing file operations. However, there are a few constraints that first must be fulfilled. First of all the file system must be fragmented, without any large areas of free, unallocated clusters and the files to be written have to be larger than the largest available free area to force the allocation algorithm to fragment them. Consequently there should not be any deallocations of areas larger than the next file to be written. If a new larger free area becomes available the decreasing trend will probably be restarted from there. Apparently stream writing interleaved with block writing is not affecting the decreasing size allocation pattern, but such files are not included in the decreasing pattern. Unfragmented files will also be excluded from the pattern.

The decreasing fragment size behavior can for example be used in digital forensic investigations to get a relative timestamp or the sequence of writing of a collection of block written files. By comparing the sizes of the first and second last allocated fragments of files their timestamps can be verified, or their internal age relative each

other be decided.

It is also possible to separate a block written file from a stream ditto. If that knowledge is combined with knowledge on what type of writing strategy different software packages in a Windows computer use the probable source of a file can be decided. This is for example useful in triage situations where it will be enough to scan the NTFS meta data (the MFT records) to determine the main source of a file.

Also the stream writing behavior of the allocation algorithm can be of use for a digital forensic examiner. Our result shows that the size of the first fragment of a stream written file is 1 cluster in 71% of the cases (100% in the 64 GiB virtual hard disk running Windows 7) and none of the stream written files started with a fragment larger than 5 clusters. That corresponds to files between 4 and 20 KiB in size. We have earlier found the average amount of files in a standard office computer to be approximately 350000 by counting the files in 25 (office) computers running mostly Windows 7 [6, 7]. If we combine that with the official disk space requirement of 20 GiB for a standard Windows 7 to 10 installation [17–19] we get an average file size of 60 KiB. This is a theoretical lower bound, because we did not take the size of the user files included in the computer average of 350000 files into account. Hence any allocated area smaller than 60 KiB belongs to the first part of a stream written file with a high probability. This knowledge can for example be used in file carving processes to quickly find the data type of files directly from the data, without the need to trust the file name, by searching for small areas with similar data content and then extracting any magical bytes from them.

Since we have reverse engineered the allocation algorithm without access to any documentation we have no written proofs of the reasons behind the behavior we have found. We therefore can only hypothesize, but a valid reason for the block writing behavior of creating similarly sized fragments is to create an even distribution of the data over several cylinders or flash memory capsules for faster access and wear levelling.

The reason behind the stream writing behavior of starting small and increasing the fragment sizes might be two-fold. By starting from very small fragment sizes any free clusters left over when doing best fit allocation can be taken care of. The other reason is the fact that without knowledge on the actual size of the file to be written the algorithm has to estimate the final size of the file. By increasing the size of the fragments as more data is written we get a acceptable compromise between speed (less fragmentation) and a good utilization factor (small areas are not wasted). Hence the average fragment size will increase as the file size increases.

The file system allocation algorithm of Windows 7 and 10 when using NTFS should be best fit [5], but our experiment shows that it is not completely true. The block writing allocation is deviating from a strict best fit behavior in a few cases. Possible reasons for any differences in behavior are typically OS version, partition size, degree of file system utilization, file size and type of file writing behavior.

Regarding the influence of the OS we have not noticed any differences in allocation behavior. The result might have been influenced by the fact that we only used two versions of Windows, but by using both the first and the latest versions of the latest generation of Windows we cover for any differences introduced in Windows 8 and 8.1.

Any differences in allocation strategy depending on the partition size is covered by the use of both a 64 GiB and a 1 TiB virtual hard disk. During the experiments we noticed small differences in behavior between the small and the large hard disks, especially in the stream writing case where the deviations from the increasing fragment size were smaller for the 1 TiB hard disk than for the 64 GiB hard disk. The most probable reason for this behavior is the larger amount of varied sizes of free areas to choose from in the larger disk.

We found that the allocation algorithm does not use a best fit strategy when allocating block written files in Windows 10 when using the unmodified *BM 10:0* allocation layout. Instead it allocates chunks of the large area of free clusters at the end of the file system, which seems reasonable from a wear levelling point of view. We did not observe the same behavior in the Windows 7 case, but there we had gotten rid of all large unallocated areas before the experiment started.

The allocation behavior during the 4 MiB file operations in Windows 10 (using the *BM 10:0* layout) where the algorithm allocates parts of one of the large free areas at the end of the partition cannot be explained by the fact that the block writing operations and the stream operations were interleaved during the experiment. Nor can it be explained by any system files being written to the free areas that would have been appropriate to use. After file operation 5 in Table C.4 there still are several free areas of a few thousand clusters left that could have been used instead. The same applies to the file write operation 5 shown in Table C.5. This behavior more resembles a worst fit allocation strategy than a best fit ditto.

The knowledge on the allocation behavior of Windows and NTFS gained through the experiments presented in this paper will benefit our previous work within the file carving area [20–25]. There we experimented with different algorithms to detect the file type of data fragments using only the information held in the fragments themselves. Using for example the fact that the allocation algorithm has different behavior for block writing and stream writing can help identify and separate data types that are written in different ways. Also the fact that the free areas often are not fully utilized can help improve data type separation in heavily fragmented cases by explaining very small areas with different properties intertwined between larger areas of data with equal properties. Especially since it is well known that many file types contain areas of different types of data [20, 26].

C.5. Conclusion and future work

By writing files ranging in size between 4 MiB to 1,5 GiB to a Windows 7 virtual machine having a 64 GiB hard drive, as well as a Windows 10 virtual machine having a 1 TiB hard drive, we have found that the Windows NTFS allocation algorithm is more complex than the best fit strategy described in the literature [5]. In most of the file operations executed during our experiment the algorithm behaved as a strict best fit type, but when having access to a very large area of free clusters it started to allocate parts of that area instead of using options corresponding to a best fit allocation strategy. Looking at data from previous experiments [6, 7] we have found that having a few very large areas of free clusters at the end of a NTFS partition is the standard situation, thus the allocation strategy used by Windows together with NTFS is only best fit in special circumstances. Likewise the allocation strategy is not strictly best fit when dealing with stream written files, where the allocation algorithm is creating increasingly larger fragments as more data is written (the fragment size and the currently written size correlates). However, the fragments are allocated to the best fitting free areas, so there is a foundation of best fit behavior in the algorithm.

We have not found any literature empirically studying the inner workings of the allocation strategy used in Windows 7 and 10 partitions formatted as NTFS. Therefore already the existence of this work contributes to the digital forensics field. The result can furthermore be used to verify timestamps, rebuild files in file carving and determine the type of file on a high level by looking at how the sizes of the allocated areas increases or decreases. Block written files are allocated in decreasing order of fragment size and stream written files are allocated in increasing order of fragment size. We also found that very small allocated areas (1 to 5 clusters) belong to the start of stream written files with high probability.

As future work we will expand the experiment to be able to isolate the different parameters affecting the behavior of the allocation algorithm. We also need more data to further strengthen our results and conclusions. It would also be of great interest to include other OSs and file systems than Windows and NTFS in the experiment.

C.6. Bibliography

- [1] A. Pal and N. Memon. “The evolution of file carving.” In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [2] R. Poisel and S. Tjoa. “A Comprehensive Literature Review of File Carving.” In: *2013 International Conference on Availability, Reliability and Security*. Sept. 2013, pp. 475–484. DOI: 10.1109/ARES.2013.62.

- [3] S. Willassen. “Finding Evidence of Antedating in Digital Investigations.” In: *2008 Third International Conference on Availability, Reliability and Security*. Mar. 2008, pp. 26–32. DOI: 10.1109/ARES.2008.149.
- [4] S. Willassen. “Methods for Enhancement of Timestamp Evidence in Digital Investigations.” PhD thesis. Norwegian University of Science, Technology, Faculty of Information Technology, Mathematics, and Electrical Engineering, Department of Telematics, Jan. 2008.
- [5] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005. ISBN: 0321268172.
- [6] M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Using NTFS Cluster Allocation Behavior to Find the Location of User Data.” In: *Digital Investigation 29* (2019), S51–S60. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.018.
- [7] M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8.
- [8] A. Silberschatz, P. Galvin, and G. Gagne. *Operating System Concepts*. 9th ed. Wiley, Dec. 2012.
- [9] W. Stallings. *Operating Systems — Internals and Design Principles*. 7th. Upper Saddle River, New Jersey, USA: Pearson Education Inc., 2012.
- [10] A. Tanenbaum and H. Bos. *Modern Operating Systems*. 4th. Upper Saddle River, New Jersey, USA: Pearson Education Inc., 2015.
- [11] J. Hughes. *The Four Stages of NTFS File Growth*. Last accessed 24-10-2018. Oct. 2009. URL: <https://blogs.technet.microsoft.com/askcore/2009/10/16/the-four-stages-of-ntfs-file-growth/>.
- [12] Microsoft. *How NTFS Works*. Last accessed 30-09-2018. 2018. URL: [https://technet.microsoft.com/pt-pt/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx).
- [13] maaartinus, L. Osterman, and PC Guru. *What block allocation algorithm does NTFS use?* Last accessed 24-01-2019. Mar. 2017. URL: <https://superuser.com/questions/274855/what-block-allocation-algorithm-does-ntfs-use>.
- [14] W. Tse. “Forensic analysis using FAT32 file cluster allocation patterns.” MA thesis. University of Hong Kong, 2011.

C. Disk Cluster Allocation Behavior in Windows and NTFS

- [15] W. Minnaard. “The Linux FAT32 allocator and file creation order reconstruction.” In: *Digital Investigation* 11.3 (2014). Special Issue: Embedded Forensics, pp. 224–233. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.06.008.
- [16] B. Carrier. *TSK Tool Overview*. 2014. URL: http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.
- [17] Microsoft. *System requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-gb/help/12660/windows-8-system-requirements>.
- [18] Microsoft. *Windows 10 system requirements*. Last accessed 30-04-2018. Nov. 2017. URL: <https://support.microsoft.com/en-us/help/4028142/windows-windows-10-system-requirements>.
- [19] Microsoft. *Windows 7 system requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-us/help/10737/windows-7-system-requirements>.
- [20] M. Karresand. “Completing the Picture — Fragments and Back Again.” Licentiate thesis. Linköping Institute of Technology, Linköping University, Sweden, May 2008.
- [21] M. Karresand and N. Shahmehri. “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*. Piscataway, NJ, USA: IEEE, 2006, pp. 140–147. DOI: 10.1109/IAW.2006.1652088.
- [22] M. Karresand and N. Shahmehri. “Oscar — File Type and Camera Identification Using the Structure of Binary Data Fragments.” In: *Proceedings of the 1st Conference on Advances in Computer Security and Forensics, ACSF*. Ed. by J. Haggerty and M. Merabti. Liverpool, UK: The School of Computing and Mathematical Sciences, John Moores University, July 2006, pp. 11–20.
- [23] M. Karresand and N. Shahmehri. “Oscar — File Type Identification of Binary Data in Disk Clusters and RAM Pages.” In: *Security and Privacy in Dynamic Environments, Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22-24 May 2006, Karlstad, Sweden*. Vol. 201. Lecture Notes in Computer Science. Springer, 2006, pp. 413–424. DOI: 10.1007/0-387-33406-8_35.

- [24] M. Karresand and N. Shahmehri. “Oscar — Using Byte Pairs to Find File Type and Camera Make of Data Fragments.” In: *Proceedings of the 2nd European Conference on Computer Network Defence, in conjunction with the First Workshop on Digital Forensics and Incident Analysis (EC2ND 2006)*. Ed. by A. Blyth and I. Sutherland. Springer Verlag, 2007, pp. 85–94. DOI: 10.1007/978-1-84628-750-3_9.
- [25] M. Karresand and N. Shahmehri. “Reassembly of fragmented JPEG images containing restart markers.” In: *Proceedings - 4th Annual European Conference on Computer Network Defense, EC2ND 2008*. 2008, pp. 25–32. DOI: 10.1109/EC2ND.2008.10.
- [26] V. Roussev and S. Garfinkel. “File Fragment Classification-The Case for Specialized Approaches.” In: *2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*. May 2009, pp. 3–14. DOI: 10.1109/SADFE.2009.21.

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

The layout of the article has been lightly edited to fit the overall layout of the thesis. This text and a full citation of the article has been added below the title. Since the work is focused on disk partitions (Windows volumes) the term Logical Block Addressing (LBA) in the article has been corrected to LPVA where applicable. The content is otherwise unchanged and corresponds to the original, published, version.

M. Karresand, G. Dyrkolbotn, and S. Axelsson. “An Empirical Study of the NTFS Cluster Allocation Behavior Over Time.” In: *Forensic Science International: Digital Investigation* 33 Supplement (July 2020), p. 301008. ISSN: 2666-2817. DOI: [10.1016/j.fsidi.2020.301008](https://doi.org/10.1016/j.fsidi.2020.301008)

Abstract

The amount of data to be handled in digital forensic investigations is continuously increasing, while the tools and processes used are not developed accordingly. This especially affects the digital forensic sub-field of file carving. The use of the structuring of stored data induced by the allocation algorithm to increase the efficiency of the forensic process has been independently suggested by Casey and us. Building on that idea we have set up an experiment to study the allocation algorithm of New Technology File System (NTFS) and its behavior over time from different points of view. This includes if the allocation algorithm behaves the same regardless of Windows version or size of the hard drive, its adherence to the best fit allocation strategy and the distribution of the allocation activity over the available (logical) storage space. Our results show that space is not a factor, but there are differences in the allocation behavior between Windows 7 and Windows 10. The results also show that the allocation strategy favors filling in holes in the already written area instead of claiming the unused space at the end of a partition and that the area with the highest allocation activity is slowly progressing from approximately 10 GiB into a partition towards the end as the disk is filling up.

D.1. Introduction

The amount of data to be handled during digital forensic case work is rapidly increasing and is a major challenge to the digital forensic community [1]. The problem has been of concern to the digital forensic field for many years [2–6], but the problem has not yet been solved.

Casey [7] and also Karresand et al. [8–10] have independently suggested to use the inherent structures in the stored data to improve the digital forensic process. The principle builds on taking advantage of the pattern introduced by the allocation algorithm and in that way improve for example the efficiency when rebuilding files, extracting temporal information (time stamps) from raw data and direct searches to the areas most likely to contain important (user related) data.

The principle is especially valid for the digital forensic sub-field of *file carving*, which is used in digital forensic investigations when there is no file system available in the investigated media. The file carving process is based on using only the properties of the stored data itself [11, 12]. File carving is highly valuable to the digital forensic investigator, but computationally intensive to perform, hence much effort is put into mitigating the increasing amounts of data by different means. In a survey by Quick and Choo [1] the following concepts to decrease the amount of work needed to be done are listed; data mining, data reduction and subsets, triage, intelligence analysis and digital intelligence, distributed and parallel processing, visualization, digital forensics as a service (DFaaS) and different artificial intelligence techniques.

The foundation of digital forensics is to use the inherent structures of data, but the idea to use the inherent structures introduced by the storage process in stored data is rather new and not yet fully investigated. Therefore the actual behavior of the allocation algorithm has to be found for any relevant file system. The behavior of file systems from the open source field can be found by studying their code base, but for closed source operating systems (OSs) the behavior is best found by empirical studies of the allocation behavior in experiments and the real world. The currently most popular OS is Windows, which has almost 86% of the market share [13]. Windows uses New Technology File System (NTFS) as the default file system and we have therefore chosen to study the allocation behavior of NTFS in recent versions of Windows.

The aim of the project is to gain knowledge on the allocation behavior of NTFS in different modern versions of Windows (versions 7 to 10) and especially if and how the behavior changes over time. This includes the adherence to the *best fit* [14] allocation strategy and if the allocation activity is evenly spread over the (logical) addresses of the storage area. We will also test whether the allocation algorithm of Windows and NTFS is version and/or size dependent.

To be able to answer the research questions we have executed an experiment where we used a weighted random distribution to write, expand, shrink and delete files in

four different versions of Microsoft Windows (7, 8, 8.1 and 10). To be able to find similarities and differences between the Windows versions we ran eight instances of the same virtual machine for each Windows version. Four of the machines for each Windows version used the same file operation pattern, also referred to as the *standard pattern*, and the other four machines ran unique patterns. The virtual machines used 64 GiB disk, but we also set up three extra virtual machines with 256 GiB disks to detect any differences in allocation behavior due to larger disk sizes. These virtual machines used a modified version of the the standard pattern, where each file operation was increased by a factor of 4.6 to compensate for the larger disk size. Every operation was followed by the extraction of the current cluster allocation status taken from the \$Bitmap file in the Master File Table (MFT). The allocation status was then used to find the difference in cluster allocation between each operation. The experiment was set to run 10000 iterations in each virtual machine.

The rest of this paper is organized as follows: The remaining parts of Section Introduction presents related work and our contributions. In Section Experiment we describe the experimental platform and how the experiment was implemented. Section Result presents the result of the study. In Section Discussion we discuss the effects and implications of our result to the research field of digital forensics and especially file carving. Section Conclusion and future work concludes the work and presents ideas of future work to be done.

D.1.1. Background

[15] describes the theory of file system construction. A file system keeps track of data stored on secondary storage and is organized in different ways. However, all implementations 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.

Most modern file systems use an index allocation strategy to keep track of the data on disk. The index allocation strategy separates the metadata and the file data and hence the index itself does not suffer from external fragmentation (free holes being too small to be filled with new data), but the data part can if heavily used give rise to fragmented files, requiring regular defragmentation of the file system. There is also a risk of disk space being wasted when using index allocation, 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 in the data part that is to be populated by new files. The *best fit* algorithm is meant to reduce the risk of file data fragmentation by always utilizing the free space best fitting the file to be written. The idea is to reduce the free space remaining in a block of free clusters due to large differences in what is needed and what is available. However, this strategy

requires all the free spaces available to be compared at each file operation before the best fit can be chosen.

NTFS (Microsoft Windows) is using an index allocation strategy [16, 17], the index is called the MFT [16] and the individual posts are called MFT records. The problem of space being wasted when using index allocation is solved by storing the data of smaller files (up to approximately 700 B¹) in the MFT records themselves. To allocate space for file data NTFS uses a best fit allocation strategy [14].

When formatting an NTFS partition 12.5% of the space is reserved for the MFT as default [16]. The MFT records 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 Logical Partition Volume Address (LPVA) order. If a cluster is allocated the corresponding bit in the \$Bitmap file is set to 1, hence 0 represents a free cluster.

A file can either be written as a stream or as one large block at once [8]. In the first case the OS does not know the final size of the file and therefore cannot optimize the allocation accordingly. This often leads to file fragmentation, but the behavior is partly mitigated by the internal buffering of the OS. Writing a file in one piece gives the OS information on its size and it can therefore optimize the storage by using its standard allocation algorithm. This behavior is probably more common when dealing with smaller files that easily can be held in Random Access Memory (RAM), than for large files. The specific write behavior is software dependent and might incorporate temporary writing of files to protect the data in case of a power loss or hardware failure.

D.1.2. Related work

Since the research area is new there is not much related work to be found. We have therefore also included work from the file carving area containing some material related to the allocation algorithm of file systems, as well as work related to the placement of data on disk.

We have presented the novel idea of creating a map of the probability of finding user data at different LPVA positions of hard disk partitions in three earlier articles [8–10]. In the first article [10] we tried to find any static areas of NTFS partitions. We defined static areas as areas containing the same data at the same logical position in different partitions. During that work we also found that the \$MFT started exactly 3 GiB into the NTFS formatted partitions of the over 30 unrelated real world hard drives we looked at. The hard drives had different sizes and contained different versions of

¹The 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 to 700 bytes. Microsoft reports a 900 byte limit [16].

Windows [10]. We also used the \$Bitmap file to study which part of an NTFS partition had the highest allocation activity [9]. The highest activity was found approximately 10 GiB into a partition. The activity then slowly decreased towards the end of the partition, which was expected. In the third article [8] we presented new information on the detailed behavior of the allocation algorithm in Windows using NTFS in different file writing situations (writing a file like one block or writing it as a stream of data). The results showed that the size of the allocated blocks (i.e. file fragments) decreased during block writing and increased during stream writing. The study was however based on a low number of writing operations, Windows versions and partition sizes. We will therefore validate our results in a larger experiment.

Casey [7] introduces a new digital forensic research field called digital stratigraphy that draws inspiration from archaeology, which share many common features with (digital) forensics. The idea is to look upon file system activities as layers (strata) that are structured. This structure can be used to complement, improve and expand the information currently retrieved from hard drives and disk images. In the article [7] shows how the next fit allocation strategy used in File Allocation Table (FAT) file system (for example FAT16 and FAT32 in Windows) clearly indicates the order of storage of file data. He also touches upon the best fit allocation algorithm used in NTFS and how it behaves, but the bulk of the NTFS part is focused on specific behavior regarding valid data length (VDL) slack that is currently not properly covered by available digital forensic tools. He also covers the effect file tunneling has on the reliability of file system meta data.

Key [18] has developed an EnScript module to the EnCase software which creates a map of the recoverable sectors of a file found in a file system. It can handle situations where other tools do not work, for example partially damaged files. It is very processor intensive and therefore can only create maps of a few files at a time. Key does not mention to what extent any knowledge on the allocation pattern of the OS and file system is used in the article.

Gladyshev and James [2] have studied 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 outperform standard linear carving algorithms, but their solution was not generally applicable at the time of writing. Gladyshev and James [2] mention using the distribution of data on disk, but do not explain if they take advantage of any structures introduced by the allocation algorithm.

In two articles by Baar et al. [19] and Beek et al. [20] outlining the DFaaS system Hansken [20] and its predecessor Xiraf [19] the concept of non-linear extraction of data from images is discussed. Both van Baar and van Beek suggest that the MFT records 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. As we understand they do not base the priority on the allocation pattern of the analyzed system, but on file name and other higher level meta data found in the MFT records.

Jones et al. [21] 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. This work connects to our experiments, because free areas are meant to be reused by the file system, but depending on the size of the free area the best fit allocation algorithm might not be able to use it for a certain amount of time. The concept of data persistence is of interest to us because the persistence at different areas of storage media indicates that these positions are not reused. This information might correlate with the allocation pattern and its development over time, which is what we are studying.

Fairbanks and Garfinkel [22] present 12 factors affecting data persistence in storage media. Fairbanks [23] and Fairbanks [24] also has described the low-level functions of fourth extended filesystem (EXT4) and their effect on digital forensics. Although the articles do not describe the inner workings of NTFS the principle is still of interest to us, especially in future extensions of our experiment.

Our main contribution to the digital forensic research field is the result showing that there are differences in the allocation behavior of Windows 10 and older versions of Windows and that the best fit allocation strategy is not fully used. The maximum and median fragment sizes of Windows 7 show interesting linear properties, which are not found in Windows 10. There are also areas within the file system that are rarely used, forming bands of low allocation activity through the file systems. We also show how the allocation of new data is concentrated to an area close to (just before) the middle of a partition, but also how that area is slowly moving towards the middle of the partition, regardless of the size of the partition. The result can be used to determine the sequential order of files, estimate the proper size of file fragments to be carved and where in the create and erase cycle a file system is through the leeward effect found in all Windows versions. The results can also be used to improve the efficiency of the file carving process by helping the digital forensic investigator to prioritize where to start searching for user related data.

D.2. Experiment

The experiment was based on iteratively creating, deleting, expanding and shrinking files in unused NTFS formatted partitions in 32 virtual machines running Windows versions 7, 8, 8.1 and 10. The aim was to empirically study how the cluster allocation pattern develops over time and how the allocation frequency varied at different LPVA

positions. Each file operation iteration contained the following elements:

1. Boot the virtual machine.
2. Based on a precomputed list 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)

Since each file operation iteration required the virtual machine to be rebooted a full iteration took several minutes to complete. There were also extra time slots inserted at critical moments to compensate for any variations in execution time during an iteration.

We allowed the experiment to run for 16 days before shutting it down due to time constraints. Most, but not all, of the virtual machines then had completed 10,000 iterations. All virtual machines were run in parallel in a computer cluster to save time, if we had had to run them one-by-one the experiment would have taken up to $35 * 16 = 560$ days to execute, excluding setup time.

The foundation of the experiment was built on having four virtual machines installed with one Windows version each using standard parameters. Then a Python 2.7 execution environment was installed together with the file operation scripts. An auto-started `.bat` script was placed in the virtual machine to check when the boot sequence was finished. The script was small enough to fit into an MFT record and hence did not require any new cluster allocation outside the MFT. The path setting was modified and the security level of Windows was lowered to allow logging in without a password. The goal was 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 were however a number of system processes that also modified the file system in each iteration, which were beyond our control.

We let 16 virtual machines (four machines for each version of Windows, half of the total amount of machines) use exactly the same file operation pattern (the standard pattern) to test if there was any deterministic behavior connected to the allocation (if any similarities of the allocation patterns could be found). Hypothetically it should be, since the virtual machines within each Windows version were exact copies of each other.

To be able to see any deterministic allocation behavior we used `scp` when distributing the virtual machines to the computer cluster nodes. We did not use the VirtualBox clone function because it make small changes to the virtual machine's settings, which in turn might affect the OS and hence trigger an unintended write operation. Using `scp` to copy the virtual machines guaranteed them to be identical, which was verified using Secure Hash Algorithm 1 (SHA-1) hash summing. However, we did not have full

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

control of the cluster allocation and deallocation during an iteration of the experiment because of internal OS processes, thus we still had uncontrolled variables affecting the outcome of the standard pattern sub experiment.

Due to instability in the VBoxManage interface and unforeseen popup windows appearing in the virtual machines several of them had to be restarted during the course of the experiment. This might have affected the result of the experiment, but since we used at least four virtual machines for each combination of Windows version and partition size in the experiment the effects of the unplanned reboots were diminished.

The experiment was run on eleven nodes in a large computer cluster. The cluster is managed by Swedish Defence Research Agency (FOI) and we therefore were not allowed to make any changes to the cluster nodes' OS or configuration, which forced us to use alternative tools to extract the data. This did however not affect the experimental results.

Each cluster node ran four virtual machines, one for each version of Windows in our test (see Table D.1). The \$Bitmap file from the MFT of NTFS was used to check which clusters were affected by each file operation. To enable us to extract the \$Bitmap file after each operation the virtual machines were configured to use fixed size disks, which can be directly handled by common Linux tools. We limited the size of the fixed virtual disks to 64 GiB to be able to use four virtual machines in each node and still have space for the \$Bitmap file copies. Each copy was 2 MiB large and there would be 40,000 \$Bitmap copies (over 78 GiB) in each cluster node when the experiment was finished. If we had used larger virtual disks we would have had to decrease the number of virtual machines, which in turn would have affected the reliability of the results. The hard drive size of 64 GiB was therefore found to be a reasonable trade-off between reliability and a realistic hard drive size.

Table D.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 Enterprise	1703

The virtual machines used four internal Python scripts for the experiment, one for each type of file operation. The scripts and the resulting \$Bitmap files were placed in an external folder shared with the host, one for each machine, to isolate the machines from each other. This also meant that we avoided cluttering the virtual disk with data and

therefore minimized the risk of unspecified behavior due to several machines accessing the same file at the same time. The file operations were executed as the local user of the virtual machines to simulate the activity of a real user.

The execution of the experiment was controlled by a Python script on the host node. The script selected one of four actions; *create*, *delete*, *increase* and *decrease* based on a configuration file containing a precomputed weighted random selection. The selection was biased towards file creation and extension, where $\frac{1}{4}$ of the operations were set to create, $\frac{9}{40}$ to erase, $\frac{11}{40}$ to increase and $\frac{1}{4}$ to decrease. The process was set to create files until the disk was 30% full and then switch back to either erase- or create-only mode if the usage of the disk reached above 95% or below 5%. The communication between the host script and the virtual machine scripts was done using the VBoxManage interface.

The experiment emulated a file sharing or multimedia consuming user that alternated his or her file operations between small and large files. The size of the small files varied between 4 KiB and 4 MiB and the size of the large files between 1 MiB and 1 GiB. The size of the large files might not seem very large, but since the virtual disks were only 64 GiB in size a 1 GiB file corresponds to a 32 GiB file on a 2 TiB disk.

All write operations were stream writing operations, i. e. the OS of the virtual machine did not know the size of the file to be written in advance, which was meant to represent for example a file being downloaded from the internet. However, stream writing operations might give a more fragmented allocation result than block writing operations [8].

The script responsible for managing the write operations in the virtual machines on a host node checked if the currently active virtual machine was started before it sent the file operation command. There was also a check of the completion of a file operation, as well as a check of the exit status of the virtual machine when it was being shut down. The exit status of each file operation was also checked and if it indicated an error the transaction counter was decremented and the same operation was retried. Every transaction was logged in a file indicating the sequence number, the action performed, the name of the affected file and its current size.

The three Python scripts that executed write operations on the virtual machines were set to write the iteration sequence number and a individual sequence number into every 512 byte sector of the file to be written. This enabled us to see the raw write pattern in the virtual disk file if ever needed. The iteration sequence number was also given as file name to further increase the traceability. The *create* and *decrease* file scripts both wrote new files (using the *wb* flag in the Python open command). The *increase* script appended new data at the end to an existing file, using the *ab* flag. Therefore increased files could contain multiple number sequences.

To avoid unnecessarily burdening the virtual machines the four file operation scripts run by the virtual machines were kept as simple as possible and most of the control functions (for example the status check of the virtual machine and file operation) were

executed by the main script on the host node. The randomization of the file operations was done beforehand and held in a configuration file used to control the script on the host. In that way the same file operations could be executed on several machines in parallel. This also avoided the problem of having to individually seed several random functions, now the seeding was centralized.

Since we were not allowed to install any software on the host nodes in the computer cluster we chose to use the `dd` tool to extract the \$Bitmap file in each iteration. That required us to know the exact location of the \$Bitmap file in advance, which we solved by using fixed size virtual disks. The size and location of the \$Bitmap file would not change since the size of the disk was static. A better solution might be to use the `icat` tool from the Sleuth Kit by Carrier [25]. That will be fixed in the next version of our experimental platform.

When the experiment was finished we extracted the LPVA position of the affected clusters in each file operation from the \$Bitmap copies. The extraction process gave us information on all clusters that had been allocated or freed during each file operation. Since we could not control the behavior of the OS any allocation changes induced by the OS were also included.

To test if there were any differences in allocation behavior connected to the size of the hard drive we extended the experiment with three virtual machines having 256 GiB hard drives. The machines were installed with Windows 7, 8.1 and 10 in the same way as the 64 GiB machines. The standard pattern was used to enable comparison to the 64 GiB machines using the same pattern. Due to the 256 GiB machines being started later than the 64 GiB machines the Windows 10 machine only executed 8,331 iterations before being stopped. We therefore have limited the result to the first 8,331 iterations in all machines.

D.3. Result

To increase the readability of the paper we have chosen to only show graphs for Windows 7 and 10. The differences between the graphs for Windows 7, 8 and 8.1 are often small and we therefore let Windows 7 represent all three Windows versions below Windows 10. We can of course use data from all three versions in the same graph, but that will decrease the visibility of the specific features we want to show, because the data are not equal, only similar.

Please observe that the figures are showing the statistical properties of the allocation patterns, not the actual allocations for each file operation. Showing the actual allocations would require us to plot up to hundreds of thousands of data points for each file operation, which obviously is not feasible in this publication format. We also use different scales (log and linear) on the Y-axis of the plots to increase the visibility. The

maximum allocatable position of a 64 GiB partition is almost 17,000,000 clusters and approximately 67,000,000 clusters for 256 GiB partitions. Since some of the figures use different units the maximum value of their Y-axes might differ.

In many graphs there seems to be a disturbance visible as an area of low activity centered around file operation 5350. This effect comes from the rapid decrease in file system utilization that can be seen in for example Figures D.1 and D.2, where the thick black curve at the top of each graph shows the degree of utilization. Please observe that the utilization curve has been moved upwards with approximately 2,500,000 clusters to increase visibility.

As can be seen in Figure D.1 the mean position of the newly allocated clusters for each file operation in the 64 GiB virtual main partitions correlates with the amount of allocated clusters in the file system, i. e. the degree of utilization of the file system. The mean allocation position patterns are similar for all four Windows versions, but not equal. Each partition in the experiment adds a few unique outliers to the graph. The file system utilization plot, derived from the standard pattern file operations configuration file, added on top of the mean position graph has been raised by 2,526,780 clusters (approximately 9.6 GiB) to increase its visibility. The value represents the difference between the maximum value of the standard pattern file operations configuration file and the maximum allocated cluster position of one of the Windows 7 machines using that file.

The correlation between the mean position and the utilization of the file system shown in Figure D.1 also appear in the main partition of the 256 GiB virtual disks, regardless of the installed OS. This is shown in Figure D.2. The included plot of the file system utilization is multiplied with 4 to compensate for the larger hard disk size and also to increase the visibility. As can be seen the bulk of the mean allocation positions in the 256 GiB disks, as well as the highest mean allocation values, correlate well with the mean allocation positions in the 64 GiB disks.

The maximum allocation position is an indication of how the OS utilizes the free area at the end of the file system. This is shown in Figure D.3. As can be seen the highest allocated position for each file operation also increases as the number of operations increases. The increase is divided into steps, which are correlated to increases in the utilization of the file system. There are however no corresponding rapid decreases in the maximum positions when the utilization decreases. Instead the current level is only slowly decreasing until the utilization gets a new maximum value. In the plots the effect looks like the formation of clouds on the leeward of a mountain range. Although the effect is strictly visual and has nothing to do with how physical clouds are formed, we will be referring to the effect as the “leeward effect” in the rest of the article.

In Figure D.4 the maximum allocation position for Windows 10 is shown. The leeward effect is less distinct here, instead the maximum allocation positions remain at the same level until the next increase in the file system utilization, making the plot look

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

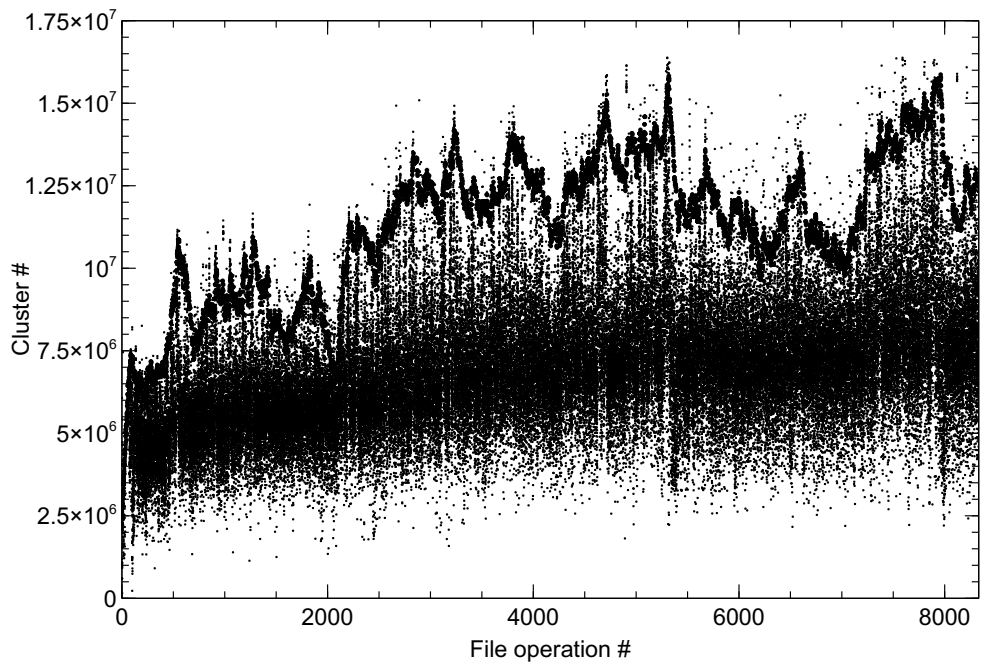


Figure D.1.: The mean allocation position for all included Windows versions in 64 GiB hard drives using the standard file operation pattern. We have also included a plot of the file system utilization, which corresponds to the black line at the top of the graph. The file system utilization plot is raised by 2,526,780 clusters to increase the visibility.

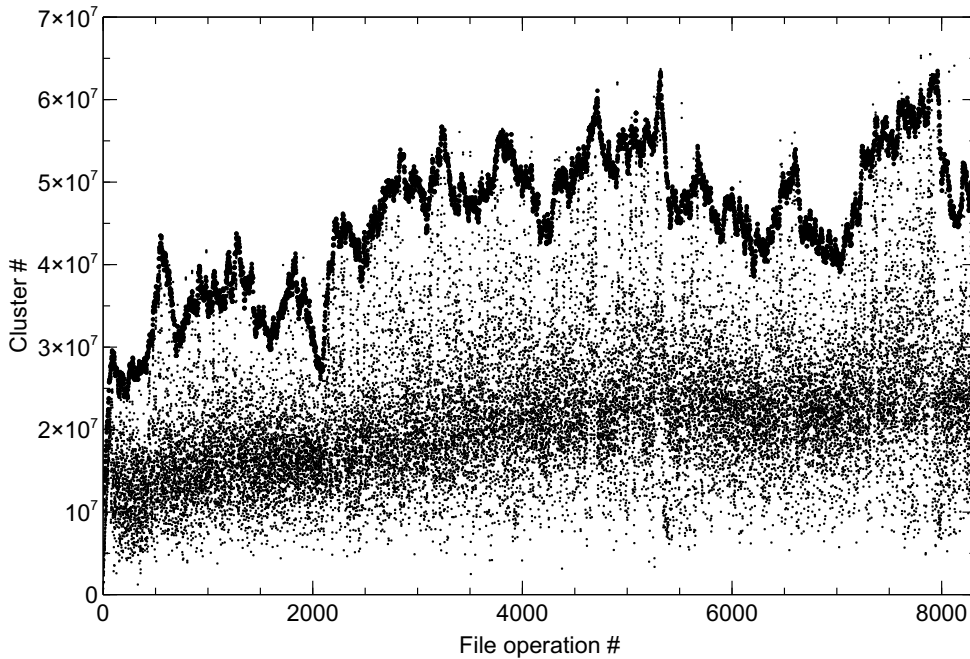


Figure D.2.: The mean allocation position of the Windows 7, 8.1 and 10 having 256 GiB hard drives. We have also included a plot of the file system utilization (multiplied by 4 to fit the larger disk size) to enable comparison with the corresponding 64 GiB hard drives. Please observe that the maximum allocatable position in a 256 GiB partition is approximately 67,000,000 clusters and that the scale of the Y-axis therefore differs from the corresponding plots of the 64 GiB partitions.

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

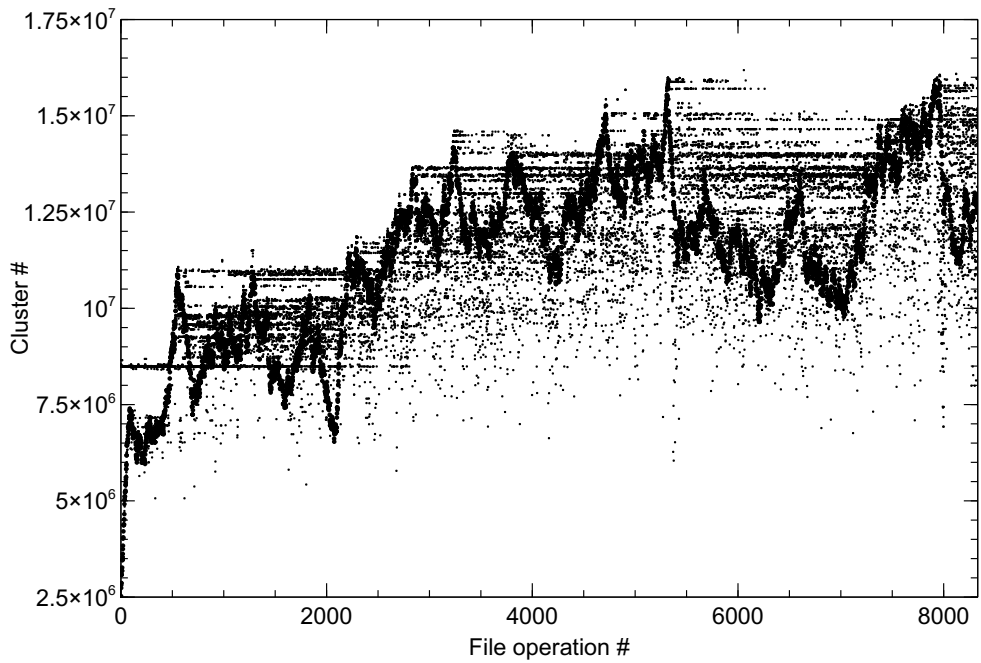


Figure D.3.: The maximum allocation position for Windows 7 in 64 GiB hard drives using the standard file operation pattern. We have added the file system utilization curve (the black line at the top) to the graph to increase the visibility of the leeward effect of the allocations.

more like heavy fog than leeward clouds. The larger amount of allocations at high cluster addresses is also manifested by the lower amount of allocations at positions below the file system utilization curve.

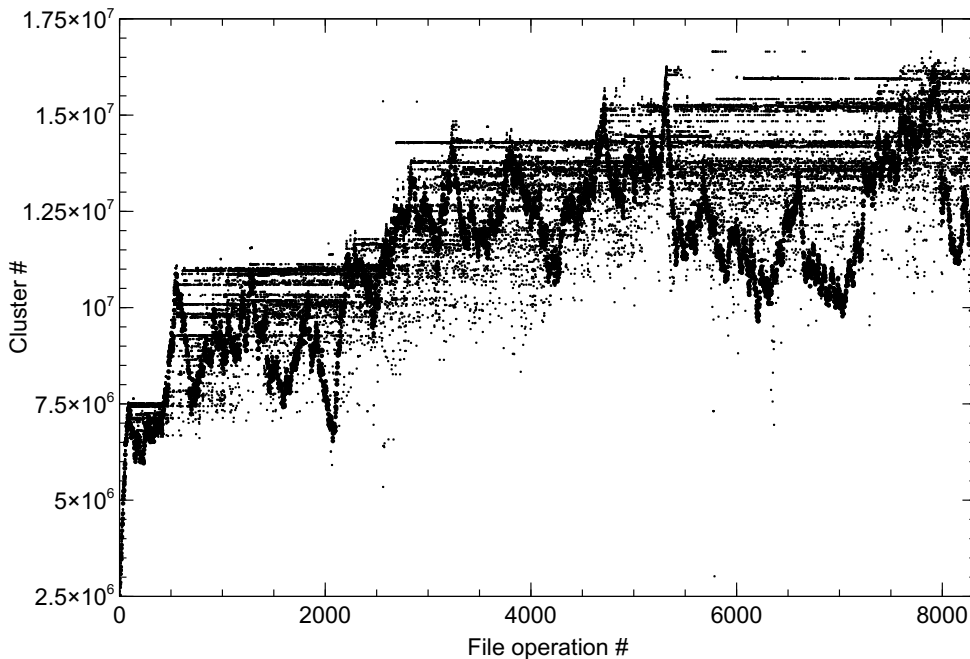


Figure D.4.: The maximum allocation position for Windows 10 in 64 GiB hard drives using the standard file operation pattern. We have added the file system utilization curve (the black line at the top) to the graph to increase the visibility of the leeward effect of the allocations.

The median allocation position graph of Windows 10 lacks a feature that the graph of the older Windows versions show (see Figure D.5). Windows 7, 8 and 8.1 all have an approximately 100,000 clusters wide unused area in the middle of their partitions centered around cluster 8,600,000 for Windows 7 and 8,400,000 for Windows 8 and 8.1. The area is more or less visible for all three Windows versions, but in Windows 7 it is visible from the start of the file operations (see Figures D.3 and D.5) from a significant lower bound of the unused area, which is not the case for Windows 8 and 8.1.

The graph (see Figure D.6) of the statistical mode, here defined as the middle position of the largest consecutive group of clusters allocated in a file operation, of Windows 7 also shows an unused area in the middle of the partition, which can also be seen in Figure D.5. The allocated positions in the mode graph are however almost evenly distributed in the allocated area and also show a sharp border to the sparsely al-

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

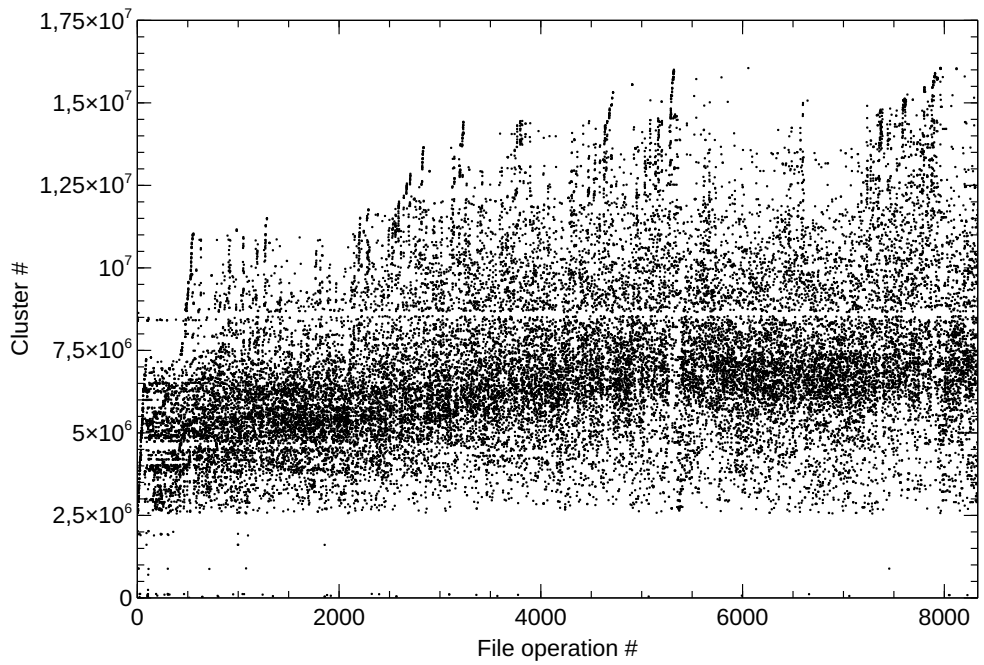


Figure D.5.: The median allocation position for Windows 7 in 64 GiB hard drives using the standard file operation pattern. Please observe the horizontal sparse part in the middle, which is missing in Windows 10.

located area between cluster position 125,000 and 2,550,000. This border is less sharp in Figure D.5, but that might be an effect of the median being a calculated value in difference to the mode being a factual value. Hence the mode value is closer to the actual behavior of the file allocation algorithm. As for the median allocation position graph in Figure D.5 the unused area in the middle of the Windows 8.1 partitions start to vanish around file operation 9,000 and is not present in Windows 10.

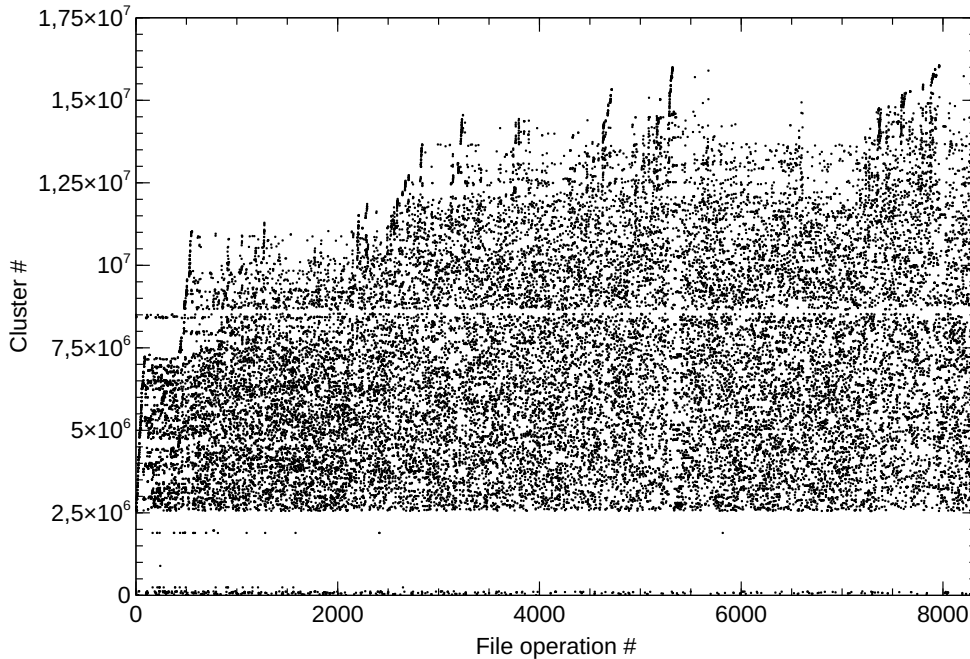


Figure D.6.: The mode allocation position for Windows 7 in 64 GiB hard drives using the standard file operation pattern. Please observe the thin horizontal sparse area in the middle of the plot, the same sparse area can be seen in Figure D.5

The Windows 10 mode graph in Figure D.7 is similar, but not equal, to the Windows 7 graph in Figure D.6. Both graphs show data from the virtual machines using the standard file operation pattern. The unused area close to the middle of the partition is lacking in Figure D.7 and there is more allocation activity at the first part of the partition. The Windows 10 graph also shows how the highest allocated positions are reused after their initial allocation to a higher degree than in Windows 7 (see Figure D.6). This is manifested by the higher amount of leeward effect in Figure D.7 (the peaks are not as visible in the Windows 10 plot as in the Windows 7 plot). Since the figures show the statistical mode of the allocation for a file operation each data point corresponds to

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

the middle of the largest consecutively allocated area of that file operation, thus they show real allocations.

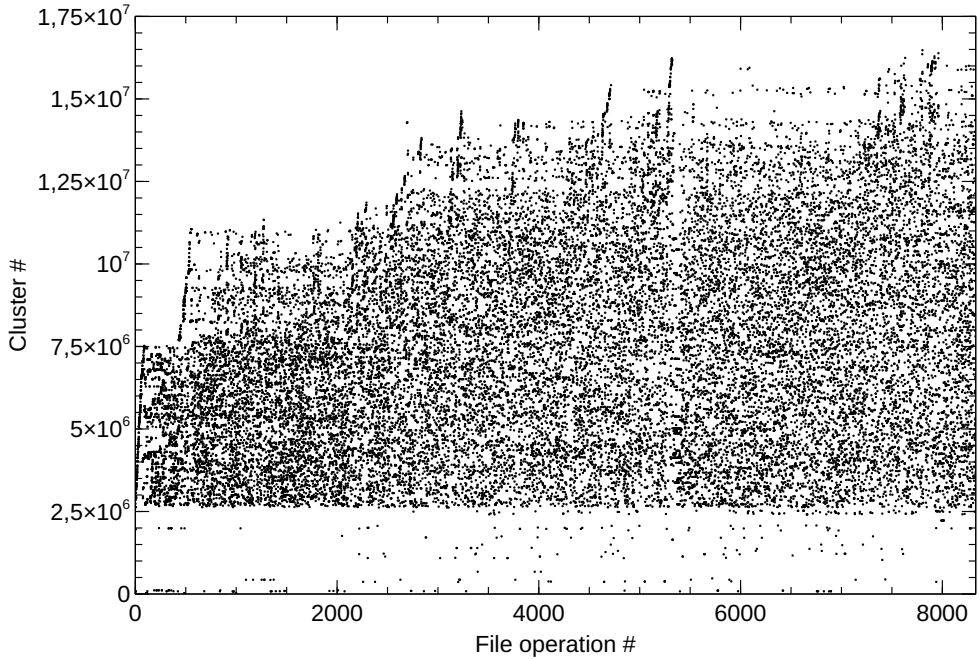


Figure D.7.: The mode allocation position for Windows 10 in 64 GiB hard drives using the standard file operation pattern.

The sparsely allocated area below cluster 2,550,000 (see Figure D.6) has a denser allocation pattern for Windows 8 and 8.1, than for Windows 7. The order of usage from sparse to dense for that area is Windows 7, 10, 8 and 8.1. The size of the sparse area is the same in the 256 GiB hard disks, hence it does not represent the 12.5% of the volume size set aside for the MFT. However, all areas contain the MFT, which starts exactly 3 GiB into the main partition in all hard disks, regardless of size and version of Windows [10]. When checking the allocation of every 50,000 cluster in the sparse area we found that almost all of the files are OS related files and no more than 5% of the files are created by the scripts.

The standard deviation value of the allocated positions after each file operation is high, between 2,000,000 and 3,000,000 clusters, and is rapidly increasing at the beginning (up to approximately 500 file operations), where it levels out. This can be seen in Figure D.8. The rapid increase at the beginning of the graph is due to the large contiguous area of free space when the disk is newly formatted (when the best fitting area available for allocation is much larger than the required space). Please observe that

standard deviation is measured in clusters, not cluster number (position) and therefore the Y-axis of the standard deviation graphs do not show the full size of a 64 GiB partition.

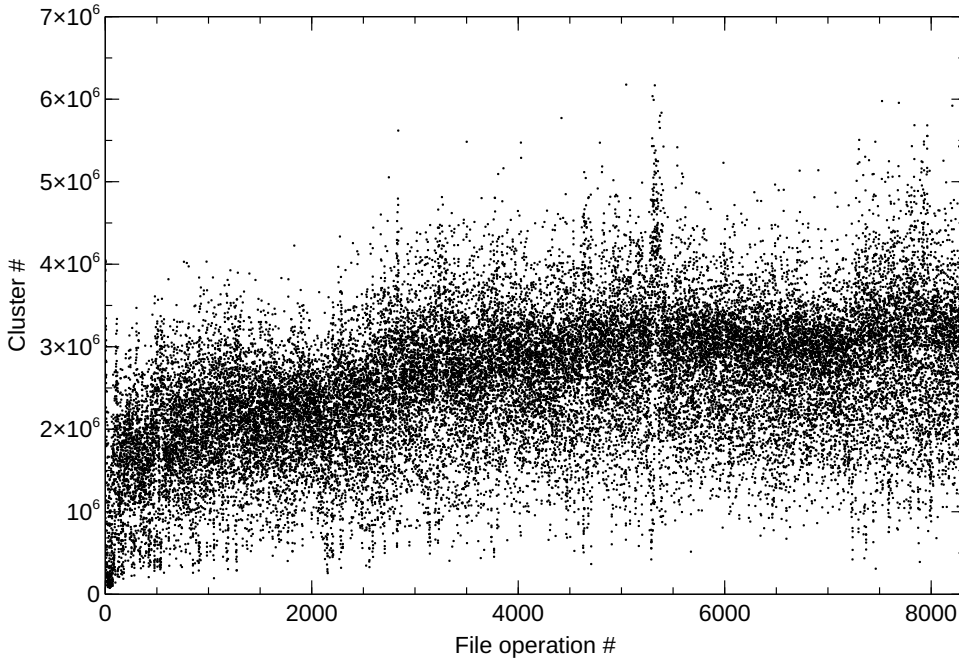


Figure D.8.: The standard deviation of the allocation position for Windows 7. The four 64 GiB partitions from the experiment using the standard file operation pattern are included. Since the standard deviation is measured in clusters, not cluster number (position), the Y-axis does not show the full size of a 64 GiB partition.

Worth noticing is that the Windows 10 standard deviation, which can be seen in Figure D.9, is more dense and less varied than for Windows 7. On the other hand it does not level out to the same degree as for Windows 7. Neither the standard deviation graph in Figure D.8 nor the graph in Figure D.9 change much if we include allocation data from all file operation patterns.

We also collected statistics on the file fragments (groups of allocated clusters) during the experiment. Three metrics are worth noticing; the number of fragments, as well as the maximum and median size of the fragments. Please observe the log scale of the Y-axis in the file fragment graphs (Figures D.10 to D.15).

The number of fragments is an indicator of the allocation algorithm's priority regarding filling holes versus keeping file data contiguous. Figure D.10 shows that in Win-

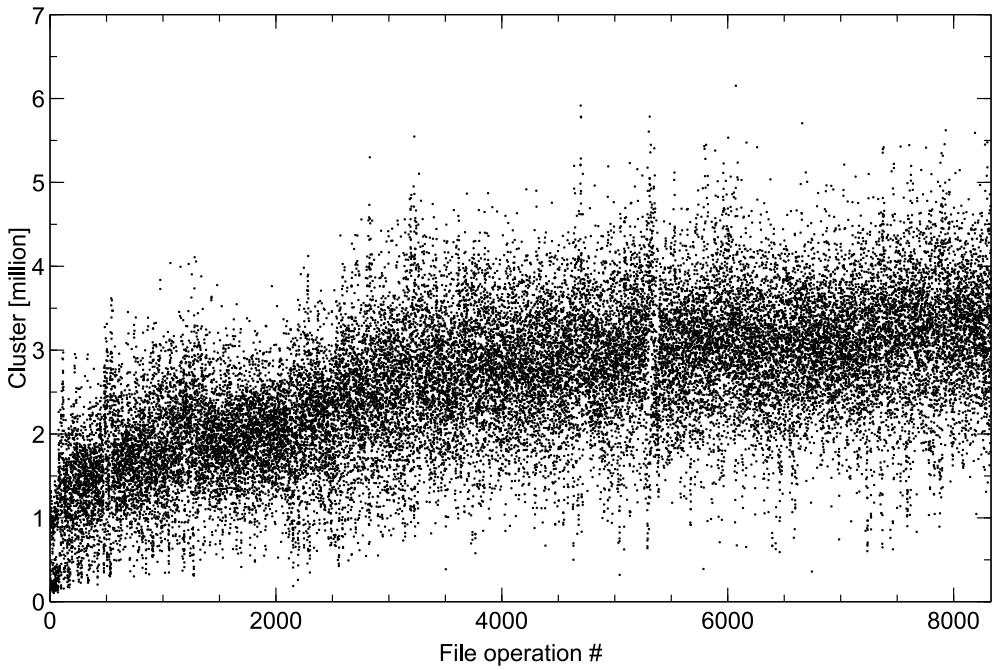


Figure D.9.: The standard deviation of the allocation position for Windows 10. The four 64 GiB partitions from the experiment using the standard file operation pattern are included. Since the standard deviation is measured in clusters, not cluster number (position), the Y-axis does not show the full size of a 64 GiB partition.

dows 7 most of the file operations (using the standard file operation pattern) generate approximately 20 fragments. As can be seen the number of fragments never reaches above 500 for Windows 7. The number of fragments also slowly increases over time, but with a solid foundation around 20 fragments per file operation.

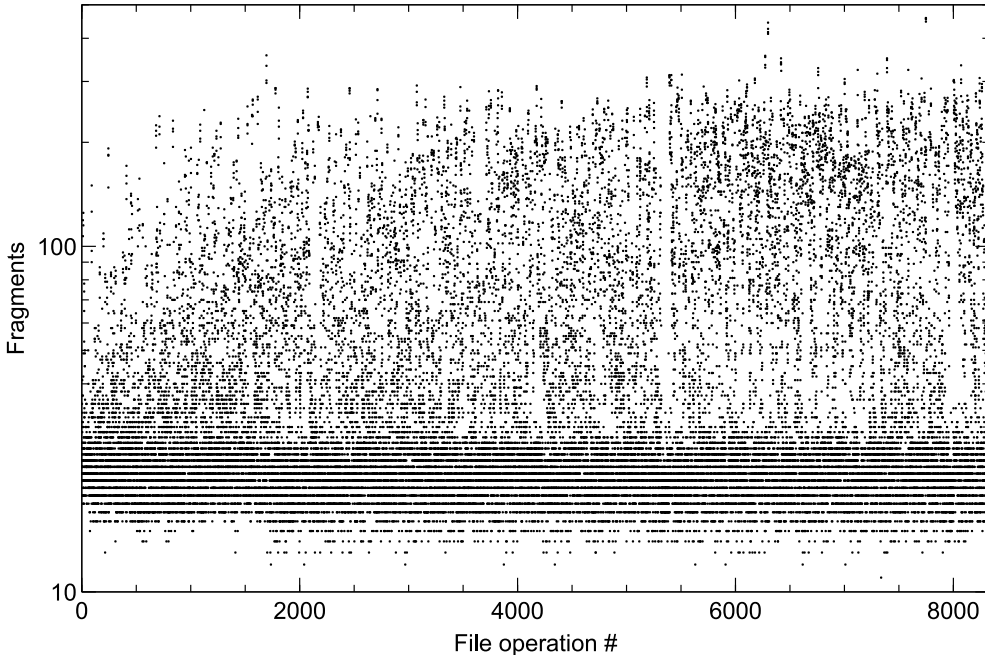


Figure D.10.: The number of fragments allocated in each file operation for Windows 7 in 64 GiB hard drives using the standard file operation pattern. Please observe the log scale of the graph.

The number of fragments per file operation for Windows 10, as shown in Figure D.11, is approximately three times larger than for Windows 7, giving a general size of 60 fragments per file operation. However, all but one file operation give well below 500 fragments, with an outlier of 2043 fragments at file operation 5765. As in Figure D.10 the trend is a slow increase of the number of fragments as the number of file operations increases.

The median size of the fragments for Windows 7 using the standard file operation pattern is shown in Figure D.12. Large fragment size values indicate that the allocation algorithm tries to keep the fragmentation down and as can be seen there are a number of median fragment sizes above 7000 clusters in the Windows 7 virtual hard drives. As can also be seen the median fragment size has a line that increases linearly from 0 to approximately 2,000 clusters (the logarithmic scale transforms the line to a curve).

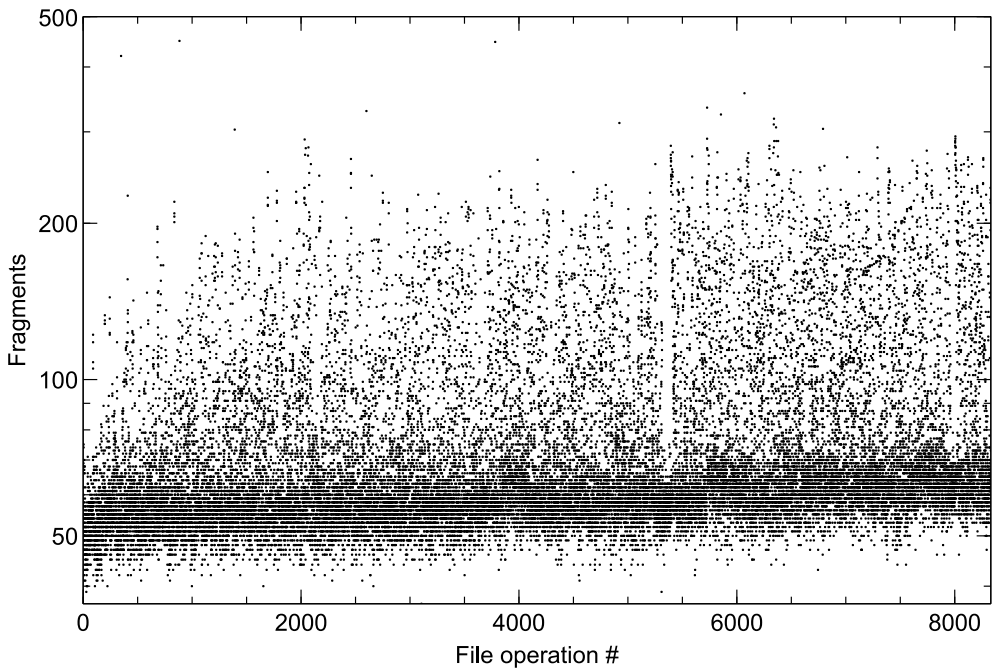


Figure D.11.: The number of fragments allocated in each file operation for Windows 10 in 64 GiB hard drives using the standard file operation pattern. Please observe the log scale of the graph and the truncated maximum value of the Y-axis, which hides an outlier of 2043 fragments at file operation 5765.

There are sharp horizontal lines at approximately the same distances in the graph, which means that due to the logarithmic scale of the Y-axis they are placed at exponentially increasing distances. There is much activity in a band centered around fragments of 64 clusters. The amount of very large median fragment sizes slowly increases towards the end of the graph.

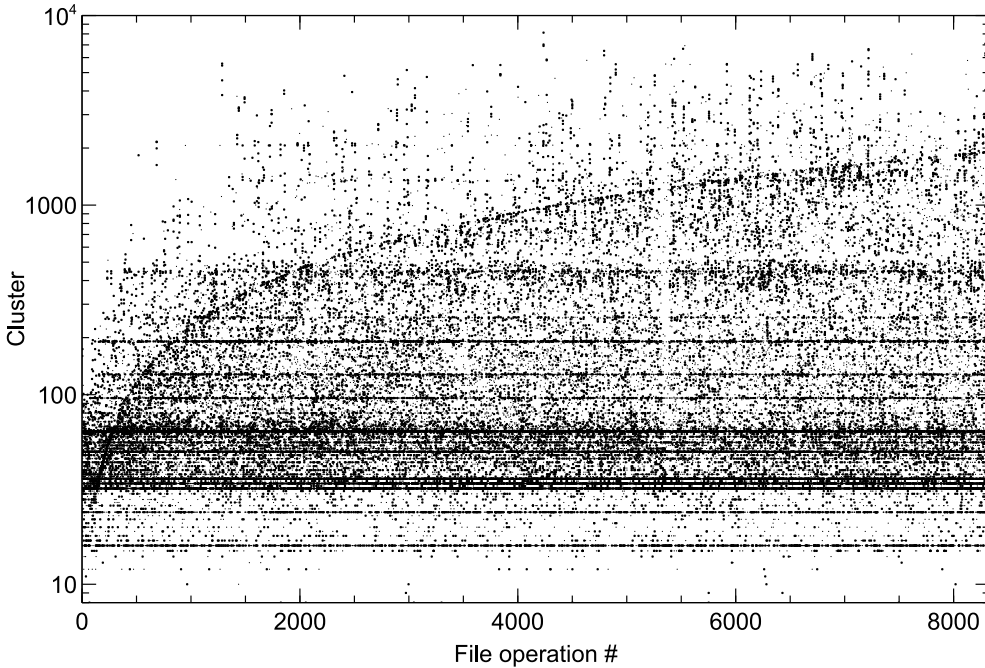


Figure D.12.: The median size of the allocated fragments for Windows 7 in 64 GiB hard drives. There is a linearly increasing trend from 0 to approximately 2,000 clusters and also a number of horizontal lines at exponentially increasing distances. Please observe the log scale of the graph.

The graph showing the median fragment size for Windows 10 in Figure D.13 lacks the linearly increasing trend found in Figure D.12. However, the horizontal lines in the Windows 7 graph are present in Windows 10 too, although starting at a lower level. The standard fragment size in Windows 10 is 16 clusters according to our results. Windows 10 has smaller median fragment sizes than Windows 7 in general, but still has the same slowly increasing amount of large median fragment sizes as Windows 7.

Figure D.14 shows how the size of the largest file fragment for each file operation is decreasing as the number of file operations is increasing. There is also a large amount of fragments of approximately 500 clusters created at the beginning of the experiment. After approximately 1,500 file operations the size of the biggest fragments level out at

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

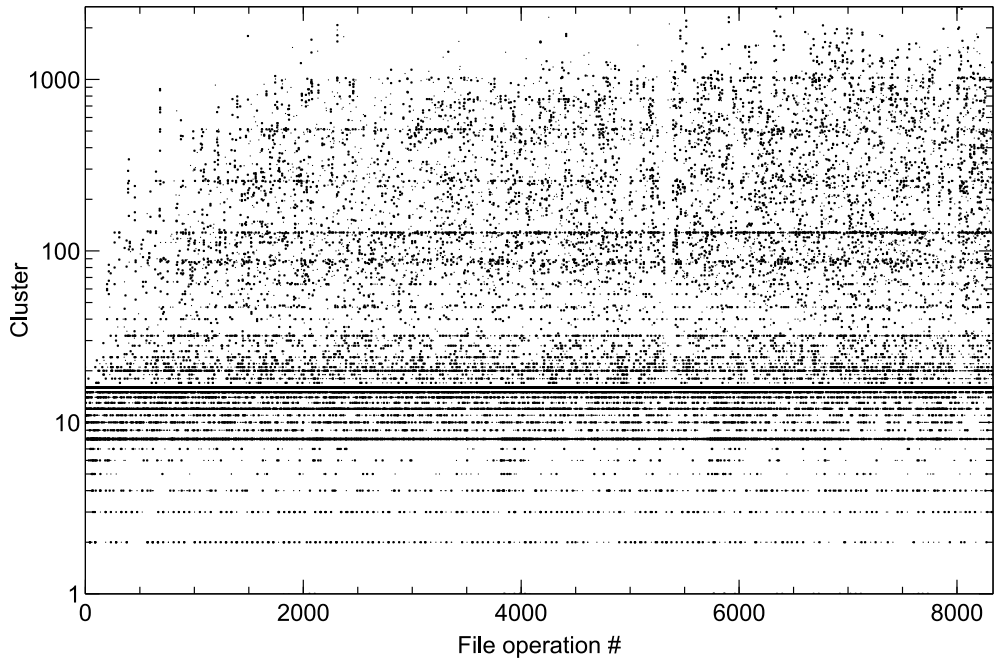


Figure D.13.: The median size of the allocated fragments for Windows 10 in 64 GiB hard drives. There are clearly visible lines at approximately 100, 250, 500, 750 and 1,000 clusters. Please observe the log scale of the graph.

approximately 20,000 contiguous clusters and the band at 500 cluster is thinner. There is also a significant amount of outliers, many of them as large as 200,000 clusters, a few times even higher. The data in Figure D.14 includes both the standard file operation pattern and unique patterns, still there is a clearly visible linearly increasing trend (remember the log scale of the Y-axis) starting at fragments of approximately 500 clusters and reaching to 2000 clusters at the end. The same line can be seen in Figure D.12. Finally there is a thin horizontal line of fragments of approximately 1500 clusters in size, which gets weaker at approximately 7000 file operations, when the linearly increasing trend reaches it.

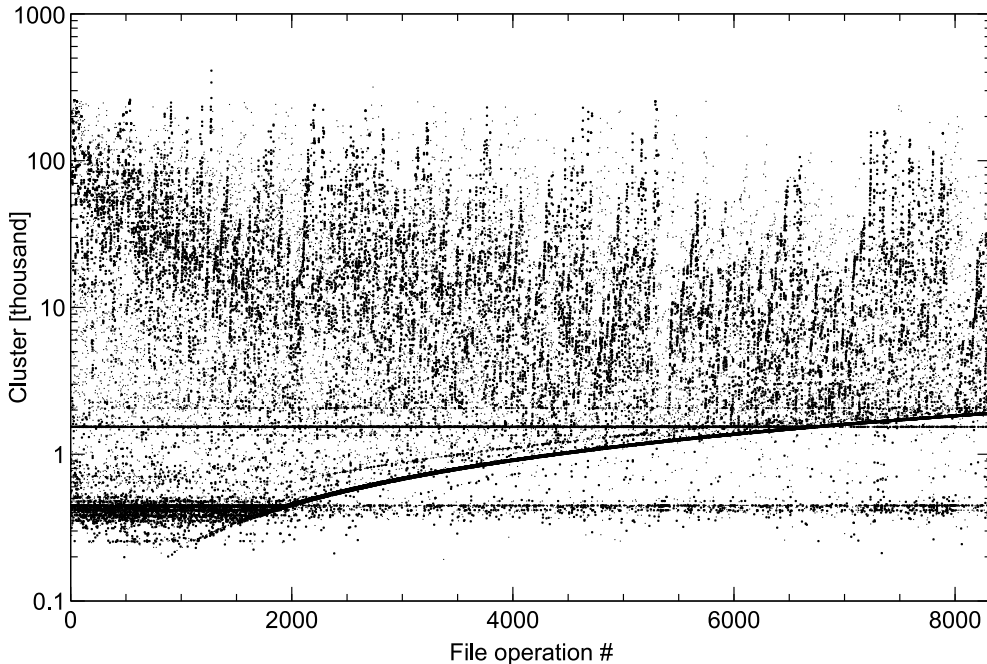


Figure D.14.: The size of the largest sequence of allocation positions (file fragments) for Windows 7 in 64 GiB hard drives. Please observe the log scale of the Y-axis.

The graph showing the largest fragment for each file operation in Windows 10 (see Figure D.15) lacks the linearly increasing trend (please do not forget the log scale) found in Windows 7. Instead there is a band of maximum fragment sizes centered around 1500 clusters. Apart from that the graph shows the same decreasing maximum fragment sizes at the beginning and the same leveling out at approximately 20,000 clusters large fragments as for Windows 7 (see Figure D.14).

As mentioned in the beginning of Section Result we omitted showing graphs of

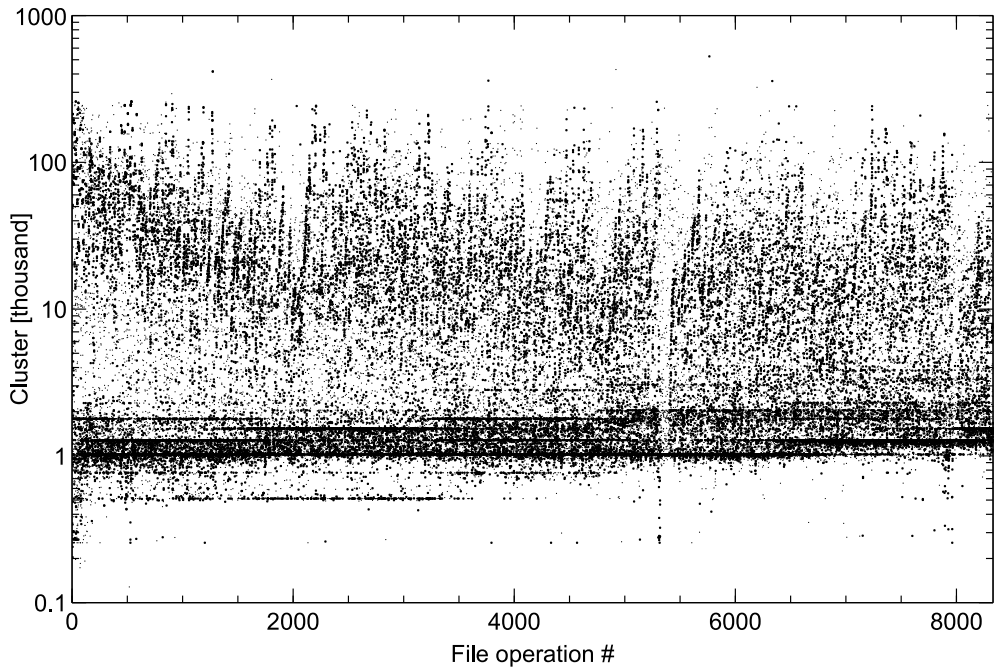


Figure D.15.: The size of the largest sequence of allocation positions (file fragments) for Windows 10 in 64 GiB hard drives. Please observe the log scale of the Y-axis.

the results for Windows 8 and 8.1 for readability reasons. Most of these results were close to the Windows 7 results, but with a few exceptions. Most notably the fragment statistics of Windows 8 and 8.1 were closer to the Windows 10 results, than the Windows 7. However, the differences were small in all cases.

D.4. Discussion

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 and its records. Since we only want to see where (which LPVAs) the OS allocates clusters when writing data the deallocation operations are irrelevant. We therefore have filtered out operations where allocated clusters have been freed. The remaining data will include clusters allocated by the system too, but that is a minor problem because the system activities often affects already allocated clusters (appending information to existing log files for example). 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 possibly give rise to a new cluster being allocated in the MFT (not until the preallocated MFT space is used up). When for example log and system files grow and require a new cluster to be allocated the cluster position will most probably be allocated to the same areas as ordinary user files. The inclusion of system file operations will therefore have a low impact on the statistical metrics used.

The main conclusion to draw from the result is that the allocation behavior differs in Windows 10 compared to the older versions of Windows, an important fact to remember during digital forensics case work involving, for example, suspicion of manipulation of file system time stamps. Another important conclusion to draw is that the allocation activity is highest in the lower middle cluster positions and only slowly moving towards the end of the partition as the file system ages. Hence any file carving searches for user data should preferably start there and not at the beginning of a hard drive.

We can also conclude that similar file operations executed in differently sized hard drives still generate similar, but not equal, results (compare Figure D.1 and D.2). The similarity might actually be even higher in reality, because the instability of the experimental platform caused unique system states for the individual virtual machines, causing system files to be written at different occasions in each machine. Those activities therefore might have allocated free areas that were allocated to files written by the scripts in other machines.

The file system utilization plots included in Figures D.1, D.2 and D.3 have been raised by 2,526,780 clusters. That corresponds to the area in the file system where the

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

OS files are written during installation. The same area is clearly visible in Figure D.6 showing the statistical mode of the allocation pattern. We found that Windows 7 and 10 are less likely to allocate files in that area than Windows 8 and 8.1 and that the sparse area has the same size regardless of size of the hard drive. Of course the size of the area will differ depending on the size of the installed OS, but the required size of a Windows installation is the same for at least Windows 7 to 10 [26–28].

Since we do not differentiate between allocations originating from the file operation scripts and system file allocations we cannot be sure what type of file has been allocated to the sparse area containing the OS files (we can only see what is currently allocated there). Furthermore, the statistical metrics only show parts of the reality, hence the allocation activity in the sparse area might be high, but only for small files. Nevertheless the results show that the allocation activity differs between areas in the partitions and between the versions of Windows included in the experiment, which is important to know in for example file carving investigations.

We have not yet found any theoretically or scientifically sound explanation of the unused area found in the middle of the Windows 7 partitions (see the median and mode graphs in Figures D.5 and D.6). There are no system files allocated there in the virtual machines from the experiment, neither the `\$MFTMirr` file as suggested by [14], nor the `pagefile.sys` as suggested by colleagues. This is also true for the six unrelated home and office computers running different versions of Windows (from 7 and up) we checked to see if the hypothesis holds for real world computers. Hence the system file hypothesis is falsified.

When checking the unused area in the middle of the Windows 7 partitions the area is allocated to files written during the experiment, although we can only see file system information for the last few hundred file operations due to (possibly) earlier deletions. The files found in the unused area have all been written after file operation 8,331, which is the upper bound used for the graphs due to a few virtual machines having to be stopped prematurely. When checking the data for the virtual machines that executed all 10,000 file operations the unused area is present for all operations for Windows 7, but for Windows 8.1 it is vanishing in the last 1,000 file operations. The `$Bitmap` files of the Windows 8 virtual machines all got out of sync for different reasons during the last 1,000 operations and hence we did not get any reliable data from them after file operation 9,000. The most probable reason is a breakdown of the `VBoxManage` service, which caused the script to download the `$Bitmap` file at the wrong occasion. The Windows 8 machines had become slower and slower over time and finally the restart timeout of 10 minutes was exceeded. The failure had a severe impact on the results after file operation 9,000. However, up to file operation 9,000 the results are reliable and the unused area is clearly visible when we plot the Windows 8 data. We can also conclude that the area is not present in Windows 10, thus Microsoft seems to have updated the allocation algorithm in Windows 10.

The unused area in the middle of the partitions might also be an artifact of the experimental setup. Since we tried to keep the virtual machines identical an error during the installation of the OS might have had a large impact on the results for the machines using the standard pattern. However, the fact that half of the included virtual machines ran unique sequences of file operations and still retained the same unused area contradicts the installation error hypothesis.

The mode allocation position graphs (Figures D.6 and D.7) for both Windows 7 and 10 are more or less evenly distributed over the used area of the storage media. This might simply be an effect of the random deletion of files during the experiment. However, if it is not the effect is that the area where there might be interesting material in a partition is increasing as the file system is utilized, hence an old hard disk requires a larger area to be searched. This is however contradicted by the slowly increasing mean and median allocation position seen in Figures D.1 and D.5.

The phenomenon of the maximum allocation position, described as the leeward effect on clouds of a mountain range, is interesting. There is a clear difference between Windows 7 and 10, where the latter is biased towards continuing using any high allocation addresses reached. This means that Windows 10 is using the storage area more evenly than Windows 7. All virtual machines used the default settings in the storage section of VirtualBox, which therefore emulated a mechanical hard disk. Hence all virtual machines should behave the same based on the hardware setup. Consequently there is a difference in the behavior of the allocation algorithms between Windows 7 and Windows 10, which needs to be studied further.

The decrease of the maximum file fragment size as the file system grows, which can be seen in Figure D.14 is natural, since when the free areas fill up and files are deleted the groups of contiguous free cluster areas will be smaller. The spikes in the graph at higher file operation numbers originate from the still unused areas at the end of the partition. If we had been able to run the experiment for an even longer period the maximum fragment size would probably have decreased even more.

The large standard deviation of the allocated positions for each file operation clearly indicates significant file fragmentation and consequently the allocation algorithm's focus on filling holes in the already used area of a partition before allocating files to the yet unused part at the end of a partition. We do not know the exact reason for this behavior, but we think it might be introduced by the fact that all file write operations use stream writing. When the OS does not know the size of the file in advance the strategy is to assume it is small and hence use it to fill in any holes in the already used area of a partition. If the file then turns out to be larger, the size of the allocated areas will automatically grow, since all small holes are occupied. On the other hand, if there is a large free area available, it is better to use that first to at least postpone file fragmentation to a situation when the partition is more heavily used. Hence the chosen strategy depends on the focus of the allocation algorithm; filling in holes or avoiding file fragmentation.

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

The best fit allocation strategy that NTFS uses is meant to decrease the amount of file fragmentation by optimizing the used area with regard to lost space at the ends of the free area that is being allocated. The behavior we can see from the result is however not fully adhering to that strategy, but that might be questioned from a philosophical point of view. If the focus lies on minimizing the lost (remaining) free area after each allocation the behavior of not using the free space at the end of the partition first and then start using the free areas left from file deletions can be understood. Fitting an allocation into a hole left by a file deletion actually leaves less remaining space around the allocated area, than if the large unused area at the end of the partition was used. If we take the large number of file fragments created by that strategy into consideration this type of behavior becomes less understandable, especially since the OSs saw the disks as mechanical hard drives, which are negatively affected by fragmentation.

D.5. Conclusion and future work

We can conclude that there actually are differences in the allocation behavior of different Windows version using NTFS, that the size of the storage media is not affecting the allocation behavior and that the behavior changes over time as the file system grows. Likewise the adherence to the best fit allocation strategy can be questioned. The allocation activity is not evenly spread over the storage area, instead it is concentrated to the already used areas. A strictly best fit allocation strategy would not fragment files if there where free space available to fit the file in one block. All Windows versions used in the experiment differentiate between mechanical hard drives and solid-state drives (SSDs), but since all virtual drives were set to emulate mechanical hard drives such differentiation cannot be the reason behind the behavior.

The results from the experiment are directly applicable to the digital forensic case work by showing that it is more probable to find older data closer to the beginning of the partition and newer data closer to the end of the used area. In the same way we have shown that the priority of the allocation algorithm is to get rid of holes left by file deletions, not to use the whole disk to decrease the risk of file fragmentation. The knowledge gained from the experiment is especially important in file carving where the goal is to reconnect fragments of files into the original files again. By decreasing the area to be search for file fragments the process will be more efficient and hence faster.

The results can also be used to improve the creation of time lines (work as another source of time stamp information) by the fact that the size of file fragments decreases as the file system grows. A file having large (and few) fragments has a higher probability of being older than a file with many small fragments, although the effect is small.

As future work we will stabilize the experimental platform and expand the scope

of the experiment to also include other file systems, hard drive sizes and OSs, as well as both stream writing and block writing file operations. We will also isolate our file operations from the OS related operations and use tools from the Sleuth Kit to increase the resolution and reliability of the results. Together these improvements will enable us to determine if it is possible to use the allocation pattern as a means to improve the reliability of time stamps and possibly even work as a sequential time stamp, showing the writing order of files. The results will also be used to find out more about the standard fragment size, number of fragments, their probable placement on disk (logical position) etcetera, which will be of great help in file carving situations. The information on differences between stream and block writing operations can also be used to improve file carving processes by giving a first indication of the type of file of a fragment and also when finding the correct ordering of the found fragments.

D.6. Acknowledgements

The research leading to these results has received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project Ars Forensica grant agreement 248094/O70. We would also like to thank FOI for their support by letting us use their Cyber Range And Training Environment (CRATE) computer cluster.

D.7. Bibliography

- [1] D. Quick and K. Choo. “Impacts of increasing volume of digital forensic data: A survey and future research challenges.” In: *Digital Investigation* 11.4 (2014), pp. 273–294. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.09.002.
- [2] P. Gladyshev and J. James. “Decision-theoretic file carving.” In: *Digital Investigation* 22.Supplement C (2017), pp. 46–61. ISSN: 1742-2876. DOI: 10.1016/j.diin.2017.08.001.
- [3] European Police Office (Europol). *Internet Organised Crime Threat Assessment (IOCTA) 2016*. Tech. rep. European Cybercrime Centre (EC3), 2016.
- [4] D. Quick and K. Choo. “Data reduction and data mining framework for digital forensic evidence: Storage, intelligence, review and archive.” In: *Trends & Issues in Crime and Criminal Justice* 480 (Sept. 2014), pp. 1–11. ISSN: 1836-2206.
- [5] F. Breitingner, G. Stivaktakis, and H. Baier. “FRASH: A framework to test algorithms of similarity hashing.” In: *Digital Investigation* 10.Supplement (2013). The Proceedings of the Thirteenth Annual DFRWS Conference, S50–S58. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.06.006.

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

- [6] V. Roussev. “Managing Terabyte-Scale Investigations with Similarity Digests.” In: *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*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 19–34. ISBN: 978-3-642-33962-2. DOI: 10.1007/978-3-642-33962-2_2.
- [7] E. Casey. “Digital Stratigraphy: Contextual Analysis of File System Traces in Forensic Science.” In: *Journal of Forensic Sciences* 63.5 (2018), pp. 1383–1391. DOI: 10.1111/1556-4029.13722.
- [8] M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Disk Cluster Allocation Behavior in Windows and NTFS.” In: *Mobile Networks and Applications* 25.1 (Feb. 2020), pp. 248–258. ISSN: 1572-8153. DOI: 10.1007/s11036-019-01441-1.
- [9] M. Karresand, S. Axelsson, and G. Dyrkolbotn. “Using NTFS Cluster Allocation Behavior to Find the Location of User Data.” In: *Digital Investigation* 29 (2019), S51–S60. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.018.
- [10] M. Karresand, Å. Warnqvist, D. Lindahl, S. Axelsson, and G. Dyrkolbotn. “Creating a Map of User Data in NTFS to Improve File Carving.” In: *Advances in Digital Forensics XV*. Cham: Springer International Publishing, 2019. Chap. 8, pp. 133–158. ISBN: 978-3-030-28752-8. DOI: 10.1007/978-3-030-28752-8_8.
- [11] R. Poisel and S. Tjoa. “A Comprehensive Literature Review of File Carving.” In: *2013 International Conference on Availability, Reliability and Security*. Sept. 2013, pp. 475–484. DOI: 10.1109/ARES.2013.62.
- [12] A. Pal and N. Memon. “The evolution of file carving.” In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [13] Net Applications.com. *Desktop Operating System Market Share*. Sept. 2017. URL: <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0>.
- [14] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005. ISBN: 0321268172.
- [15] A. Silberschatz, P. Galvin, and G. Gagne. *Operating System Concepts*. 9th ed. Wiley, Dec. 2012.

- [16] Microsoft. *How NTFS Works*. Last accessed 30-09-2018. 2018. URL: [https://technet.microsoft.com/pt-pt/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx).
- [17] J. Hughes. *The Four Stages of NTFS File Growth*. Last accessed 24-10-2018. Oct. 2009. URL: <https://blogs.technet.microsoft.com/askcore/2009/10/16/the-four-stages-of-ntfs-file-growth/>.
- [18] S. Key. *File Block Hash Map Analysis*. Last accessed 28-04-2018. 2012. URL: <https://www.guidancesoftware.com/app/File-Block-Hash-Map-Analysis>.
- [19] R. van Baar, H. van Beek, and E. van Eijk. “Digital Forensics as a Service: A game changer.” In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S54–S62. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.03.007.
- [20] H. van Beek, E. van Eijk, R. van Baar, M. Ugen, J. Bodde, and A. Siemelink. “Digital forensics as a service: Game on.” In: *Digital Investigation* 15 (2015). Special Issue: Big Data and Intelligent Data Analysis, pp. 20–38. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.07.004.
- [21] J. Jones, T. Khan, K. Laskey, A. Nelson, M. Laamanen, and D. White. “Inferring Previously Uninstalled Applications from Residual Partial Artifacts.” In: *Annual ADFSL Conference on Digital Forensics, Security and Law*. 2016, pp. 113–130.
- [22] K. Fairbanks and S. Garfinkel. “Column: Factors Affecting Data Decay.” In: *Journal of Digital Forensics, Security and Law* 7 (2012). DOI: 10.15394/jdfsl.2012.1116.
- [23] K. Fairbanks. “A Technique for Measuring Data Persistence Using the Ext4 File System Journal.” In: *2015 IEEE 39th Annual Computer Software and Applications Conference*. Vol. 3. July 2015, pp. 18–23. DOI: 10.1109/COMPSAC.2015.164.
- [24] K. Fairbanks. “An analysis of Ext4 for digital forensics.” In: *Digital Investigation* 9. Supplement (2012). The Proceedings of the Twelfth Annual DFRWS Conference, S118–S130. ISSN: 1742-2876. DOI: 10.1016/j.diin.2012.05.010.
- [25] B. Carrier. *TSK Tool Overview*. 2014. URL: http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.
- [26] Microsoft. *System requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-gb/help/12660/windows-8-system-requirements>.

D. An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

- [27] Microsoft. *Windows 10 system requirements*. Last accessed 30-04-2018. Nov. 2017. URL: <https://support.microsoft.com/en-us/help/4028142/windows-windows-10-system-requirements>.
- [28] Microsoft. *Windows 7 system requirements*. Last accessed 30-04-2018. Apr. 2017. URL: <https://support.microsoft.com/en-us/help/10737/windows-7-system-requirements>.

Part III.

Appendices

E. Extended Introduction

Although the research described in the thesis is applicable to any situation where digital data are handled, the focus of the thesis is applications within the digital forensics field and especially file carving. This chapter therefore gives a brief background on the matter to readers not familiar with the digital forensics field or file carving.

E.1. Forensics

The term *forensic* goes back to classical Latin, where it meant

of or belonging to the Forum, of or connected with the law courts. [1, Etymology:]

Consequently the term is strongly connected to properties such as law, traceability, logic and argumentation. The current meaning is similar, stated as

[o]f, relating to, or associated with proceedings in a court of law; suitable for or appropriate to pleading in court. Now chiefly in legal use. [1, A. *adj.* 1. a.].

Edmond Locard [2] started the first forensic laboratory in 1910 in Lyon, France. In 1934 he formulated what is now known as his *exchange principle*¹. The principle points to the fact that any contact between two (physical) entities leaves a trace on both of them, however not always visible by the naked eye. The work of Locard led to the start of the forensic sciences [2], sometimes also called criminalistics.

The term forensic sciences is defined as

the application of the methods of the natural and physical sciences to matters of criminal and civil law. [...] Almost any science can be a forensic science because almost any science can contribute to solving a crime or evaluating a civil harm. [6].

¹Edmond Locard wrote “Le principe est celui-ci. Toute action de l’homme, et a fortiori, l’action violente qu’est un crime, ne peut pas se dérouler sans laisser quelque marque” on page 8 of his book *La Police et Les Méthodes Scientifiques* from 1934, printed by Editions Rieder, Paris [3, p. 23]. The sentence translates to “The principle is this. Any action of an individual, and obviously, the violent action constituting a crime, cannot occur without leaving a trace” [4, 5].

E. Extended Introduction

Kirk studied Locard's exchange principle closely and saw its limitations. He therefore pointed out the importance of *individualization*, that it is not enough to prove the contact between any two entities, there must also be a proven connection to an individual perpetrator to enable its use in a court of law. A typical example is fingerprints on a murder weapon. Kirk wrote that

[t]he real aim of all forensic science is to establish individuality, or to approach it as closely as the present state of the science allows. *Criminalistics is the science of individualization*. [7, p. 236]

Hence without being able to correctly connect a crime to a perpetrator we cannot solve crimes and the legal system will not work. Especially since the legal system in democratic countries use benefit of doubt to avoid sending innocent persons to jail.

E.2. Digital Traces

Since also the digital domain is physical at the lowest level Locard's exchange principle is valid there too[8, 9], but even at higher levels exchanges of information is carried out [10] and consequently leaving traces. Regarding individualization there is however a problem.

In a physical environment the exchange ties an individual to a specific location and action (*Individualization*), and in the virtual environment the exchange can achieve the same goal for a *user account*. Unlike a physical crime scene, it will never be possible to use the Exchange Principle to physically identify the person operating a system at a given time, strictly because an exchange took place. No physical exchange takes place between the physical user and the virtual environment. [...] It is left to other means to determine identification of the physical person who was using the account on the system at the time the data were created. [8, p. 22]

Although it is stated in the citation that there is no physical exchange between the user and the virtual (digital) environment, there still is an indirect exchange. As long as a user is interacting with a digital device there is a connection between the two domains. There are for example individual differences in how we handle the keyboard [11, 12]. Furthermore, stronger authentication methods, as well as biometric based authentication techniques, makes the individualization easier in the digital domain.

The digital traces can either be *actively* left by the user through intentional actions, such as saving a photo or document, or they can be left *passively* through for example logs and browser history [13]. When the term *user data* is used in this thesis it should

be interpreted as referring to both types, although the primary meaning is data created by direct actions of a user.

User activities are usually of higher value to a digital forensic investigator than passively created data. However, the value of logs and browser history should not be neglected and can sometimes even be more valuable than actively created user data. The two types of user data are also tightly connected. Often a log post is added when a file is saved, or any other significant action involving a file or data is taken by the user.

The data (traces) left by the user can furthermore be divided into *content* and *meta-data*. Content is for example the words and sentences of documents, the sounds and images of multimedia and the conversations of instant messaging applications. The metadata on the other hand are data about the content, for example time stamps, location data of images and formatting information of documents. Depending on the situation one or several of the combinations of actively or passively deposited content or metadata can be of interest to the digital forensic investigator [13]. The possible combinations can be seen in Table E.1.

Table E.1.: The 2x2 matrix showing the connection between the two ways to deposit data and the two types of data.

Active/Content	Passive/Content
Active/metadata	Passive/metadata

There are also *system data* held in the file system in the form of system files. These are files that are part of the OS and most of them are created at the time of installation of the system. From an allocation point of view there is no difference between user files and system files. The latter are created from strict system activities like installation and updating the system or software in the background.

E.3. Digital Forensics

The forensic science used when solving crimes in the digital domain is called *digital forensics*, computer forensics or cyber forensics. There is not yet any strict definition of the term digital forensics and it is better described by general wording, than through strict definitions, according to Subramaniam [14]. Yet there are a number of definitions given in the literature.

In 2001 a group of digital forensic researchers at the first Digital Forensic Research Workshop (DFRWS) USA agrees on defining digital forensics as

[t]he use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, docu-

E. Extended Introduction

mentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations. [15, p. 16]

The DFRWS definition is rather long and complex due to its specific listing of actions to be executed. A shorter alternative is presented by US-CERT in 2008. They define computer forensics as

[...] the discipline that combines elements of law and computer science to collect and analyze data from computer systems, networks, wireless communications, and storage devices in a way that is admissible as evidence in a court of law. [16, p. 1]

The US National Institute of Standards and Technology (NIST) also presents three definitions of digital forensics [17] and gives computer forensics as a synonym. The definitions are taken from different sources. The first definition is part of a longer definition from the US Department of Defense (DoD) and states that digital forensics is

[i]n its strictest connotation, the application of computer science and investigative procedures involving the examination of digital evidence - following proper search authority, chain of custody, validation with mathematics, use of validated tools, repeatability, reporting, and possibly expert testimony. [18, p. 14]

The second definition is taken from a publication by NIST, which states that

The application of science to the identification, collection, examination, and analysis, of data while preserving the integrity of the information and maintaining a strict chain of custody for the data. [19, p. C-1]

The third definition of digital forensics originates from a NIST publication on cloud forensics. The definition of the term is there given as

[t]he process used to acquire, preserve, analyze, and report on evidence using scientific methods that are demonstrably reliable, accurate, and repeatable such that it may be used in judicial proceedings. [20, p. 24]

An alternative to the NIST definitions is given in a book from 2018. This definition is less scientific in formulation. The definition states that

[d]igital forensics is a branch of forensic science that uses scientific knowledge for collecting, analyzing, documenting, and presenting digital evidence related to computer crime for using it in a court of law. The ultimate goal is knowing what was done, when it was done, and who did it. [21, p. 2]

To separate digital forensics from other forms of forensics that happen to involve computers and forensic computing, for example automatic fingerprint matching, the following definition has been proposed:

One possible characterization of digital forensics is that it examines events (or traces of events) that happened in the digital realm; the purpose of digital forensics then is to determine the root cause of, or to reconstruct events that happened in the digital realm. [22, p. 56]

Three more definitions of digital forensics are given in [14]. From the large amount of definitions, also in recent years, we can draw the conclusion that the (research) field is maturing, but still is driven by operational needs and not yet by scientific theory.

There are suggestions for a formalization of the research field, allowing the forensic process to be scientifically defined. However, there are only two possible scientific claims to be made within digital forensics [22].

1. That the digital data examined is an example of a specific class of artifact; and/or
2. That the digital data examined proves or disproves a claim that the data was [sic!] the result of specific data transformed by a specific computational process. [22, p. 53].

What the definition is saying is that digital data are not randomly appearing, they always belong to a directly or indirectly human controlled action and/or that they only show whether they are the result of a specific action, or are not the result of that specific action. In other words, we cannot draw too extensive conclusions from the existence of some specific data, only that a (known) process created them.

Currently there are many subcategories to digital forensics, new categories are continuously added and existing categories are merged. The following list contains 11 subcategories. However, the list only represents a snapshot of the situation in 2018, when the list was created [23].

- file system forensics
- memory forensics
- operating system forensics

E. Extended Introduction

- multimedia forensics
- network forensics
- database forensics
- malware forensics
- mobile device forensics
- e-mail forensics
- firewall forensics
- financial forensics

Since new digital technologies and services are constantly emerging the content of the list will change. It is included to show the width of the applicability of digital forensics. Two new fields not in the list are car forensics and IoT forensics.

E.4. Forensically Sound

In the digital forensic domain the term *forensically sound* is used to denote evidence that is trustworthy and faithful to the original. The reason for the need to have a certain term for high fidelity evidence is the volatility of digital data. The acquisition process often requires a digital tool to be loaded onto the suspect's device to enable extraction of any data, especially when working with volatile storage like RAM. The loading process and also the execution of the tool will affect the content of the RAM.

The modification of the source data might however be acceptable as long as the original meaning of the evidence is preserved. The tool's effects on the evidence should also be fully known and documented [24]. The situation is described with the wording "generally considered forensically sound" [24, p. 50], i. e. it is not strict. The full statement reads:

Provided the acquisition process preserves a complete and accurate representation of the original data, and its [sic!] authenticity and integrity can be validated, it is generally considered forensically sound. [24, p. 50]

The term forensically sound is often taken for granted and implicitly defined in terms of usability in a court of law. This view is manifested in a book [21] from 2019, one of a few that actually tries to define the term.

“Forensically sound” is a term used in the digital forensics community to describe the process of acquiring digital evidence while preserving its integrity to be admissible in a court of law. [21, p. 3]

A thorough definition of the term, together with a discussion on previous definition attempts, and accompanied by four criteria to be used to evaluate a candidate for a forensically sound process, is stated as

[t]he application of a transparent digital forensic process that preserves the original meaning of the data for production in a court of law. [25, p. 10]

The four criteria that has to be met to fulfill the definition of a forensically sound process [25] are:

Meaning *Has the meaning and, therefore, the interpretation of the electronic evidence been unaffected by the digital forensic process?* [25, p. 11] Here we see the collision between *unaltered* (which often means not possible to extract) and *preserved* (meaning that even though the bits might be different, the intention of them is preserved). An example is a time stamp shown in another format than the original, but with the same temporal information.

Errors *Have all errors been reasonably identified and satisfactorily explained so as to remove any doubt over the reliability of the evidence?* [25, p. 12] All errors should be detectable and their implications for the process and evidence fully understandable. A known error is not good, but possible to handle. An unknown error breaks the chain of evidence.

Transparency *Is the digital forensic process capable of being independently examined and verified in its entirety?* [25, p. 12] Transparency is similar to the scientific requirement of repeatability. Any process should be described with such detail that it can be repeated independently by someone else achieving the same result.

Experience *Has the digital forensic analysis been undertaken by an individual with sufficient and relevant experience?* [25, p. 13] The requirement of possessing enough relevant experience to conduct an examination can be seen as an extra level of security. If the three previous requirements have been fulfilled the process should (logically) be forensically sound. However, an experienced person will lessen the risk of creating general errors in the process and has the capability to detect any errors, uphold the transparency and decide whether the meaning of the found evidence is preserved.

Even though the requirements of a forensically sound process often are implicit, the prevailing lack of a common understanding of the term lessens the value of the evidence. By adhering to some standard or definition of the term mistakes weakening the chain of evidence value can be avoided [24, 25].

E.5. Digital Forensic Models

There are probably as many models of the digital forensic process as there are definitions. However, most of the models are built around a core of four basic steps [26]. These steps were first mentioned in 1995 and have been included ever since [27].

Four steps similar to those in [26] are given by the US National Institute of Justice (NIJ) in 2001 and the same four steps are later accompanied by three more in the DFRWS version [15]. The DFRWS model has been further enhanced to include nine steps that incorporate the full digital forensic process, from detection of an incident, to the evidence being handled after the case has been closed [28].

The steps presented in [28] are²:

Identification of an incident and its type.

Preparation of the needed administrative and operational steps later in the process.

★ **Approach strategy** includes planning the detailed steps to be taken to be able to collect as much original evidence as possible without unnecessary harm to the victim.

Preservation of the evidence to protect it from manipulation and destruction before collection.

★ **Collection** of the data (information) from the seized information carrier(s) in a forensically sound way.

★ **Examination** of the collected data in a systematic and thorough way to identify and locate evidence therein.

★ **Analysis** of the evidence identified during the examination step. The examination and analysis steps might be repeated if necessary, until a satisfactory amount of evidence is collected.

Presentation prepare and document conclusions drawn from the evidence found in the analysis step.

²The original four steps [26] are italicized and the star (★) marks the steps in the model where the results of the PhD project can be applied.

Returning evidence to the original owner and remove unimportant evidence from the case.

The digital forensic models have developed further since the model presented above. They have been extended to better fit the actual work of the law enforcement personnel, been parallelized or distributed, divided into phases to accommodate even more detailed steps and modified in other ways depending on the focus of the research done by the model creators. More and updated information on the different models can be found in for example [27, 29, 30].

The *examination* step in the digital forensic models can be further divided into three distinct levels [31], namely

A survey or triage forensics inspection, which is similar to the medical triage process and is used to get a fast review of the content of the available information carriers at a crime scene.

A preliminary forensic examination, which is used to give investigators enough information of the contents of the seized information carriers to be able to proceed with their evidence collection and forming a first idea of possible leads to follow.

An in-depth forensic examination, which is more thorough than the previous two and is meant to give the investigators an extensive view of the information in the material at hand. This will then help them to fully understand the offense and address the important questions.

All three levels are addressed in the PhD thesis. The connections to the model steps described in this section are further discussed in Chapter 6.

E.6. Digital Forensic Challenges

Apart from the challenge of the increasing amount of data to handle there are a number of other challenges affecting the digital forensic process. In 2010 the major need for future research was the creation of abstraction levels to cope with the diversity and rapid development within the digital device area [32]. One of the needed abstractions (research fields) was given as

File system metadata e.g. such as timestamps, file ownership, and the physical location of files in a disk image. [32, p. S69]

A number of digital forensic professionals were asked what they saw as the greatest challenges to the field in 2021. Data recovery was given as the second most important challenge by 68% of the participants. Only network forensics was given a higher

E. Extended Introduction

score by 77% of the participants. The three most pressing needs within data recovery were recovery of encrypted content (48%), memory forensics (32%) and disk and storage (13%) according to the participants. The remaining non-zero needs were carving, recovery of multimedia files and data hiding at 3% each [33].

The opinions on the challenges for the digital forensics community until 2023 have been collected from 24 digital forensic experts. There are a large number of challenges listed, but those relevant to this thesis are [34]

- structured mapping of all devices
- data reduction within legal boundaries
- tools for spotting unusual patterns within data

The above challenges are not explained, only listed, hence structured mapping of all devices might be interpreted in two ways. Either is related to a need for the first responder not to miss any digital devices due to an unstructured collection behavior, or it should be interpreted as a need for a structured view of the content of all devices. The first alternative is trivial and can be solved by the first responder following standard procedures. Hence it would probably not be mentioned as a problem at all, which leaves us with the conclusion that the problem concerns the lack of a structured view of the contents of all devices.

All challenges listed above can be related to the lack of information on the inherent structures caused by the allocation algorithm in the file system. And based on the publication dates of the references it does not yet seem to have been solved.

E.7. File Carving

The digital forensics data processed in the models are most of the time consisting of files taken from a working file system, or any other working structuring of data, for example a database. However, sometimes the structuring has been broken for some reason, due to a hardware failure or intentional deletion by the user. In those cases the investigator must use *file carving* techniques to extract the data.

File carving is a forensics technique that recovers files based merely on file structure and content and without any matching file system meta-data. [35, p. 62]

The research field of file carving is an important part of digital forensics and is used in situation where the file system of the partition either cannot or shall not be used. Instead the stored data are handled directly and often regarded as an unsorted set of blocks of different types of data (files) [36]. The blocks of data forming a file need

not be consecutive (due to fragmentation), neither need the data fragments be stored in any specific order (because the unallocated areas best fitting a file might be found anywhere in the partition).

A formalization of the file carving research field has been suggested. By formalizing the field investigators and other professionals working with digital forensics get a better understanding of the process and a common vocabulary to help them discuss, compare and evaluate their results and tools [37].

The formalization of the file carving research field includes the use of a new taxonomy of file recovery. It also incorporates the use of different standardization frameworks to improve the quality and reliability of the software tools used for file recovery. The tools should be accompanied by a documented verification of their quality, including comprehensive testing to lower the risk of bugs and errors. This will in the end increase the confidence in the digital forensics discipline and improve the rule of law [37].

There are two major ways of carving a file or fragment to be used for comparison with existing material. The first method directly reads the data, which are categorized on different properties of the fragments of the file content (*file fragment carving*). In this way any file, previously known or unknown, can be found and reassembled [38–40].

The second method of carving files is using blocks of data (one or more fragments) that first are hashed and then compared to hashes of equally sized blocks of known data from the original file (*hash-based carving*). The large amount of hash comparisons made by a hash-based carving algorithm puts an extra burden on the forensic process. Therefore different strategies, techniques and algorithms to improve the execution speed of the hash-based carving algorithms have been developed [41–47]

A more detailed division of the file carving field than file fragment and hash-based carving is to divide the field into five different approaches [48]. These are general and covers both the fragment classification, as well as the reassembly of the found fragments:

General carving focuses on detecting the file type of fragments by using magic numbers in the file header and footer. Also the use of file (name) extensions is included, although it is not strictly file carving since it uses information from the file system. General carving cannot properly handle file fragmentation, because it assumes the files are written contiguously in the partition [48].

Specific file type carving focuses on carving a specific file type, for example JPEG. Since the structure of the specific file type is used for carving these algorithms cannot handle other types of files or data, but are good at what they do [48].

File system carving focuses on utilizing structures and features of the file system

E. Extended Introduction

allocation behavior to carve files. Each file system has its own features to use and each carving algorithm is only applicable for its specific file system [48].

Carving based on structure focuses on the specific features of individual files, but the foundation of the carving is a general applicability given by the use of machine learning and AI algorithms [48]. However, that implies access to a large amount of training data (files having features similar to the file to be carved).

Carving based on fragmentation focuses on the type and amount of fragmentation of a file. The algorithms can also carve files with missing headers, footers or fragments, hence it can carve incomplete files [48].

Carving based on fragmentation is further divided in two parts. Both are mentioned by the authors as promising techniques with future potential. The two parts are:

Content based carving utilize for example byte frequency distribution and entropy metrics of the content of the file fragments to be carved, hence they are file fragment carving based methods [48].

Smart carving is a hash-based³ file carving method where (cryptographic) hash algorithms are used to match blocks of unknown data with blocks of known data in a database [48].

There is currently no general file carving algorithm that handles all types of files and situations. However, by combining algorithms from several approaches and utilizing machine learning, automation, content-based carving and smart carving, a generic file carving ability can be reached [48].

Another way of dividing the file carving field is presented by Poisel et al. [49], who describe the research field through a taxonomy of data fragment classification techniques. The authors have divided the research field into five classes:

Signature-based approaches build on the concept of magic numbers, i. e. specific byte sequences in the header and footer of files. For example a JPEG file starts with 0xFFD8 and ends with 0xFFD9. The category also include methods from the hash-based carving field.

Statistical approaches utilize different statistical metrics for classification, for example entropy or the Byte Frequency Distribution (BFD).

³A (cryptographic) hash of a block of data can be seen as a much shorter fingerprint of the data in the block. However, due to the fact that a large amount of data is mapped to a smaller amount (the fingerprint) there might be collisions, where data blocks with different content get identical hash values. However, such collisions are highly unlikely, the longer the hash value, the lower the collision probability.

Computational intelligence based approaches use machine learning and artificial intelligence for classification. Typically algorithms such as k-Nearest Neighbor and Support Vector Machine (SVM) are used.

Approaches considering the context use information from surrounding fragments for classification. The method is based on the assumption that the external fragmentation is low in most file systems.

Other approaches contain for example methods to visually separate data types and methods based on combinations of the other approaches.

The signature-based methods using hash algorithms are related to the techniques used for hash-based file carving, where hashes of fragments of known files are compared to hashes of unknown file fragments.

When the fragments of a file have been found they should be reassembled into the original file again, if possible [35, 36, 50, 51]. The reassembly process can be divided into three parts based on the technique used [36]. The parts are:

File-signature based approaches use file headers and footers, as well as other meta-data present in the detected fragments.

Mapping based approaches use a discriminator that reads a file sequentially and continuously verifies its integrity. This is combined with an ability to accept jumps in the position of the fragments found in the partition. When the position reaches the end of the media it wraps around and allows for a continuation of the sequence of fragments at positions closer to the start of the partition. This requires the fragments to be placed sequentially further and further away from the start of the partition (except when the position wraps around the end of the media).

Graph based approaches use a graphing algorithm that builds a graph of all fragments and then traverses it based on different features (often using a greedy algorithm) finding the optimal path through the graph. In this way the top candidates of the reassembled file can be evaluated and the correct alternative be found. The approach builds on the existence of a function that can evaluate the probability of a correct joint between two fragments.

The graph based Parallel Unique Path (PUP) algorithm is currently popular and is used in many file carvers. It was introduced into the file carving field in 2006 and is a variant of the Dijkstra shortest path algorithm [52]. There are also other algorithms for calculating the correct combination of fragments as a path through the graph [53].

When file carving is used outside of the forensic domain it is often done by archivists, librarians and other professions dealing with information in old storage media, which might be more prone to breaking than newer media [54–57]. Often the carving methods are used to explore the contents of a partition with the goal of sorting, ordering and reassembling as many of the files as possible, based only on properties of the data set itself. It is therefore important to be able to correctly categorize the data type of the fragments found, as well as being able to rapidly reassemble the fragments into (parts of) the original file again [54, 56].

Different challenges are affecting the file carving field. Two of them are relevant to the PhD project. They are processing time, and (massive and complex) fragmentation [48]. Both areas can benefit from the results of the PhD project. The processing time can be lowered with the help of the information on the probable file (fragment) position within a partition. The problem of complex fragmentation will benefit from the study of different allocation patterns depending on the type of file to be written.

E.8. Bibliography

- [1] OED Online. *forensic, adj. and n.* Last accessed 17-04-2022. Mar. 2022. URL: <https://www.oed.com/view/Entry/73107?result=1&rskey=Nvwb7M&>.
- [2] J.-P. Brodeur, G. Kelling, T. Whetstone, W. Walsh, and M. Banton. *Crime-scene investigation and forensic sciences*. Last accessed 17-04-2022. Dec. 2021. URL: <https://www.britannica.com/topic/police/Crime-scene-investigation-and-forensic-sciences>.
- [3] F. Crispino. “Le principe de Locard est-il scientifique? Ou analyse de la scientificité des principes fondamentaux de la criminalistique.” PhD thesis. Institut de Police Scientifique, Ecole des Sciences Criminelles, Faculte de Droit, Université de Lausanne, May 2006.
- [4] S. Dobrowski. *Forensic Fact of the Day - Quotations - Edmond Locard*. Jan. 2013. URL: <https://castleviewuk.com/blog/index.php?forensic-fact-of-the-day---quotations---edmond-locard>.
- [5] S. Wilding. *Locard’s Exchange Principle*. Last accessed 04-04-2023. 2012. URL: <http://www.forensichandbook.com/locards-exchange-principle/>.
- [6] J. Siegel. *forensic science*. Last accessed 17-04-2022. June 2020. URL: <https://www.britannica.com/science/forensic-science>.

- [7] P. Kirk. “The Ontogeny of Criminalistics.” In: *Journal of Criminal Law, and Criminology, and Police Science* 54.2 (June 1963), pp. 235–238. URL: <http://www.jstor.org/stable/1141173>.
- [8] M. Andrew. “Defining a Process Model for Forensic Analysis of Digital Devices and Storage Media.” In: *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE’07)*. 2007, pp. 16–30. DOI: 10.1109/SADFE.2007.8.
- [9] K. Zatyko and J. Bay. “The Digital Forensics Cyber Exchange Principle.” In: *Forensic Magazine* 8.6 (2011).
- [10] A. Antwi-Boasiako and H. Venter. “A Model for Digital Evidence Admissibility Assessment.” In: *Advances in Digital Forensics XIII*. Ed. by G. Peterson and S. Sheno. Cham: Springer International Publishing, 2017, pp. 23–38. ISBN: 978-3-319-67208-3. DOI: 10.1007/978-3-319-67208-3_2.
- [11] I. Tsimperidis, C. Yucel, and V. Katos. “Age and Gender as Cyber Attribution Features in Keystroke Dynamic-Based User Classification Processes.” In: *Electronics* 10 (2021), pp. 1–14. DOI: 10.3390/electronics10070835.
- [12] S. Shute, R. Ko, and S. Chaisiri. “Attribution Using Keyboard Row Based Behavioural Biometrics for Handedness Recognition.” In: *2017 IEEE Trustcom/BigDataSE/ICSS*. 2017, pp. 1131–1138. DOI: 10.1109/Trustcom/BigDataSE/ICSS.2017.363.
- [13] U. N. O. on Drugs and C. (UNODC). *Digital evidence*. Last accessed 21-04-2022. Mar. 2019. URL: <https://www.unodc.org/e4j/en/cybercrime/module-4/key-issues/digital-evidence.html>.
- [14] K. Subramaniam. “Digital Forensics – As we know it today...” In: *IEEE India Info*. 13.4 (2018), pp. 89–92.
- [15] D. F. R. W. Attendees. *A Road Map for Digital Forensic Research*. DFRWS Technical Report DTR-T001-01-Final. Last accessed 21-12-2021. Digital Forensic Research Workshop (DFRWS), Nov. 2001. URL: https://dfrws.org/wp-content/uploads/2019/06/2001_USA_a_road_map_for_digital_forensic_research.pdf.
- [16] US-CERT. *Computer Forensics*. Last accessed 18-04-2022. 2008. URL: <https://www.cisa.gov/uscert/sites/default/files/publications/forensics.pdf>.
- [17] National Institute of Standards and Technology (NIST). *digital forensics*. Last accessed 18-04-2022. URL: https://csrc.nist.gov/glossary/term/digital_forensics.

E. Extended Introduction

- [18] DoD CIO. *Department of Defense DIRECTIVE — DoD Executive Agent (EA) for the DoD Cyber Crime Center (DC3)*. Tech. rep. 5505.13E. Last accessed 18-04-2022. Department of Defense (DoD), July 2017.
- [19] K. Kent, S. Chevalier, T. Grance, and H. Dang. *Guide to Integrating Forensic Techniques into Incident Response*. NIST Special Publication 800-86. Last accessed 18-04-2022. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology (NIST), Aug. 2006.
- [20] M. Herman, M. Iorga, A. Salim, R. Jackson, M. Hurst, R. Leo, R. Lee, N. Landreville, A. Mishra, Y. Wang, and R. Sardinas. *NIST Cloud Computing Forensic Science Challenges*. National Institute of Standards and Technology Interagency or Internal Report 8006. Last accessed 18-04-2022. National Institute of Standards and Technology (NIST), Aug. 2020.
- [21] N. Hassan. *Digital Forensics Basics: A Practical Guide Using Windows OS*. New York, New York, USA: APress, 2019. DOI: 10.1007/978-1-4842-3838-7.
- [22] M. Olivier. “On a scientific theory of digital forensics.” In: *Advances in Digital Forensics XII*. Ed. by G. Peterson and S. Sheno. Springer, 2016, pp. 3–24. DOI: 10.1007/978-3-319-46279-0_1.
- [23] X. Lin. “File Carving.” In: *Introductory Computer Forensics: A Hands-on Practical Approach*. Cham: Springer International Publishing, 2018. Chap. 9, pp. 211–233. ISBN: 978-3-030-00581-8. DOI: 10.1007/978-3-030-00581-8_9.
- [24] E. Casey. “What does ”forensically sound” really mean?” In: *Digital Investigation 4.2* (2007), pp. 49–50. DOI: 10.1016/j.diin.2007.05.001.
- [25] R. McKemmish. “When is Digital Evidence Forensically Sound?” In: *Advances in Digital Forensics IV*. Ed. by I. Ray and S. Sheno. Boston, MA: Springer US, 2008, pp. 3–15. ISBN: 978-0-387-84927-0. DOI: 10.1007/978-0-387-84927-0_1.
- [26] M. Pollitt. “Computer Forensics: an Approach to Evidence in Cyberspace.” In: *Proceedings of the 18th National Information Systems Security Conference*. 1995, pp. 487–491.
- [27] A. A. Thakar, K. Kumar, and B. Patel. “Next Generation Digital Forensic Investigation Model (NGDFIM) - Enhanced, Time Reducing and Comprehensive Framework.” In: *Journal of Physics: Conference Series* 1767.1 (Feb. 2021), p. 012054. DOI: 10.1088/1742-6596/1767/1/012054.
- [28] M. Reith, C. Carr, and G. Gunsch. “An Examination of Digital Forensic Models.” In: *International Journal of Digital Evidence* 1.3 (2002).

- [29] R. Montasari. “The Comprehensive Digital Forensic Investigation Process Model (CDFIPM) for Digital Forensic Practice.” PhD thesis. University of Derby, June 2016. URL: <https://repository.derby.ac.uk/item/9458q/the-comprehensive-digital-forensic-investigation-process-model-cdfipm-for-digital-forensic-practice>.
- [30] S. Raghavan. “Digital forensic research: current state of the art.” In: *CSI Transactions on ICT* 1.1 (Mar. 2013), pp. 91–114. DOI: 10.1007/s40012-012-0008-7.
- [31] E. Casey, M. Ferraro, and L. Nguyen. “Investigation Delayed Is Justice Denied: Proposals for Expediting Forensic Examinations of Digital Evidence*.” In: *Journal of Forensic Sciences* 54.6 (2009), pp. 1353–1364. ISSN: 1556-4029. DOI: 10.1111/j.1556-4029.2009.01150.x.
- [32] S. Garfinkel. “Digital forensics research: The next 10 years.” In: *Digital Investigation* 7 (2010). The Proceedings of the Tenth Annual DFRWS Conference, S64–S73. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.009.
- [33] R. Hranický, F. Breitingner, O. Ryšavý, J. Sheppard, F. Schaedler, H. Morgenstern, and S. Malik. “What do incident response practitioners need to know? A skillmap for the years ahead.” In: *Forensic Science International: Digital Investigation* 37 (2021), p. 301184. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2021.301184.
- [34] L. Luciano, I. Baggili, M. Topor, P. Casey, and F. Breitingner. “Digital Forensics in the Next Five Years.” In: *Proceedings of International Conference on Availability, Reliability and Security, Hamburg, Germany, August 27–30, 2018 (ARES 2018)*. 2018, Article 46. DOI: 10.1145/3230833.3232813.
- [35] A. Pal and N. Memon. “The evolution of file carving.” In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [36] R. Poisel and S. Tjoa. “A Comprehensive Literature Review of File Carving.” In: *2013 International Conference on Availability, Reliability and Security*. Sept. 2013, pp. 475–484. DOI: 10.1109/ARES.2013.62.
- [37] E. Casey, A. Nelson, and J. Hyde. “Standardization of file recovery classification and authentication.” In: *Digital Investigation* 31 (2019), p. 100873. ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.06.004.
- [38] A. Bhat, A. Likhite, S. Chavan, and L. Ragha. “File Fragment Classification using Content Based Analysis.” In: *ITM Web Conf.* 40 (2021), p. 03025. DOI: 10.1051/itmconf/20214003025.

- [39] S. Axelsson, K. Bajwa, and M. Srikanth. “File Fragment Analysis Using Normalized Compression Distance.” In: *Advances in Digital Forensics IX: 9th IFIP WG 11.9 International Conference on Digital Forensics, Orlando, FL, USA, January 28-30, 2013, Revised Selected Papers*. Ed. by G. Peterson and S. Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 171–182. ISBN: 978-3-642-41148-9. DOI: 10.1007/978-3-642-41148-9_12.
- [40] M. Karresand. “Completing the Picture — Fragments and Back Again.” Licentiate thesis. Linköping Institute of Technology, Linköping University, Sweden, May 2008.
- [41] S. Garfinkel and M. McCarrin. “Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb.” In: *Digital Investigation* 14.Supplement 1 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S95–S105. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.001.
- [42] F. Breiting, C. Rathgeb, and H. Baier. “An Efficient Similarity Digests Database Lookup - A Logarithmic Divide & Conquer Approach.” In: *Journal of Digital Forensics, Security and Law* 9.2 (2014), pp. 155–166. DOI: 10.15394/jdfs1.2014.1178.
- [43] F. Breiting and K. Petrov. “Reducing the Time Required for Hashing Operations.” In: *Advances in Digital Forensics IX - 9th IFIP WG 11.9 International Conference on Digital Forensics, Orlando, FL, USA, January 28-30, 2013, Revised Selected Papers*. Ed. by G. Peterson and S. Sheno. Vol. 410. IFIP Advances in Information and Communication Technology. Springer, 2013, pp. 101–117. DOI: 10.1007/978-3-642-41148-9_7.
- [44] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks. “Distinct Sector Hashes for Target File Detection.” In: *Computer* 45.12 (Dec. 2012), pp. 28–35. ISSN: 0018-9162. DOI: 10.1109/MC.2012.327.
- [45] K. Foster. “Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus.” MA thesis. Monterey, California, USA: Naval Postgraduate School, Sept. 2012.
- [46] S. Garfinkel, A. Nelson, D. White, and V. Roussev. “Using purpose-built functions and block hashes to enable small block and sub-file forensics.” In: *Digital Investigation* 7.Supplement (2010). The Proceedings of the Tenth Annual DFRWS Conference, S13–S23. ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.003.

- [47] S. Collange, Y. S. Dandass, M. Daumas, and D. Defour. “Using Graphics Processors for Parallelizing Hash-Based Data Carving.” In: *2009 42nd Hawaii International Conference on System Sciences*. Jan. 2009, pp. 1–10. DOI: 10.1109/HICSS.2009.494.
- [48] N. Ramli, S. Hisham, and G. Badshah. “Analysis of File Carving Approaches: A Literature Review.” In: *Advances in Cyber Security*. Ed. by N. Abdullah, S. Manickam, and M. Anbar. Singapore: Springer Singapore, 2021, pp. 277–287. ISBN: 978-981-16-8059-5. DOI: 10.1007/978-981-16-8059-5_16.
- [49] R. Poisel, M. Rybnicek, and S. Tjoa. “Taxonomy of Data Fragment Classification Techniques.” In: *Digital Forensics and Cyber Crime: Fifth International Conference, ICDF2C 2013, Moscow, Russia, September 26-27, 2013, Revised Selected Papers*. Ed. by P. Gladyshev, A. Marrington, and I. Baggili. Springer International Publishing, 2014, pp. 67–85. DOI: 10.1007/978-3-319-14289-0_6.
- [50] Y. Tang, J. Fang, K. Chow, S. Yiu, J. Xu, B. Feng, Q. Li, and Q. Han. “Recovery of heavily fragmented JPEG files.” In: *Digital Investigation* 18.Supplement (2016), S108–S117. ISSN: 1742-2876. DOI: 10.1016/j.diin.2016.04.016.
- [51] A. Pal, H. T. Sencar, and N. Memon. “Detecting file fragmentation point using sequential hypothesis testing.” In: *Digital Investigation* 5.Supplement (2008). The Proceedings of the Eighth Annual DFRWS Conference, S2–S13. ISSN: 1742-2876. DOI: 10.1016/j.diin.2008.05.015.
- [52] N. Memon and A. Pal. “Automated reassembly of file fragmented images using greedy algorithms.” In: *IEEE Transactions on Image Processing* 15.2 (2006), pp. 385–393. DOI: 10.1109/TIP.2005.863054.
- [53] S. Sari and K. Mohamad. “A Review of Graph Theoretic and Weightage Techniques in File Carving.” In: *The 2nd Joint International Conference on Emerging Computing Technology and Sports (JICETS) 2019*. Vol. 1529. 5. May 2020, p. 052011. DOI: 10.1088/1742-6596/1529/5/052011.
- [54] T. Owens and T. Padilla. “Digital sources and digital archives: historical evidence in the digital age.” In: *International Journal of Digital Humanities* 1.3 (July 2021), pp. 325–341. DOI: 10.1007/s42803-020-00028-7.
- [55] J. Durno. “Digital Archaeology and/or Forensics: Working with Floppy Disks from the 1980s.” In: *The Code4Lib Journal* 34 (2016). ISSN: 1940-5758.

E. Extended Introduction

- [56] D. Dietrich and F. Adelstein. “Archival science, digital forensics, and new media art.” In: *Digital Investigation* 14 (2015). The Proceedings of the Fifteenth Annual DFRWS Conference, S137–S145. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.004.
- [57] C. Lee, M. Kirschenbaum, A. Chassanoff, P. Olsen, and K. Woods. “BitCurator: Tools and Techniques for Digital Forensics in Collecting Institutions.” In: *D-Lib Magazine* 18.5/6 (2012). DOI: 10.1045/may2012-lee.

F. A Layman's Introduction

This chapter introduces the thesis and the mapping concept to anyone unfamiliar with the digital world. To explain the subject of the thesis the digital forensics area will conceptually be compared to the activities involved in running a public library. The work of a digital forensic investigator is represented by the information retrieval of the librarians when they help someone to find the answer to a question. The librarians need to be unbiased and select relevant and trustworthy sources, like the digital forensic investigator that needs to look for evidence regardless of whether it supports or contradicts his or her hypothesis. The digital forensics field of *file carving* is represented by the process of rescuing and preserving information from old, incomplete or otherwise damaged books and documents in the library.

Due to the influx of new books and documents the size of new library buildings (*storage media*) constantly increases. This also increases the time taken to handle all the books and documents in the new library. The time taken to retrieve a book or add a new is lowered by the manufacturers of library equipment and systems through slimmer shelves, better layout of the library storage and flooring with lower friction, but their work do not fully compensate for the ever-increasing amount of books.

To fill the shelves in the library the librarians follow different process descriptions (*allocation algorithms*) aiming at for example efficient retrieval of stored books, high speed identification of free space when adding a new book, or minimization of the waste of storage space, both in the shelves and also in the library as a whole. Many libraries use the same standard (*OS and file system*) to allocate space in their shelves. Currently the dominant standard (*Microsoft Windows OS and NTFS*) is used in almost all libraries in the world.

Sometimes the index of all the books of the library is lost for some reason (*file system corruption*). The order of the books then has to be restored in some way (*file carving*). Of special interest is to find the most valuable books (*evidence*) first, for example unique copies of rare books or the newest procurements (*user files*). The current methodology is to scan all the books linearly from the entrance of the library to the farthest shelf at the back to try to restore the original order and index used by the librarians. Hence the inherent structure of the library (*the file allocation pattern*) is ignored.

The proposal of a map of user data locations presented in this thesis is comparable to creating a map showing the book shelves with the highest lending frequency in a generic library (*partition*). The map is based on the predominant standard (*NTFS*) for

F. A Layman's Introduction

shelf allocation and its inherent structures introduced when adding books to the shelves of the library. The map is then used to direct the searches to the library shelves where the probability of finding the sought after book (*file*) is higher. The concept of using a map of a generic partition is also applicable to other areas, for example when archiving and rescuing digital information.

G. Extended Publications

Since the first publication by the author, a technical report written at FOI in 2001, a number of both peer-reviewed articles, theses and technical reports have been published. The publications of relevance to the PhD project are presented at the beginning of the thesis. The below lists contain a selection of publications unrelated to the PhD project.

The following MSc thesis has been defended by the author.

- M. Karresand. “A Proposed Taxonomy of Software Weapons.” MSc thesis. Department of Electrical Engineering, Institute of Technology, Linköping University, Sweden, Dec. 2002

In addition to the previously presented publications the following peer-reviewed articles have also been written by the author.

- M. Karresand. “A proposed taxonomy for IT weapons.” In: *Proceedings of the 7th Nordic Workshop on Secure IT Systems (Nordsec 2002)*. Ed. by S. Fischer Hübner and E. Jonsson. Nordsec 2002 Karlstad University. Nov. 2002, pp. 244–260
- M. Karresand. “Separating Trojan horses, viruses, and worms - A proposed taxonomy of software weapons.” In: *IEEE Systems, Man and Cybernetics Society Information Assurance Workshop*. 2003, pp. 127–134. DOI: 10.1109/SMCSIA.2003.1232411
- C. Duma, M. Karresand, N. Shahmehri, and G. Caronni. “A trust-aware, P2P-based overlay for intrusion detection.” In: *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA*. 2006, pp. 692–697. DOI: 10.1109/DEXA.2006.21
- H. Holm, M. Karresand, A. Vidström, and E. Westring. “A survey of industrial control system testbeds.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9417 (2015), pp. 11–26. DOI: 10.1007/978-3-319-26502-5_2

The author has also written a number of technical reports, which have been peer-reviewed at FOI.

G. Extended Publications

- M. Karresand. *TEBIT — Tekniskt Beskrivningsmodell för IT-vapen*. Tech. rep. FOI-R-0305-SE. Eng. title: TEBIT — Technical Description Model for IT Weapons. Totalförsvarets forskningsinstitut: Ledningssystemteknik, Aug. 2001
- M. Karresand. *Intrusion analysis in military networks — an introduction*. Tech. rep. FOI-R-1463-SE. Ledningssystemteknik, 2004
- M. Karresand. *Parametrar för intrångsanalys*. Tech. rep. FOI-R-1831-SE. Eng. title: Parameters for Intrusion Analysis. Command and Control Systems, 2005
- M. Karresand. *Pålitliga IT-plattformar — Kopplingar mellan Försvarens behov och litteraturen*. Tech. rep. FOI-R-3903-SE. Eng. title: Trustworthy IT Platforms — Connections between needs of the Swedish Defence Forces and the literature. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2014

The following list presents a selection of co-authored technical reports, which have been peer-reviewed at FOI.

- M. Karresand, M. Persson, and D. Lindahl. *Scenarion och trender i framtida informationskrigföring ur ett tekniskt perspektiv*. Tech. rep. FOI-R-1283-SE. Eng. title: Scenarios and trends for future information warfare from a technical viewpoint. Command and Control Systems, 2004
- M. Karresand and D. Nordqvist. *Utvärdering av Autoclass C*. tech. rep. FOI-R-1484-SE. Eng. title: Evaluation of Autoclass C. Command and Control Systems, 2005
- I. Rodhe and M. Karresand. *Overview of formal methods in software engineering*. Tech. rep. FOI-R-4156-SE. Information and Aeronautical Systems, 2015
- D. Eidenskog, M. Karresand, U. Sterner, and Å. Waern. *Litteraturstudie om trafikskydd*. Tech. rep. FOI-R-4255-SE. Eng. title: Transmission Security — a Literature Survey. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2016
- I. Rodhe and M. Karresand. *Verktyg för att åstadkomma pålitlig programvara*. Tech. rep. FOI-R-4290-SE. Eng. title: Tools to accomplish trustworthy software. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2016
- H. Karlzén, M. Karresand, and Å. Waern. *Informationsinfrastruktur för anpassningsbara system*. Tech. rep. FOI-R-4353-SE. Eng. title: Information infrastructures for situationally adapted systems. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2017

- A. Gudmundson Hunstad and M. Karresand. *Monitorerings- och övervakningssystem — En kategorisering och översikt inom IIS*. tech. rep. FOI-R-4420-SE. Eng. title: Monitoring and policing systems — A categorization and survey within ICS. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2017
- D. Eidenskog and M. Karresand. *Risker med virtualisering av IT-system*. Tech. rep. FOI-R-4448-SE. Eng. title: Risks When Using Virtualization of IT Systems. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2017
- A. Gudmundson Hunstad and M. Karresand. *Molntjänster inom industriella informations- och styrsystem*. Tech. rep. FOI-R-4597-SE. Eng. title: Cloud services for ICS. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2018
- C. Valassi and M. Karresand. *NCS3 — Komponenter på avstånd. Säkerhetsbeaktanden för direkt adresserbara trådlöst nätverksanslutna komponenter i industriella informations- och styrsystem*. Tech. rep. FOI-R-4757-SE. Eng. title: Security Implications for Wireless Components in ICS. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2019
- C. Valassi and M. Karresand. *IT-angrepp mot industriella informations- och styrsystem*. Tech. rep. FOI-R-4929-SE. Eng. title: IT Attacks Against Industrial Control Systems — Early signs of an attack. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2020
- C. Valassi and M. Karresand. *Cyberfysiska sårbarheter i tunga fordon*. Tech. rep. FOI-R-5067-SE. Eng. title: Cyber-physical Vulnerabilities in Heavy Vehicles. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR), 2020

Further publications written or co-authored by the author can be found by using the search function at <https://www.foi.se/en/foi/reports.html>, Google or any other web search engine.

ISBN 978-82-326-7046-8 (printed ver.)
ISBN 978-82-326-7045-1 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)



NTNU

Norwegian University of
Science and Technology