

CRANFIELD UNIVERSITY

SARAH LOUISE ANGELA MORRIS

AN INVESTIGATION INTO THE IDENTIFICATION,
RECONSTRUCTION, AND EVIDENTIAL VALUE OF THUMBNAIL
CACHE FILE FRAGMENTS IN UNALLOCATED SPACE

Defence and Security

PhD

Academic Year: 2013

Supervisor: Professor Howard Chivers
April 2013

CRANFIELD UNIVERSITY

Defence and Security

PhD

Academic Year 2008- 2013

SARAH LOUISE ANGELA MORRIS

An Investigation into the identification, reconstruction, and evidential
value of thumbnail cache file fragments in unallocated space

Supervisor: Professor Howard Chivers
April 2013

© Cranfield University 2013. All rights reserved. No part of this
publication may be reproduced without the written permission of the
copyright owner.

ABSTRACT

This thesis establishes the evidential value of thumbnail cache file fragments identified in unallocated space. A set of criteria to evaluate the evidential value of thumbnail cache artefacts were created by researching the evidential constraints present in Forensic Computing. The criteria were used to evaluate the evidential value of live system thumbnail caches and thumbnail cache file fragments identified in unallocated space. Thumbnail caches can contain visual thumbnails and associated metadata which may be useful to an analyst during an investigation; the information stored in the cache may provide information on the contents of files and any user or system behaviour which interacted with the file. There is a standard definition of the purpose of a thumbnail cache, but not the structure or implementation; this research has shown that this has led to some thumbnail caches storing a variety of other artefacts such as network place names.

The growing interest in privacy and security has led to an increase in user's attempting to remove evidence of their activities; information removed by the user may still be available in unallocated space. This research adapted popular methods for the identification of contiguous files to enable the identification of single cluster sized fragments in Windows 7, Ubuntu, and Kubuntu. Of the four methods tested, none were able to identify each of the classifications with no false positive results; this result led to the creation of a new approach which improved the identification of thumbnail cache file fragments.

After the identification phase, further research was conducted into the reassembly of file fragments; this reassembly was based solely on the potential thumbnail cache file fragments and structural and syntactical information. In both the identification and reassembly phases of this research image only file fragments proved the most challenging resulting in a potential area of continued future research. Finally this research compared the evidential value of live system thumbnail caches with identified and reassembled fragments. It was determined that both types of thumbnail cache artefacts can provide unique information which may assist with a digital investigation.

This research has produced a set of criteria for determining the evidential value of thumbnail cache artefacts; it has also identified the structure and related user and system behaviour of popular operating system thumbnail cache implementations. This research has also adapted contiguous file identification techniques to single fragment identification and has developed an improved method for thumbnail cache file fragment identification. Finally this research has produced a proof of concept software tool for the automated identification and reassembly of thumbnail cache file fragments.

Keywords:

File carving, evidential value, contextual analysis, fragment identification, fragment reassembly, forensic computing, and cybercrime

ACKNOWLEDGEMENTS

“And above all, watch with glittering eyes the whole world around you because the greatest secrets are always hidden in the most unlikely places. Those who don't believe in magic will never find it.”

The Minpins, Roald Dahl,

Firstly and most importantly I would like to thank my supervisor Professor Howard Chivers for providing support, sanity and guidance throughout what has been an amazing journey. I'd also like to thank all the members of my thesis committee (Professor Peter Hill, Dr Mike Edwards, Dr Phil Noble and Professor Tony Sammes) for their support.

The Centre for Forensic Computing has provided me with so many opportunities to learn and develop over the past few years that I would like to thank every member of the team for their kindness. In particular thank you to Dr Chris Hargreaves for always encouraging me to try harder, providing a sounding board and being a good friend.

Thank you to my grandparents: Betty and the late Eric Morris who have provided so much love and support throughout the years, also Reginald and Pauline Dixon. Thank you to my Uncle Tony Dixon, who has always been supportive and there when needed.

Thank you also to my parents who have patiently listened to my career aspirations for the last two decades and have provided support as I continue up the academic ladder. Thank you to my sister Poppy who being practically perfect is always on hand to offer her wisdom, love and support.

Finally I'd like to thank my wonderful partner Gavin for putting up with me through the highs and lows.

TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDGEMENTS.....	iii
LIST OF FIGURES.....	xi
LIST OF TABLES	xiii
1 Introduction.....	1
1.1 Introduction	1
1.2 Justification	2
1.3 Aim.....	3
1.4 Hypothesis	3
1.5 Methodology	4
1.6 Thesis Outline	5
1.7 Contributions.....	9
2 Related Research.....	11
2.1 Introduction	11
2.2 Forensic Computing.....	11
2.2.1 Methodologies.....	14
2.2.2 Tools used in Forensic Computing.....	17
2.2.3 Current Opportunities	20
2.3 Data and information.....	22
2.3.1 Storing information	22
2.3.2 Metadata	25
2.3.3 Event Reconstruction	27
2.3.4 Corrupting data	30
2.4 Thumbnails and the Thumbnail Cache	32
2.4.1 Visual thumbnails and their applications in computing	33
2.4.2 Operating system Thumbnail Caches	34
2.4.3 Forensic significance.....	36
2.5 Conclusion	38
3 Research Objectives, Considerations and Tools.....	41
3.1 Introduction	41
3.2 Objectives	41
3.3 Potential Considerations	43
3.4 Tools	44
3.5 Conclusion	45
4 Defining Evidential Value.....	47
4.1 Introduction	47
4.2 Evidential Constraints	47
4.2.1 Legal Constraints	48
4.2.2 Evidential Constraints.....	50
4.2.3 Reliability Constraints.....	51

4.2.4 Procedural Constraints.....	53
4.2.5 Human Behavioural Constraints.....	55
4.3 Criteria for evaluating the evidential value of thumbnail cache artefacts.....	58
4.3.1 Provenance of Artefacts.....	58
4.3.2 Interpretation of Results	59
4.4 Process Criteria	60
4.4.1 Provenance of Artefacts.....	60
4.4.2 Legal and ethical constraints.....	61
4.5 Conclusion	61
5 The Structure and Behaviour of the Operating System Thumbnail Cache Implementations used in Ubuntu and Kubuntu.....	63
5.1 Introduction	63
5.2 Methodology	64
5.3 Thumbnail caches.....	65
5.4 Default Installations.....	67
5.5 Identifying the structure.....	68
5.5.1 The thumbnail cache directory structure	68
5.5.2 The structure of a thumbnail Cache subrecord	72
5.6 Identifying the behaviour.....	80
5.6.1 The creation of subrecords.....	80
5.6.2 The modification of subrecords	87
5.6.3 Fragmentation of the thumbnail cache	89
5.7 Identifying traces left outside the thumbnail cache	89
5.8 Alteration of thumbnail cache artefacts.....	90
5.8.1 Metadata	90
5.8.2 Visual thumbnail.....	91
5.8.3 Original source file	92
5.9 Forming a relationship between the thumbnail subrecords and the source files.....	92
5.9.1 Metadata	93
5.9.2 Images	93
5.9.3 MD5 hashes	94
5.9.4 Event timeline.....	94
5.10 Audio and Media thumbnails.....	96
5.10.1 Audio thumbnails.....	96
5.10.2 Media thumbnails	96
5.11 Discussion	97
5.11.1 Provenance of artefacts	97
5.11.2 Interpretation of results.....	100
5.12 Conclusion	102

6 The structure and behaviour of the Windows 7 Operating System	
Thumbnail Cache	105
6.1 Introduction	105
6.2 Methodology	106
6.3 Default installation.....	108
6.3.1 Initial Searches.....	108
6.3.2 Initial state of the thumbnail cache	109
6.4 Identifying the structure.....	109
6.4.1 Methodology.....	110
6.4.2 The thumbnail cache directory structure	113
6.4.3 thumbcache_idx	114
6.4.4 thumbcache_32, 96, 256 and 1024.....	116
6.4.5 thumbcache_sr.....	121
6.5 Identifying the behaviour	122
6.5.1 The creation of records and subrecords.....	122
6.5.2 The modification of records and subrecords	130
6.5.3 The deletion of records and subrecords	133
6.5.4 Fragmentation of the thumbnail cache	134
6.6 Identifying traces left outside the thumbnail cache	134
6.6.1 Windows.edb.....	136
6.6.2 Registry	136
6.6.3 Shortcuts	138
6.6.4 External Media	139
6.6.5 Encrypted Containers.....	140
6.7 Alteration of thumbnail cache artefacts	141
6.7.1 Metadata	141
6.7.2 Visual thumbnail.....	142
6.7.3 Original source file	143
6.8 Forming a relationship between the thumbnail subrecords and the source files.....	143
6.8.1 Visual Thumbnail.....	144
6.8.2 Metadata	145
6.8.3 Thumbnail cache ID	146
6.8.4 Event timeline.....	147
6.9 Audio and Media thumbnails.....	148
6.9.1 Audio thumbnails.....	148
6.9.2 Media thumbnails	150
6.10 Discussion	151
6.10.1 Provenance of Artefacts	152
6.10.2 Interpretation of Results	154
6.11 Conclusion	155
7 Identification of thumbnail cache fragments	157

7.1 Introduction	157
7.2 Traditional Data Carving	158
7.2.1 <i>Brute Force</i> Approach	159
7.2.2 <i>Structural and Syntactical</i> Analysis	160
7.3 Data Carving Research.....	161
7.3.1 Statistical Methods	162
7.3.2 <i>Neural Networks</i>	164
7.3.3 Recovering deleted information from other operating systems	165
7.4 Methodology	165
7.5 Information to be carved	167
7.5.1 Corpus Generation	167
7.5.2 File Fragment Classification	169
7.5.3 Preparing the File Fragments.....	170
7.5.4 Distribution of File Types.....	172
7.5.5 Training and Testing Sets	173
7.6 <i>Brute Force</i> Approach.....	175
7.7 <i>Structural and Syntactical</i> Approach	177
7.7.1 H1:.....	177
7.7.2 H2.....	180
7.7.3 H3.....	181
7.7.4 H4.....	182
7.7.5 H5.....	183
7.7.6 H6.....	183
7.8 Statistical Approach	183
7.8.1 Construction	184
7.8.2 Using <i>Bayesian Networks</i> for Classification	185
7.9 <i>Neural Network</i> Approach.....	187
7.10 Results.....	188
7.10.1 Method 1: <i>Brute Force</i> approach.....	188
7.10.2 Method 2: <i>Structural and Syntactical</i> approach.....	190
7.10.3 Method 3: Statistical approach	192
7.10.4 Method 4: <i>Neural Network</i> approach.....	194
7.11 Comparison of Classification Methods.....	196
7.12 Discussion	198
7.13 Chapter Conclusion	201
8 The creation of a Hybrid Identification Approach.....	203
8.1 Introduction	203
8.2 Problem Definition.....	204
8.3 Methodology	207
8.4 JPEG Identification Research	207
8.5 Visual Thumbnail JPEG File Fragment Identification	209
8.6 A hybrid approach for thumbnail cache identification.....	215

8.6.1 Stage 1: Preliminary Checks	217
8.6.2 Stage 2: H1 Validation Checks.....	218
8.6.3 Stage 3: H2 Validation Checks.....	218
8.6.4 Stage 4: H3 Validation Checks.....	219
8.6.5 Stage 5: H4 Validation Checks.....	219
8.6.6 Stage 6: H5 Validation Checks.....	221
8.6.7 Logs	221
8.6.8 Accepted Fragments	221
8.7 Results.....	222
8.8 Discussion	226
8.9 Chapter Conclusion	229
9 Thumbnail cache fragment reassembly.....	231
9.1 Introduction	231
9.2 Image Reassembly	232
9.3 Methodology	233
9.3.1 Potential clashes and missing fragments	234
9.3.2 Logging	235
9.3.3 Data sets.....	235
9.3.4 Training and testing.....	236
9.3.5 A general reassembly approach.....	236
9.4 Thumbcache_idx.....	238
9.4.1 Thumbcache_idx file fragment categories.....	240
9.4.2 Thumbcache_idx file fragment reassembly	242
9.5 Thumbcache_32, 96, 1024	246
9.5.1 Thumbcache_32, 96, 1024 file fragment categories	246
9.5.2 Thumbcache_32, 96, 1024 file fragment reassembly.....	248
9.6 Thumbcache_256	255
9.6.1 Thumbcache_256 file fragment categories	256
9.6.2 Thumbcache_256 file fragment reassembly.....	256
9.7 Linux Thumbnail Records	257
9.7.1 Linux thumbnail file fragment categories	258
9.7.2 Linux thumbnail file fragment reassembly	259
9.8 Results.....	260
9.9 Artefacts from fragments identified in unallocated space.....	262
9.10 Discussion	263
9.11 Conclusion	266
10 Establishing the evidential value of thumbnail cache file fragments identified in unallocated space.	267
10.1 Introduction	267
10.2 Replication, Corroboration and Clear Documentation	267
10.3 Justifiable Method, Understandable Consequences and Scientific Method.....	269

10.4 Typical Operation, Behaviour, Relationships and Maximising Corroboration.....	271
10.5 Maximum Amount of Data.....	273
10.6 Automation.....	274
10.7 Discussion.....	275
10.8 Conclusion	277
11 Discussion.....	279
11.1 Introduction	279
11.3 Structure and Behaviour of Complete Thumbnail Caches	280
11.4 Thumbnail Cache File Fragment Identification.....	282
11.5 Thumbnail Cache File Fragment Reassembly	283
11.6 Image Format Problems	285
11.7 Generalising the approach to file carving.....	287
11.8 Conclusion	288
12 Conclusions and future work	291
12.1 Research Conclusions	291
12.2 Contributions of Research	291
12.3 Future Work	292
12.4 Summary	293
REFERENCES.....	297
APPENDICES	309
Appendix A Thumbnail Cache File Structures	311
Appendix B Bayesian probability tables	315
Appendix C Peer Reviewed Publications.....	325

LIST OF FIGURES

Figure 1-1:A breakdown of the proposed methodology for this research	4
Figure 5-1: Ubuntu 10.10 and the Nautilus File Manager	66
Figure 5-2: Directory structure for experimental test files	68
Figure 5-3: The structure of thumbnail caches in Ubuntu and Kubuntu.....	69
Figure 5-4: A figure showing the thumbnail cache subrecords which make up the complete record for the file z2nxaujt.jpg.....	70
Figure 5-5: The directory 'Downloads' being viewed in the file browser within Kubuntu.....	85
Figure 6-1: Directory structure for experiment test files.....	110
Figure 6-2: The centralised thumbnail cache structure.....	114
Figure 6-3: Thumbcache_idx record structure.....	115
Figure 6-4: Subrecord header	117
Figure 6-5: The creation of subrecords in thumbcache_1024.db	124
Figure 6-6: Introduction of a save thumbnail option.....	125
Figure 6-7: The relationship between a file and a record in the thumbnail cache	128
Figure 6-8: A screenshot from a test to see if records were created for files which had not been viewed	129
Figure 6-9:Searching for a GUID.....	137
Figure 6-10: Identifying the Default Icon.....	138
Figure 6-11: An example contents from an imageres.dll	139
Figure 6-12: A Shortcut Icon in Windows Explorer	139
Figure 6-13: Restore points for the directory Explorer	146
Figure 6-14: Audio Thumbnails	149
Figure 6-15: The use of audio thumbnails	149
Figure 6-16: The structure of a media thumbnail.....	150
Figure 6-17: The use of media thumbnails	151
Figure 7-1: Traditional file carving from clusters of unallocated space on the storage media	159
Figure 7-2: Classifications for thumbcache_IDX fragments.....	178

Figure 7-3: Bayesian network for H1 and H2 classification	186
Figure 7-4: Bayesian Network for H3, H4 and H5 classification	186
Figure 7-5: Bayesian Network for H6 classification	187
Figure 8-1: Jigsaw piece classification	206
Figure 8-2: Single byte frequencies for 4 visual thumbnails	211
Figure 8-3: A summary of the hybrid approach	216
Figure 9-1: Flow Chart for File Fragment Reassembly.....	239
Figure 9-2: Thumbcache_IDX Classifications.....	241
Figure 9-3: Thumbcache_IDX reassembly	245
Figure 9-4: Reassembling fragments with a metadata join.....	249
Figure 9-5: Reassembling an image fragment.....	250
Figure 9-6: Calculations for image reassembly	251
Figure 9-7: Reassembling an image by trying every permutation.....	253
Figure 9-8: Fragment reassembly using file structure	257
Figure 11-1: A breakdown of aim of this research into smaller research problems	280

LIST OF TABLES

Table 5-1: A summary of the user activity which may lead to subrecords being present in the thumbnail cache	99
Table 6-1: A summary of the user activity which may lead to subrecords being present in the thumbnail cache	135
Table 7-1: Breakdown of file fragment types per classification.....	172
Table 7-2: Breakdown of potentially similar file fragment types in H6	173
Table 7-3: A list of thumbnail cache keywords	176
Table 7-4: Method 1 Results for the Training Set (Half of Data Set 1)	188
Table 7-5: Method 1 Results for the 1 st Testing Set (Half of Data Set 1).....	189
Table 7-6: Method 1 Results for the 2 nd Testing Set (Data Set 2)	189
Table 7-7: Method 1 Results for the unknown data set (Data Set 3)	190
Table 7-8: Method 2 Results for the Training Set (Half of Data Set 1)	190
Table 7-9: Method 2 Results for the 1 st Testing Set (Half of Data Set 1).....	191
Table 7-10: Method 2 Results for the 2 nd Testing Set (Data Set 2)	191
Table 7-11: Method 2 Results for the unknown data set (Data Set 3)	192
Table 7-12: Method 3 Results for the Training Set (Half of Data Set 1)	192
Table 7-13: Method 3 Results for the 1 st Testing Set (Half of Data Set 1).....	193
Table 7-14: Method 3 Results for the 2 nd Testing Set (Data Set 2)	193
Table 7-15: Method 3 Results for the unknown data set (Data Set 3)	194
Table 7-16: Method 4 Results for the Training Set (Half of Data Set 1)	194
Table 7-17: Method 4 Results for the 1 st Testing Set (Half of Data Set 1).....	195
Table 7-18: Method 4 Results for the 2 nd Testing Set (Data Set 2)	195
Table 7-19: Method 4 Results for the unknown data set (Data Set 3)	196
Table 7-20: Percentage Success and False Positive rate for each method for Data Set 2	197
Table 8-1: JPEG sequence markers found in Visual thumbnails.....	213
Table 8-2: Results of visual thumbnail JPEG method	214
Table 8-3: Percentage success and false positive rates for the H3 method...	214
Table 8-4: Method 5 Results for the Training Set (Half of Data Set 1)	223

Table 8-5: Method 5 Results for the 1 st Testing Set (Half of Data Set 1).....	224
Table 8-6: Method 5 Results for the 2 nd Testing Set (Data Set 2)	225
Table 8-7: Method 5 Results for the unknown data set (Data Set 3)	225
Table 8-8: Percentage Success and False Positive rates for the hybrid approach and the commercial recovery tool Blade for Data Set 2	228
Table 9-1: Results from reassembly of data set A.....	261
Table 9-2: Results from reassembly of data set B.....	261
Table 9-3: Results from reassembly of data set C.....	262

1 Introduction

1.1 Introduction

In computing, the term *thumbnail* is used to define a small representation of a larger file. Thumbnails were originally created for visual data, such as an image, but they have developed to include audio and multimedia formats. Thumbnails are used in a variety of applications and operating systems to assist with the identification of various source files. For example, when browsing directories thumbnails can provide greater detail on the content of the file than the file name alone. Visual thumbnails are also widely used by websites to assist a user in browsing the content.

Whilst the use of visual thumbnails can enhance the usability of applications, they can be resource intensive to generate, particularly if a directory contains a large number of files which require the generation of thumbnails. A thumbnail cache allows visual thumbnails and their associated metadata to be stored. The cache prevents unnecessary re-rendering and therefore reduces the use of system resources. Such caches are of particular significance to forensic examiners as they can contain: metadata, a visual thumbnail of the original file, and potentially can provide information about the user's activity.

There is no standard implementation of a thumbnail cache, therefore software applications and operating systems each use the basic concept in their own way. This can be a source of confusion for forensic examiners as thumbnail caches which look structurally similar may not represent the same activity, and may therefore have a different evidential significance. In addition to the expected visual thumbnails and associated metadata, thumbnail caches may hold other unexpected pieces of information; such as network places and drive allocations [**Chapter 6**].

1.2 Justification

The thumbnail cache can provide an analyst with information about potential system and user activity. This is highlighted in cases such as United States V. Vosburgh [2010] and R v Ross Warwick Porter [2006] where the information contained within the thumbnail cache and the user behaviour that led to its presence on the system are important evidential concerns (See **Chapter 4**). An analyst interprets each artefact in the context of the system: by reconstructing the behaviour of the system it is possible to identify how events occurred and prove or disprove the hypothesis relating to the case. This process of contextual analysis suggests that the Forensic Computing community needs to be aware of both the structure and behaviour of an artefact in order to interpret its significance in a case. The thumbnail cache does not have a generic implementation and therefore understanding the unique structure and behaviour of each thumbnail cache can assist with interpreting the behaviour it represents.

The growing interest in privacy and increased understanding of technology has led to an increase in user awareness of system files. For example, information about thumbnail cache can be found on sites such as YouTube [2013], Facebook [2013] and Twitter [2013], which highlights the growing interest amongst users in understanding system files. This has led to users attempting to disguise their behaviour by deleting system and user files which contain potential evidence of their activities. Therefore it is becoming increasingly important to look for crucial evidential artefacts that may be present in unallocated space.

Files in unallocated space may become partially or completely overwritten, which may leave only fragments of the file on the system. Most file carving techniques focus on recovery of complete contiguous files; little research has been conducted into retrieving and reassembling individual file fragments. One

potential difficulty is the identification of a file fragment with the small amount of information they contain. The potential importance of thumbnail cache files and their varied structure suggests that research into retrieving and understanding the evidential value of these fragments could assist in providing additional artefacts in a case. These additional artefacts may alter or corroborate an existing event reconstruction and help to ensure a suitable conclusion is reached.

1.3 Aim

Understanding and recovering the data contained within the thumbnail cache is a relevant problem in the forensic computing community. As the level of understanding of a general computer user increases it becomes more likely that a user who wishes to conceal evidence of his activity will attempt to remove relevant data from his system. If the user simply deletes relevant artefacts the data will remain resident on the storage device until the clusters they occupy are reassigned to new data; therefore it is likely that clusters containing relevant artefacts may exist on the system in unallocated space. The aim of this research is to evaluate the evidential value of thumbnail cache file fragments identified in unallocated space.

1.4 Hypothesis

Operating system thumbnail caches can provide important sources of evidence in a forensic computing investigation. Therefore the research hypothesis is

Thumbnail cache file fragments identified in unallocated space contain artefacts of evidential value that may provide potential evidence when the thumbnails have been deleted from the live thumbnail cache of the system.

1.5 Methodology

This research aims to evaluate the evidential value of thumbnail cache file fragments identified in unallocated space. In order to meet the research aims the problem is broken down into smaller research areas; *figure 1.1* provides a breakdown of the areas of research. This section identifies the general methodology for this thesis; each section of this research has specific methodologies which are described throughout this thesis. First, a review of related literature is conducted to assist with the development of a methodology for this research [**Chapters 2 and 3**]. It will then be necessary to define the term “evidential value”; this definition can be used to develop a set of criteria which can be used to evaluate the artefacts identified during this research [**Chapter 4**]; the definition can then be used throughout the remainder of the research to evaluate the results of the various stages.

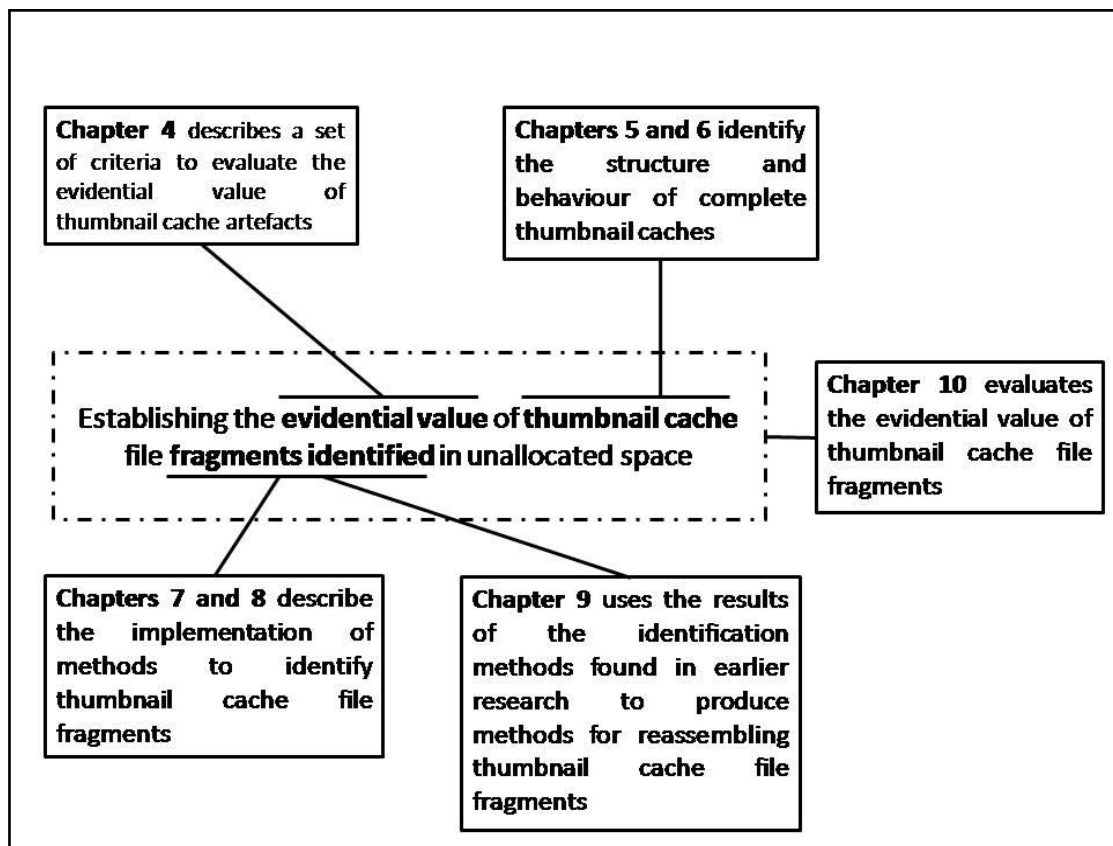


Figure 1-1: A breakdown of the proposed methodology for this research

This is followed by a structural and behavioural study of common operating system thumbnail caches [**Chapters 5 and 6**]. This study provides information on the evidential value of a complete thumbnail cache in the context of a complete system. The structure and syntax identified during the initial study provide detailed information to assist with implementing file fragment identification techniques for the different thumbnail cache types [**Chapters 7 and 8**]. After the identification phase methods are implemented to assist with reassembling the identified fragments; artefacts are then extracted from the reassembled fragments to identify the range of information available [**Chapter 9**].

The results of the structural and behavioural study are compared with the evidential value of the thumbnail cache fragments identified and reassembled from unallocated space [**Chapter 10**]. The comparison of evidential value identifies which types of artefacts are likely to be found in each location and their potential evidential value during an analysis. This is followed by a critical evaluation of the methodologies employed during this research [**Chapter 11**]; finally this thesis concludes with the research conclusions and potential future work [**Chapter 12**].

1.6 Thesis Outline

This section sets out the structure of the remainder of this thesis.

Chapter 2 establishes a working definition for the term Forensic Computing and identifies the processes involved in this field. Forensic Computing, or Digital Forensics as it is sometimes referred to, can be defined as the process of analysing a digital device with the resulting analysis being potentially produced in a Court of Law as evidence. This Chapter also describes the most common

implementations of operating system thumbnail caches and discusses available literature on their implementations.

Chapter 3 highlights the requirement to breakdown the aim of this research into smaller focused areas in order to determine the validity of the hypothesis; the objectives of the components of this research are described. This is followed by an overview of the potential considerations for this research. One of the most significant considerations for this research is the requirement to evaluate methodologies and results by appropriate forensic standards. Finally, the software tools used during this research are described.

Chapter 4 identifies the criteria which determine the evidential value of an artefact and establishes how the criteria can be applied to evidence relating to a thumbnail cache. This chapter explores the term evidence and evaluates how evidential value is assessed in court; the criteria established for assessing evidential value in general are then applied specifically to the thumbnail cache to create a specification which can be used to evaluate the results of this research.

Chapter 5 describes the structure and behaviour of the thumbnail cache implementations used in the Linux distributions Ubuntu and Kubuntu. The structural study highlights that whilst both thumbnail caches were implemented based on the same open desktop specification there are in fact small but important differences. The behavioural study identifies common user interactions with the system that would result in changes to data within the thumbnail cache; it also describes the changes that occur on the remainder of the system. The changes to the data can be used to identify possible user activity which has occurred on the system.

Chapter 6 describes the structure and behaviour of the windows 7 operating system thumbnail cache. Within this research: audio, visual, and media thumbnails were examined; interestingly it was determined that audio and media thumbnails do not generally store data within the thumbnail cache. The research also found situations where Windows 7 made use of *Thumbs.db* files; it was also noted that the thumbnail cache stores other kinds of data including allocated drive letters and network place names. The thumbnail cache structure in Windows 7 looks similar to the implementation used in Windows Vista, however it was noted there were key structural changes which included the removal of dates and times.

Chapter 7 identifies the most common file carving techniques; it describes the implementation of four common approaches adapted for single fragment thumbnail cache identification. Each technique is evaluated using the thumbnail cache structures identified in **Chapters 5 and 6**. This research divided the identification of thumbnail cache fragments into four classifications; two classifications achieved 100% accuracy with no false positives in the test set. The implemented methods achieved poor results on identifying image-only fragments from the thumbnail cache and therefore this required further separate research.

Chapter 8 describes a hybrid approach created for the identification of thumbnail cache file fragments. As a result of the evaluation of the four methods implemented in **Chapter 7**, it was noted that no single method managed to identify all the thumbnail cache file fragments without any false positives. A new hybrid approach was implemented to improve the identification statistics. This chapter also further examines the problem of identifying image-only thumbnail cache file fragments; this research focuses on JPEG file fragments. The images in the thumbnail caches studied during this research are stored in standard image formats; therefore a fragment which contains only image data from the thumbnail cache has the same structure as a standard image fragment. This

chapter evaluates possible image fragment identification methods; this research enables an approach to be created to improve the accuracy of identifying potential image-only thumbnail cache file fragments. The aim of this chapter is to maximise the number of file fragments available for the reassembly process; this led to a substantial number of standard image fragments being falsely identified as thumbnail cache fragments. It was determined that having a large number of false positives for the image category was preferable as it makes successful reassembly more likely.

Chapter 9 describes the research conducted into methods of reassembling the thumbnail cache file fragments identified in **Chapters 7 and 8**. A general reassembly model was constructed based upon the methodology for reassembling a jigsaw puzzle. Each file type used in the thumbnail caches' examined in **Chapters 5 and 6** was then separately considered to assist with the reassembly and validation of the file fragment sequences produced. This Chapter concludes with a description of the artefacts identified from data set 3 which contained data taken from unallocated space using the implemented identification and reassembly approaches.

Chapter 10 evaluates the evidential value of both complete and deleted fragments of operating system thumbnail caches. The evidential value of the thumbnail caches identified in **Chapters 5 and 6**, as well as the fragments recovered and reassembled at the end of **Chapter 9**, are evaluated against the criteria set in **Chapter 4**. This chapter highlights the fact that some artefacts, such as thumbnail caches from previous windows installations, can only be recovered from unallocated space. It also highlights the increased evidential value of using complete and deleted fragments of thumbnail caches together to increase understanding of the system's activity.

Chapter 11 evaluates the methodologies implemented during this research; it provides a critical overview of the research which has been undertaken and highlights any areas which could be improved. Each section of this research is individually evaluated; this is followed by a critical evaluation of the research as a whole. One of the main strengths of this research is the comparison of multiple file carving techniques to ensure a high number of thumbnail cache fragments were identified. A question remaining open from this research is the further evaluation into the identification of fragments which only contained image data; however in **Chapter 12** this is considered as an area for further potential work.

Chapter 12 provides a summary of the contributions and conclusions reached in this research; this is followed by an overview of potential future research.

1.7 Contributions

This research establishes a working definition for the term evidential value and creates a set of criteria to establish the evidential value of artefacts relating to operating system thumbnail caches. The criteria are used to compare the evidential value of complete thumbnail caches, which are part of the current operating system, to thumbnail cache file fragments identified and reassembled from unallocated space. This research identifies that there are potentially useful thumbnail cache artefacts which can only be found in unallocated space, such as thumbnail caches from previous operating system installations.

This research also describes both the structure and behaviour of popular implementations of operating system thumbnail caches. Each operating system implements the thumbnail cache differently; therefore an analyst needs to be familiar with the differences between implementations and their effects in order to extract meaningful artefacts [Morris, 2010; Morris, 2011a]. Research is also

conducted into establishing relationships between the thumbnail cache and other artefacts on a system, allowing the artefacts relating to the thumbnail cache to be put into context of other system events [Morris, 2011b].

During this research current file carving approaches are compared and evaluated on their ability to identify thumbnail cache file fragments [Morris, 2013]; this evaluation led to the creation of a new hybrid approach specifically refined for each type of thumbnail cache. The hybrid approach performed significantly better at identifying the various thumbnail cache file fragments than any single method.

This research also developed reassembly methods for reassembling thumbnail cache file fragments; these methods can reassemble both complete and partial files. The reassembly methods used the unique structural and syntactical characteristics of each file type to create an accurate approach to reassembling the fragments; once the fragments are reassembled it is possible to extract the various artefacts within the file and then add them to a contextual analysis of the system.

2 Related Research

2.1 Introduction

This chapter identifies potential opportunities for research within forensic computing; it also reviews literature relating to the evaluation of the evidential value of thumbnail cache file fragments in unallocated space. Research relating to data carving and reassembly are discussed in **Chapters 7,8 and 9**. In this Chapter, **Section 2.2** defines the term forensic computing, and describes the field's development; this leads on to a description of the methodologies used within the field. The section concludes with an overview of current opportunities within the field.

Section 2.3 looks at data and information, starting with how information is stored; this is followed by a review of metadata and its importance in Forensic Computing. **Section 2.4** looks specifically at the thumbnail cache, discussing how thumbnails are used within computing and why they are useful. This is followed by a more detailed examination of five common operating system thumbnail caches and a summary of their forensic significance.

2.2 Forensic Computing

Forensic computing is thought to have originated in the late 1980s [Hannan, 2004], research and development in the field is generally driven by market requirements [Noblett, 2000]. This is partially due to the rapid growth of technology throughout society and the limited resources available for analysts to perform investigations; therefore research is generally conducted to enable the analysis and understanding of current casework.

The definition of Forensic Computing has been the subject of debate; Hall [2005] compared a variety of definitions that are in use; for example Garber [2001] describes the subject as:

“the analysis of hard drives, CDs, floppy disks, zip and jazz disks, flash memory cards and other forms of storage media to uncover evidence of crimes”.

One of the most common aspects identified throughout the definitions Hall examined is the requirement to ensure the preservation of both the physical and digital evidence [Hall, 2005]. Forensic Computing has been shown to have different professional and academic concerns to those found in computer science [Cooper, 2005]; the field has more similarities with the professional and academic concerns of Forensic Science. Whilst there are many definitions available for Forensic Computing, it can still be defined as a sub-discipline of forensic science [Hall, 2005]. The Oxford English Dictionary [OED, 2011] defines forensic as *“relating to or denoting the application of scientific methods and techniques to the investigation of crime”*; therefore Forensic Computing could be considered to be the application of the principles of forensic science to digital devices.

The investigatory process aims to prove or disprove a hypothesis, using methods which can be presented in court; the hypothesis generally relates to the actions of an individual. An analyst needs to handle the digital media in a suitable way to ensure this objective is fulfilled. The focus of forensic computing is to investigate an event after it has occurred through acquiring, analysing and reporting any evidence recovered from digital media [Hannan, 2004]. McKemmish [1999] identified four key components of forensic computing in terms of their involvement with digital evidence. These are: (i) identification, (ii) preservation, (iii) analysis and (iv) presentation; each component having its own procedures and requirements.

Generally individuals, including some from the scientific community, have pre-conceived opinions of the activities involved in a forensic analysis [Palmer, 2002]. The stereotype of forensic science is predominantly the study of human corpses, which can be partially traced back to the media's interpretation of the discipline. Whilst awareness of the discipline is increasing, the current lack of public awareness about forensic computing can be an advantage.

Within Forensic Science there is a constant requirement to produce evidence which is admissible in court [Palmer, 2002]; this permits it to be used in establishing the validity of the hypothesis being disputed in the case. The forensic examination of digital media is based on scientific methods; this ensures that objective and justifiable methods [Noble, 2000] are used for analysing evidence. In Forensic Computing, data are usually stored in binary which an analyst must interpret to extract information. The interpretation of the data requires an analyst to understand the storage structure of the media, and of individual files, in order to apply an appropriate and accurate context to allow the binary to be interpreted correctly [Palmer, 2002].

Every analysis method has an error rate, which needs to be measured and should be taken into consideration when interpreting the results [Palmer, 2002]. Understanding the nature of the errors allows the data to be presented in a way which reflects its accuracy. As the field develops, information on error rates may be discovered; this information can be used to increase understanding and improve techniques. For example, Dr Solomon was an early expert witness in forensic computing, who typically used evidence relating to the system clock [Gold, 2005]. It has since been established that time and date information may not always be accurate and this increased understanding of computer behaviour has modified how such information may be used in the field.

2.2.1 Methodologies

A computer can be considered an individual crime scene and should be treated accordingly [Carrier, 2005]; therefore a suitable methodology should be employed during its investigation. This section discusses current approaches to conducting investigations in forensic computing and the development of methodologies due to advances in technology.

Depending on the crime being investigated it is important to ensure the evidential value is preserved; therefore it needs to be handled by the appropriate agency. In an incident response guide, Microsoft [2007] produced a table detailing who to contact for the most common types of crime. The incident response guide is not well publicised and therefore companies may not be aware of who is appropriate to contact if an incident occurs; this may lead to evidence being mishandled. Different organisations have different priorities during an investigation; the primary objective for law enforcement when completing a forensic analysis is prosecution. However prosecution is a secondary concern for companies and the military. Both the military and corporate organisations are primarily concerned with ensuring they can continue to operate. It is important for them to identify the damage caused by the incident to ensure appropriate action is taken to resolve the situation.

Carrier [2005] discussed eight different organisations who are involved in independently developing definitions and standards for forensic computing. This highlights the requirement for organisations to work together to unite the field with standardised terminology and methodologies for what can be considered a global problem. Performing a forensic analysis that would result in admissible evidence is considered a task that most IT departments are under qualified to do [Forte, 2006]. An IT technician may be given authority to perform an analysis or seize equipment without being aware of the constraints involved in performing forensic tasks. This is particularly likely when the incident is first

thought to be a disciplinary offence within an organisation and at a later date evidence is discovered which results in it becoming a criminal investigation.

In order to examine digital media for the purpose of extracting evidence it is necessary to ensure the individual is authorised to do so [Microsoft, 2007; Reed, 2005]; without authorisation to examine the device any evidence collected is not legal and therefore cannot be admitted in court. Therefore the technician may inadvertently harm the reliability of any evidence recovered. Both IT professionals and forensic analysts need to ensure they only perform activities they are trained to do and fully understand [Greenfield, 2002]. Educating professionals in securing evidence and contacting appropriate personnel would reduce the number of cases that are dismissed due to inadmissible evidence [Carrier, 2003a].

It has been suggested that any methodology for the analysis of digital evidence should be defined in stages which are based on how the information is created rather than being based on existing techniques [Carrier, 2003b]. This would require removing any preconceptions about forensic computing methodologies and focusing on the behaviour of digital evidence. Creating a methodology based primarily on the evidence being analysed is likely to lead to a greater understanding of the evidence and the retrieval of a greater volume of information.

Ciardhuain [2004] discusses the change in approach to methodologies around 2004 when instances of encrypted media were increasing; this led to a shift from focusing on dead machines to performing a live analysis where appropriate. A thirteen step model was proposed by Ciardhuain which broke the forensic process into clear stages. This included the transportation of evidence and proving the hypothesis; this allows each part of the investigation to be clearly documented and gives an opportunity to provide specific training for

each stage. One of the most critical steps of an investigation is imaging the digital media [Stephenson, 2003], as any analysis is performed from an image to ensure the original evidence is preserved. It is therefore necessary to protect the original evidence during imaging whilst ensuring an accurate image is acquired.

Windows operating systems have been shown to leave a variety of traces that can greatly assist a forensic investigation [Greenfield, 2002]. Greenfield looks at common sources for finding forensic artefacts in Windows, suggesting that operating system guide books can provide useful insights into how and where information is likely to be found. It has been suggested that the new features found in operating systems such as Windows Vista [Hayes, 2009], may impact the way forensic examinations are conducted, particularly with the introduction of technology such as BitLocker [Hargreaves, 2008]. If encryption is used on a machine it would significantly impact the investigation by potentially making the evidence inaccessible. Having encryption tools resident within an operating system does not mean they will have a significant uptake with the general public; therefore traditional analysis of artefacts on unencrypted systems is still important.

When considering creating an overall methodology for forensic computing investigations, or specific parts of the process, it is necessary to ensure that it is chosen based on the characteristics of the problem [Bogen, 2005]. Selecting and creating appropriate methodologies will maximise the retrieval of admissible evidence and the use of resources. Unsafe practices are occurring throughout investigations due to a lack of standardisation and definitions, which make it difficult to create appropriate new tools to assist with the analysis of machines [Carrier, 2006]. Therefore defining the problem allows for tools to be created which satisfy the necessary requirements, thereby improving the analysis process.

2.2.2 Tools used in Forensic Computing

During an investigation, tools play an important part in obtaining and analysing data; anything an analyst uses to assist him during a case can be thought of as a tool. Tools could be physical or software, however it is beyond the scope of this research to discuss physical tools in depth. There are two categories of software used by an analyst: those for researching the behaviour and structure of an object and those used for identifying evidence; some tools may be used for both purposes. This section examines software tools that are currently available and discusses the requirements for creating a new forensic tool.

A virtual machine is a software version of a computer; it enables a virtual instance of a computer to be customised and used like a standard machine. Within a virtual machine it is possible to customise the hardware and software. It is also possible to limit how the virtual machine interacts with its environment, For example by restricting input devices or network access. Virtual machines are commonly used by forensic analysts to determine the behaviour and structure of files and software [Penhallurick, 2005], as they provide a safe test environment where the analyst can monitor and control the variables being assessed. One of the main advantages of using a virtual machine is the ability to easily replicate the machine, which assists an analyst in ensuring the same basic configuration is used throughout an experiment. A virtual machine can also be used to obtain a live view of a hard drive image that has been acquired [Bem, 2007]; this can provide an analyst with information about the layout of the system that may not be immediately obvious from the image alone.

Live boot CDs can allow an analyst to view and analyse a hard drive without altering it, specific forensic versions of the Linux operating system, such as Penguin Sleuth and Helix, have been created [Forensics Wiki, 2013]. Such CDs provide appropriate tools for an analyst in a live boot format to enable them to

obtain and analyse evidence [Helix, 2011]. Forensic live boot CDs do not automatically mount the media and ensure write access is only granted if specifically selected in order to protect the reliability of the evidence. The set of tools provided are useful both during the analysis of a case and when an analyst is performing experiments; they allow the media to be analysed in an appropriate way.

A common open source forensic tool used by analysts and found on live boot CDs is the Sleuth Kit. It allows a forensic analysis of a variety of different file structures including NTFS and FAT systems [Galvao, 2006]. The Sleuth Kit provides tools that can assist in finding forensic artefacts both in case work and experiments. Whilst the Sleuth Kit is a command line set of tools, a graphical interface is provided in the form of the Autopsy browser, this assists analysts in accessing the tools they require, making it easier to understand the functionality.

A closed source forensic tool commonly used by analysts is Encase, this tool encompasses a wide range of forensic applications such as the ability to write object-orientated scripts, analyse files and generate reports [Guidance Software, 2011]. Encase is an intuitive tool that provides the analyst with a wide range of functionality; however it is a closed source application and does not provide the complete information on files it has recovered. Therefore an analyst needs to be trained in how to interpret and verify its results using another method. File recovery, also known as file carving is discussed further in **Chapter 7**.

Hexadecimal [Hex] editors like Winhex are a useful tool when performing a forensic analysis and when trying to establish the structure of data [Fleischmann, 2008]. Viewing data in a hex editor allows an analyst to see the file as it is stored and allows the analysis of individual fields to identify their

structure and behaviour. Winhex provides an analyst with various tools, such as the ability to create scripts and templates to make working with large files easier; it also contains a data interpreter which converts selected hex into other formats [Casey, 2004]. Such tools make working with hex easier and less time consuming, allowing an analyst to concentrate on their analysis. They are often particularly useful in analysing unknown binary file structures as they provide an interface to examine, compare and deconstruct files.

There is significant debate within the forensic community about the relative merits of open and closed source software. Whilst both have been shown to meet the requirements for tools used to produce evidence, open source tools have been shown to do this more comprehensively [Carrier, 2002]. Whilst the source code is not available for closed source tools any tool used during an analysis should use documented procedures; the results of a tool should also be corroborated using another method to provide independent verification. Creating a new open source method allows analysts the opportunity to explore the source code used and fully analyse the tools behaviour. Carrier suggests that this should be accompanied by written documentation on the procedures, to supplement the analyst's understanding.

Ensuring that any new tool created uses a layout that is similar to existing forensic tools will make it easier to train analysts in its use [Richard, 2005]. The tools interface should also conform to the standard layout of functions used by the operating system, making it intuitive to use. Providing the analyst with information about the processes being performed by the software and its status will ensure they are aware of its behaviour.

2.2.3 Current Opportunities

Forensic computing is a relatively new field which is growing at a fast pace and encompasses a wide range of devices. This rapid growth is due to society's expanding development and use of technology; each technology has the potential for misuse and may therefore be the subject of an investigation. In this section the opportunities that have been created for forensic computing research will be discussed.

The increased use of storage media may provide a greater chance of recovering information for a case. However It is impractical, given the limited time, equipment and financial resources forensic analysts have available, to analyse every cluster of a large storage device by hand [Hama, 1996]. This can result in information not being found particularly if it resides in partial fragments of unallocated space. The use of multi partition systems or RAID drives does not impact on the ability to identify information in unallocated space; the impact on identifying information relates to whether the file is compressed or encrypted. Any fragment which is part of a file structure which is uncompressed and unencrypted may be identified by an analyst using standard identification techniques discussed in **Chapter 7**. Therefore a current opportunity for research is to create methods to automate the retrieval of information; this enables resources to be maximised and to reduce the information that an analyst must initially review by hand.

With the use of the internet, network storage and portable media, evidence of a crime could be present in multiple locations across the globe [Berinato, 2007]. The distribution of data can cause problems in collecting and analysing evidence as it may be difficult to retrieve evidence from media in a different geographical location. The media may be in an unknown location or not even known to exist. Whilst it is impractical to retrieve all distributed data, the ability to show relationships between the main device and devices used in conjunction

with it are important aspects of an investigation. Further research is required on forming the relationship between devices; the relationship can show that they were used together and how information passed between the two. Time and date information within both the devices can also assist in event reconstruction.

The development of operating systems like Windows Vista provide new challenges for analysts [Hargreaves, 2007], including the use of new encryption technologies [Woodward, 2006]. Despite the initial concern from the forensic community over the impact of Bitlocker, the existence of it does not suggest that it will be used by a significant proportion of users. Given the limited resources available for research into technology it is appropriate to examine those which are required for casework. This approach can lead to problems if a case requires the understanding and ability to extract evidence from such technology and that level of understanding is not yet available. Key components of new operating systems that have been altered in new versions, such as the recycle bin and thumbnail cache, provide interesting research opportunities [Hargreaves, 2007] as they are likely to provide evidence in a substantial number of cases and therefore the ability to understand their behaviour and develop methodologies to extract information from them in a variety of circumstances is useful to the forensic community.

Most techniques used by forensic analysts to analyse data are not standardised [Meyers, 2004]. As discussed further in **Chapter 4**, techniques are generally developed which meet the constraints of the team involved in the investigation. Standardised techniques for analysing and retrieving data from digital media would improve the ability to find and analyse evidence consistently and would assist in communicating the methodologies to lay individuals.

Forensic computing research tends to concentrate on methods of retrieving information to present in court. It is also appropriate to investigate anti-forensic

techniques, in order to fully understand how the information on the media being analysed by an analyst may have been manipulated [Harris, 2006]. Whilst the types of anti-forensics currently employed have largely been established, their effect on the system, possible ways of reversing their effect and minimising potential damage have yet to be fully explored. Anti-forensic techniques can have a similar effect to those found in data deleted by the user; partial data may exist in fragments still on the system. Identifying, reassembling and establishing the reliability of data fragments can assist analysts in retrieving artefacts from both deleted data and anti-forensic techniques.

2.3 Data and information

Physical items such as hard disk drives, CD-ROMs and USB drives store data that is utilised by both the computer and user. This section explores: the information contained in such devices, the methods used in storing the data, how this data may be corrupted and the retrieval of corrupted files. This section defines data as a binary pattern which when interpreted in an appropriate context; information can be extracted from the data for use by the system or the user.

2.3.1 Storing information

A user's habits and requirements are generally reflected in their file usage [Douceur, 1999] which allows an analyst to identify the type of user and the function of the storage media under examination. The content of a file tends to correlate with its type; in addition the average age and lifetime of files does not tend to vary across different types of user. This suggests that information about the file type is more likely to be relevant than information about the user when trying to identify file fragments.

An important question for each artefact an analyst identifies is its position within the file system structure; he must identify the storage structure of the devices. For example *thumbs.db* files can be found throughout a Windows XP operating system and in order to establish the meaning of a particular instance of the cache it is necessary to establish its relative location [Hurlbut, 2005]. This is forensically significant as it provides an understanding of where any evidence came from in relation to the rest of the information being stored. It is equally important to clarify where a deleted file was in relation to other files on the disk. However, this can be difficult for an analyst when the folder containing the deleted file is no longer on the disk; in this case the file is known as an orphan. Hurlbut investigates methods of identifying this particular class of deleted files with relative success. For a visual thumbnail it is important to establish a relationship to the original file the image it relates to; this can provide important information about the original file, such as the file path or associated dates and times.

It would not be an effective use of resources to examine every piece of stored information in every investigation; therefore it is necessary to reduce the amount of information to be analysed. Whilst this ensures value is obtained from the resources it may also lead to some important information being missed [Hama, 1996]. Hama suggested one method of refining the information is to ignore anything that is unlikely to be relevant to an investigation; for example, standard libraries or images that come with the operating system or commercial applications. Whilst this does reduce the number of files for consideration, care must be taken to ensure that files which are discounted as a result of such pre-processing have not been altered by a criminal aware of such procedures. Files can also be discarded on the basis of their extensions; however any method would need to check that the header and contents of the file match the extension. Any unused space on the storage media would need searching for deleted or partial files. Finally string searches can be useful in locating information, but the results can vary significantly based upon the keyword used.

Kerr [2006] identified that an analyst will require a significantly longer time to analyse a computer than the user would have taken to carry out the actions under investigation. This means that with the increasing size of storage media significant resources are required to carry out a thorough examination. However, increasing the number of analysts used during investigations is unlikely to be effective. Improving the methodology currently being employed by standardising analysis techniques will enable the use of available resources to be maximised. The use of resources could be maximised if analysis techniques were standardised by developing techniques which make the most efficient use of computer and personnel time; however it could be argued that each individual forensic team is likely to develop approaches which maximise their personal set of resources. The development of analysis techniques is also likely to improve the results of analysis by ensuring a methodical approach is adopted instead of each analyst analysing a machine in their own way.

NTFS file compression may provide problems for analysts as not all tools employed can retrieve information stored in this way [Sanderson, 2006]. A file compressed in this way can be composed of both compressed and uncompressed sections, making it difficult to interpret when the Master File Table (MFT) is not available. Generally documents like those produced in Word are highly likely to be compressed whereas JPEGs are unlikely to be compressed in this method, due to the high rate of compression already employed in that format. Interestingly if a document is compressed using NTFS compression when it is deleted, then it will remain in this state after deletion [Sanderson, 2006]; this suggests such compression techniques must be taken into consideration when carving files, especially when no MFT information is available.

For this research it is necessary to understand the structure in which the thumbnail cache is being stored and how this can be affected; this ensures the evidence extracted is put into the appropriate context. Since a thumbnail cache contains at least a single file, it will need to be stored on a storage medium to be used; therefore understanding the structure used on storage media will assist in retrieving information from the cache.

2.3.2 Metadata

Metadata is commonly defined as data about data; it is information which is used to understand the original data and assist with the systems function [NISO, 2001]. Metadata can be resident within files or found in related files such as caches, it can also be found in components of the operating system. Ruhnka [2008] suggested that metadata can be important in judicial proceedings as it can establish information about files; for example the creation time and author. This information is useful for establishing where a file originated and how it came to be at its present location. Metadata can form an integral part of a file, in such cases it maintains its relationship with the document at each new location, both on the machine and external media. Removing the metadata resident in a file, can often prove difficult and requires specialist tools [Castiglione, 2007].

When an analyst is working on a case they only have the system from which to complete their analysis [Buchholz, 2004]. This makes metadata a useful source of information as it can provide information such as times and dates in relation to the files. Creating a timeline of activity can provide an added dimension to the evidence. Buchholz defines the forensic value of a system as *“the possibility to draw conclusions about events on the system from the data”*. This definition focuses on the need to have reliable information. Current operating and file systems were not designed with forensic practitioners in mind, and are therefore focused on the needs of the user; this can make the retrieval of information which satisfies evidential requirements difficult. An analyst is often trying to

answer questions such as who, what, when, where, why and how. However these cannot all be answered using the information available from the digital devices; whilst approximations can be made with experience, the lack of certainty can significantly impact upon the value of the evidence.

The average user is only aware of the obvious information displayed by a computer, and may have no idea that metadata is stored [Byers, 2004]. Documents such as Microsoft Word have been known to store a variety of metadata including information from previous versions of a document. This information is retrievable and can provide an insight into the time line of a document. The Internet provides a vast amount of information and access to resources; these include tools which assist the user in uncovering hidden information in files [Kwon, 2008]. The retrieval of hidden data can assist an analyst in identifying where documents were created and their development, these can be useful evidential artefacts.

If a user is engaging in criminal activity they are likely to attempt to hide their activities, and also to hide the fact there is hidden data present on the system [Park, 2009]. If a user hid information but did not adequately cover their tracks then an analyst may find evidence that hidden information was present and search specifically for it. In some formats such as Microsoft Office 2007 files, parts of the file which are not in a structure recognised by the software are ignored when processing it; this means a user could potentially hide information in a file without affecting the functionality of it. This unknown data moves with the file, and could be discreetly transmitted. An analyst could identify data hidden using this method by validating the structure and syntax of the file.

Modification, access and creation (MAC) times of a file can give the analyst an insight into how the files arrived in their current location. Seven rules have been created based on these times which aid the analyst in their analysis [Chow,

2007], which assist in building event timelines as they elaborate on the relationships between MAC times. They can also elaborate on the user and system behaviour effects on the file's state. Chow reviews the value of the thumbnail cache, in relation to these rules, for establishing information relating to the viewing of such files [Chow, 2007: Scenario 9]. This suggests that the thumbnail cache can be used to assist in building timelines of activity on a machine. MAC times are sensitive to manipulation [Waghorn, 2003] and require careful consideration by analysts to ensure they maintain their validity, not just because of accidental clicking on a file, but ensuring there are no traces of programs specifically designed to alter MAC times, and that the BIOS time is set as expected.

2.3.3 Event Reconstruction

Digital investigations use the same five key phases as those used in traditional physical investigations; preservation, surveying, documenting, searching, reconstruction [Carrier, 2005]. Reconstruction in digital investigations involves analysing the information recovered from a system and constructing events from it; the events can be ordered to show how the information recovered came to exist in its present state. An event can be defined as an incident that changes the state of one or more objects; in a digital investigation, an event can be thought of as an action that changes one or more of the bits in which the information is stored.

In a case, event reconstruction can assist in proving or disproving guilt by establishing the order in which events occurred. Once events have been reconstructed, characteristics can be used to determine whether the actions were committed with intent [Carney, 2004]. For example, analysing times can give an indication of whether an action was the result of a piece of software or the user, since the user will have slower reaction times than the machine; there would need to be larger gaps between events than a machine would require.

However whilst this technique currently assists analysts it is possible that criminals may use such knowledge; when creating their software they may make the times involved in the events further apart to simulate human interaction. The thumbnail cache also shows user involvement as most caches of this type only add information when it is viewed by the user. Whilst the times stored within the thumbnail cache vary in meaning they can be a useful source of information for event reconstruction, providing information on when a file was created or in some caches when the entry was added.

To reconstruct a set of events to form an accurate timeline, an analyst needs to establish the time interval in which each event may have occurred. Time is a construct which is said to have been established to assist the functioning of society [Duren, 2002]; in forensic computing this construct allows an analyst to define when events happen, to create an order in which events were completed. To enable a global definition of time it is necessary to establish a 'trusted' time to which events can be related, this is known as UTC time. Such a system is particularly useful in establishing the order of events in forensic computing; it is possible an event relevant to a case could have occurred anywhere in the world. Therefore it may be necessary to link activities between two or more time zones. Since it is difficult to measure time exactly, an event is often modelled as an instantaneous action in event reconstruction models [Gladyshev, 2004]. An analyst needs to understand how time is modelled to ensure he understands the relationship between the time shown and its meaning.

Whilst timestamps are generally recognised as showing periods when the user has been active, they can also assist in identifying periods of inactivity [Stone-Kaplan, 2003]. Inactivity can be important in establishing the behaviour of the user who created the evidence and also in linking evidence to a particular user, for example the inactivity may correlate to when a user's family were at home

and therefore he did not have private access to a machine. During event reconstruction an analyst attempts to minimise the time period during which an event may have occurred [Gladyshev, 2005]; minimising the time allows a series of events to be grouped together to assist with establishing what caused each change to the system and when the change occurred. It is not always possible to establish the time an event occurred; forming relationships between events can determine the time period they occurred in. Relationships show the cause and effect of each event occurring therefore knowing the times of some events in the chain allows an analyst to propagate this information through the chain of events he has established and update the possible times other events could have occurred.

Information on the time zone can be found in the registry [Boyd, 2004]; should a user change the time zone multiple times it can be difficult to standardise the times found on a machine [Weil, 2002]. This could provide an element of uncertainty in the event reconstruction of the evidence; in such cases it would be necessary for an analyst to minimise the possible time frame in which the changes to the system were made by the user. This would assist in standardising the times for the remaining events.

When Windows displays times for files the adjustments it makes are inaccurate. For example, when displaying times in the summer, times for files altered in the winter are inaccurate; this is due to daylight saving time differences [Stone-Kaplan, 2003]. An analyst must therefore calculate the times independently to ensure they are accurate. Dynamic date and time stamp analysis uses information that is independent of the standard MAC times to correlate the actual times [Weil, 2002]; such correlation is the equivalent of an analyst using multiple tools to check his findings and provides additional accuracy to the evidence being presented. A check list for time and date evidence has been created [Boyd, 2004], which aims to ensure such information is dealt with consistently by analysts to ensure the reliability of the evidence is maintained.

Event reconstruction can be modelled using finite state machines [Gladyshev, 2005]. However in order to construct each model it is necessary to spend a significant amount of time analysing the information to produce the states and identify the relationships; it is also time consuming to evaluate files in this way. However it is useful for identifying the order of events and can provide assistance in identifying impossible sequences of events, therefore it is necessary for an analyst to assess the potential reward from using such a system against the high use of resources.

Neural networks have also been used for event reconstruction [Khan, 2007]; with a recurrent neural network achieving the highest accuracy of those tested. However accuracy overall improved when the complexity of the data set was increased; providing a more realistic model of the file types likely to be encountered during an analysis. If the neural network is not presented with realistic data, such as a cross section of files it is likely to encounter when used on a suspects hard drive, then its ability to function on an actual hard drive may not be as high as its ability during testing.

Zeitline is an event reconstruction method, which stores events in chronological order to make it easier to identify patterns and group events together to create larger more complex events. The ability of this package to model complicated events makes it very useful to an analyst however it relies heavily on user interaction and may therefore require too many resources for some investigations.

2.3.4 Corrupting data

Corruption can be the result of deliberate manipulation of data by software, such as the deletion of files, or the result of a virus; however the reliability of

data may be affected by the behaviour of the user [Casey, 2006]. Not every user is reliable in their actions; they may not act as expected, for example a user may use a log on name other than their own to provide the appearance of another user committing a set of tasks. Since the user is often an unknown and unseen factor in the equation it can often add several degrees of separation between the analyst and the real results of an analysis. An analyst needs to learn to account for the uncertainty the user brings to an analysis and qualify the results accordingly. However Casey [2006] has shown the negligibility of the user's actions; as it is possible for an analyst to analyse the available evidence and present the available information regardless of the user's behaviour.

Whilst data may be corrupted through purposeful actions, a degree of accidental destruction of the data is often unavoidable; for example, employees who are inadequately trained may input inaccurate information or delete the wrong files [Ozdemir, 2002]. Such accidental destruction can be a significant problem for organisations, and therefore employees may not voluntarily report such behaviour, causing further reliability problems for the data and the organisation. Possible ways of managing this are discussed by Ozdemir [2002], however human error will always play a role in the corruption of data; any system for mitigation is unlikely to completely remove the problem. Since the user has no active involvement with the thumbnail cache, except for altering the original file, user error is an unlikely cause of corruption of the data. However the individuals who create software which reads and writes to the thumbnail cache may cause the corruption of data within the cache, for example, by not providing the information required by the cache in the correct format.

Hard drive failures can be caused by both physical and operational errors. When a physical sector is defined as faulty it is not possible to read and write to it without specialist equipment; the computer can generally identify failures and keep a record [Elerath, 2007]. Operation failures are generally not fixed which

makes them difficult to identify, and can affect the reliability of information being read or written to the storage media, making it corrupt.

Both users and IT professionals are often unaware of the information being stored on a hard disk. Joukov [2006], discusses common misconceptions; for example, most users believe simply deleting a file removes it, when in fact the file is still retrievable. Explaining misconceptions to users improves their understanding and may directly influence the precautions they take to secure their data. Secure overwrite systems are often used to ensure files are deleted adequately; fortunately for forensic analysts, these systems generally neglect metadata that is not stored with the file.

When deleting files, temporary files are often overlooked; these files may contain as much sensitive information as the files the user is deleting [Zetterstrom, 2002]. It is suggested that to ensure sensitive information is not recoverable the best options are to sand the disks in the hard drive or put it in acid, whilst these may seem drastic it is likely they would completely destroy the original file system [NIST, 2013]. Various tools are available to corrupt information, and although they may not be as comprehensive they would remove a significant portion of the information.

2.4 Thumbnails and the Thumbnail Cache

A Visual thumbnail can be defined as a small graphical representation of a larger object [Microsoft, 2002]; a thumbnail cache is generally considered to be a collection of these images stored with associated metadata. The primary purpose of the thumbnail cache is to prevent the rendering of a new version of a thumbnail unless the source object has altered. This section discusses the applications of visual thumbnails within computing and the benefits of using a cache to store the images. This is followed by an overview of the way the

thumbnail cache is implemented in five well known operating systems and a discussion on the forensic significance of thumbnails and the thumbnail cache.

2.4.1 Visual thumbnails and their applications in computing

Thumbnails are generally used to allow a user to browse and make a visual selection of an object, when a thumbnail is selected it can provide the user with the full sized version. This section discusses how the operating system, web and software use visual thumbnails to improve their usability.

Operating systems use thumbnails when displaying directory contents, to assist the user in identifying and browsing their files. Individual files can be shown as visual thumbnails; this provides a graphical interpretation of at least part of the source files content. In earlier systems thumbnails were usually of graphical file formats. A large variety of software such as Microsoft Office Documents, and Adobe Portable Document Format now support the creation of thumbnails by allowing the first page of documents to be seen. Despite the increased support for the creation of thumbnails the implementation of thumbnail caches is not consistent across operating systems or software applications; this discussed further in the next section.

Search engines like Google [Google, 2013] use visual thumbnails to assist users in locating relevant information, for example when they are searching for images. Thumbnails allow for a larger number of images to be shown on the screen, which makes browsing for relevant content easier. Dziadosz [2002] showed that displaying text and a visual thumbnail assists users in making decisions about the relevance of web sites to their objective. This may reflect the increase in information available to the user when they are making their decision; the user has two different sources of information which may provide a more detailed overview of the website. The research also established that users

make less conservative judgements about the relevance of websites when presented with a visual thumbnail. The visual thumbnail provides the user with an opportunity to assess the layout of a webpage and identify characteristics which may show whether it is suitable such as colour and graphics. Photographic software such as Photoshop and iPhoto contain thumbnail caches which assist in the viewing of large quantities of images. These caches are built to provide quick access and minimise the use of system resources

2.4.2 Operating system Thumbnail Caches

In Windows XP, thumbnail caches are located in every directory which has been viewed using the thumbnail display option in Windows explorer (providing the option to cache thumbnails has not been manually turned off by the user). Each thumbnail cache consists of a single binary file called '*thumbs.db*', which uses the OLE2 container format [Roukine, 2012]. Each thumbnail cache stores information on the files within that directory, and may continue to store the information after the files have been deleted. A variety of files types may have associated visual thumbnails stored in the thumbnail cache, such as Microsoft Office Documents, these show the first page or slide of these documents. The thumbnails are stored within record entries in the catalog stream of the OLE2 container; the records are stored in a red-black binary tree structure, which allows fast access to each thumbnail when the operating system requires it.

The maximum size of a single visual thumbnail stored within the *thumbs.db* cache is 256 x 256 pixels; the size is not large enough to read individual lines of text but it can allow a match to be made with the layout of the original document. The thumbnail cache also contains various metadata such as file name, path and modification time; the metadata can assist in showing a relationship between the creating file and the thumbnail. The OLE2 structure is also used with the Microsoft Office Suite (versions 6-12). From Office 2007 saving in the binary structure was an option, an XML structure was also

available for office documents; therefore users may still create files using the original OLE2 structure. It is likely that any identification of *thumbs.db* fragments will have similarities with those from office documents making them a likely source of false positives, however it may be possible to generalise the methods used for identification to include these structures.

Vista introduced a new thumbnail cache format, which centralised the holding of thumbnails for each user. This centralised cache consists of six files; each object has a record in the index file which has references for up to four different sized thumbnails of the object, each stored in a file for their size range, the purpose of the sixth file is not yet known. The four image files hold thumbnails which are up to 32 x 32, 96 x 96, 256 x 256 and 1024 x 1024 pixels in size. The introduction of larger thumbnails provides the opportunity for a more detailed image, which can show readable text, enhancing their forensic value. To show a relationship between the original object and an entry in the thumbnail cache it is necessary to establish the path where the object resided when the entry was created or modified. The path information is not stored in the cache itself, but the windows.edb file [Douglas, 2009], whilst the remainder of the metadata is located in the cache. Windows 7 uses the same centralised six file thumbnail cache format as Windows Vista, but as yet it is unclear if the structure, its interaction with the rest of the operating system, and the resulting forensic artefacts, will be the same.

Ubuntu 10.10 uses Nautilus for file management, thumbnails are stored as images in a centralised folder, and thumbnails can be stored for a variety of image formats and PDF files [Ubuntu, 2010]. Open Office [2012] created a script which allows thumbnails for open office documents to be created and stored, increasing the variety of images found in the thumbnail cache. A unique feature of Nautilus' thumbnail implementation is the creation of audio thumbnails; putting your mouse on an audio file will result in the file playing, allowing the user to browse in a similar way to image formats. Further

investigation is necessary to establish whether the audio thumbnails leave any forensic artefacts on the system. With Nautilus implementing a separate folder to store thumbnails in standard image formats, carving for visual thumbnails in slack space is a standard image carving problem, with the carving and reassembly of any metadata or audio thumbnail fragments being a separate problem.

OS X can display files in a thumbnail view, however this is not on by default, and little documentation exists about how it is implemented. When the user selects to view thumbnail size images, they can choose the size of the image, therefore the forensic value of these images will vary depending on the size selection.

2.4.3 Forensic significance

Whilst the function of a thumbnail cache has been defined, the interpretation of the structure varies considerably between implementations. This means that each thumbnail cache structure will have a different forensic value and the data stored within it may not represent the same series of events. The images stored within the thumbnail cache are forensically useful in the analysis of a wide variety of crimes, as each image can represent a wide variety of file types and information.

For cases involving illicit images it can be necessary to show the authenticity of an image [Kornblum, 2008]; to establish whether it is a genuine unaltered photograph. If an image has been altered, and the quality of that section is lower than the remainder of the photograph it is possible to identify significant differences in the wavelet statistics [Lyu, 2005]. Given the quality of editing software currently available, the difference in quality of manipulated areas of photographs may not be significant enough to use this methodology in the

future. It is possible to establish the creation method of an image based on the noise and colour interpolation coefficients within it [McKay, 2008]; in the thumbnail cache this could provide validation that a thumbnail was created using the appropriate method for that cache and has not been manipulated or planted.

In Windows XP, the thumbnail cache files are only created when a user views a directory in windows explorer and views the files in filmstrip or thumbnail mode [Hurlbut, 2005]. Therefore the existence of a *Thumbs.db* file suggests that the file was present in a directory which at some point was viewed in thumbnail mode. Mercuri [2010] described a situation where the *Thumbs.db* could have originated in a directory on a different machine which at an earlier point in time contained images which were added to the *Thumbs.db* file. The directory if copied to another computer would potentially copy across the *Thumbs.db* file which would still contain records relating to the images. The images in question were not on the current computer and the user may have no knowledge of their original existence. Mercuri's example is taken from her defence work on the case of Vosburgh [Mercuri, 2010], and highlights the importance of establishing the context for the images contained within the thumbnail cache.

Further research is required to establish the situations where other operating systems cache images and to find out if there is a relationship between viewing a file and its addition to the cache. Where thumbnail caches are created in the relevant directory they may be found on external or networked media; this can be used as supporting evidence that the media was used in collaboration with the machine and viewed by a user. Where the thumbnail cache is centralised, the existence of entries for external or networked media can also assist the analyst, particularly in cases where the media itself is damaged or cannot be located as it would provide information on the contents of the media and show it had at some time been connected to that machine.

Thumbnail caches contain metadata, such as the original files modification time, location and name [Hurlbut, 2005]. This information is useful to corroborate other metadata sources related to the file; this is particularly useful when only fragments of the original file are available. The metadata and associated visual thumbnail are not always removed from the cache when the original file is deleted or moved. This can provide the analyst with information about the event timeline for a file and can also be useful when producing evidence from deleted files.

2.5 Conclusion

This chapter has analysed research related to this thesis; this began with an overview of the field of Forensic Computing. The methodologies and tools used in this field were examined along with current opportunities for research. It was identified that artefacts in new operating systems such as the changes to the thumbnail cache were of interest to analysts and therefore understanding their structure and behaviour is an area for current research.

It was also noted that anti forensic techniques may be employed to remove sources of potential information, therefore an area of current research would be to identify and analyse any potentially useful fragments remaining on the storage media. This Chapter examined data and information, focusing on how information is stored and the forensic implications of metadata. It is important to understand both the structure of the thumbnail cache and the storage media to ensure an artefact is analysed in its original context.

Finally operating system thumbnail caches were discussed, it was highlighted that whilst the general purpose of a thumbnail cache is defined each operating system implements the construct in its own way. As well as the difference in

structure and syntax the user and system behaviour which leads to artefacts being present in the thumbnail cache differs significantly between implementations.

The next Chapter discusses the problem definition for this research and proposes a methodology for its completion.

3 Research Objectives, Considerations and Tools

3.1 Introduction

This research aims to evaluate the evidential value of thumbnail cache file fragments identified in unallocated space; in order to achieve this aim the research must be broken down into smaller research tasks. This Chapter defines the objectives for this research and discusses the potential considerations and tools to be used in the completion of this research.

3.2 Objectives

The following objectives for this research are based on the related reading discussed in **Chapter 2** and the problem definition described in **Chapter 1**:

- **Define the term evidential value**

The term evidential value is used in the hypothesis as a method of assessing the artefacts recovered from both live and fragments of thumbnail cache files. In order perform this assessment it is first necessary to establish the definition of the term evidential value; a set of criteria which can be used for the assessment of this term on the artefacts would also be required.

- **Identify the structure and behaviour of an operating system thumbnail cache**

Whilst the identification of an artefact may provide a piece of information, it is the user or system behaviour which caused the artefact to be in its current state which provides the context. The ability to show both sets of information provides insight into the meaning of the artefact and also can assist an analyst in establishing the events that occurred and their meaning.

- **Identify existing file carving techniques**

File carving as described in **Chapter 7** is the process of extracting a set of data which meet specific criteria from unallocated space. In order to recover thumbnail cache file fragments it will be necessary to implement a file carver; identifying existing techniques will provide a starting point for the recovery of thumbnail cache files.

- **Evaluate the results of existing file carving techniques on identifying and reassembling thumbnail cache file fragments**

If a single file carving method was used to retrieve thumbnail cache file fragments and the results were used to establish the evidential value, the method may impact the results. By implementing a variety of existing techniques the results can be compared in order to maximise the recovery of file fragments.

- **Develop an approach to improve the ability to identify and reassemble thumbnail cache file fragments in unallocated space**

In addition to comparing the results of existing methods any resulting research gaps will be explored by attempting to improve the accuracy of file carving the fragments should it be necessary.

- **Extract and reassemble thumbnail cache file fragments using the developed method**

The research hypothesis focuses on file fragments, the file fragments require identification which will result in classification. Attempting to reassemble the various fragments into their original files will assist in providing context to individual artefacts.

- **Evaluate evidential value of thumbnail cache file fragments identified in unallocated space**

The file fragments identified and reassembled during this research will require assessment by the criteria established to assess their evidential value in order to identify the validity of the hypothesis.

- **Compare and Contrast the evidential value of the complete thumbnail cache with thumbnail cache file fragments**

In order to assess whether file fragments contain artefacts not found in live thumbnail caches it is necessary to compare and contrast the results of assessing their individual evidential value.

3.3 Potential Considerations

During an investigation there are various evidential constraints that impact the way in which an investigation is conducted and the handling of evidence. The majority of these constraints are concerned with ensuring each artefact is admissible in court and the evidential value is maintained. Some constraints are organisation specific and as such are mainly beyond the scope of this research; relevant constraints to this research and their impact on evidential value are discussed in **Chapter 4**.

Any methodology created during this research needs to provide adequate logging features to enable an analyst to verify the results produced. Methodologies created during this research must take into consideration the need to have results which can be repeated; therefore each method should take into account how an analysis may be performed by hand to retrieve the same information. The logging system would need to provide enough information to make the results repeatable; the information needs to be displayed in a clear and understandable format.

Finally a substantial constraint on this research is time and resources; therefore consideration should be made into the number of variables to be examined

throughout this research. Within this research it has been decided that the focus will be on artefacts most likely to assist in an investigation; the methodologies created can then be generalised to other less frequently observed artefacts.

3.4 Tools

Throughout this research the most commonly used tool will be a virtual machine, specifically VMWare Workstation version 7 [2011]; the virtual machines will originally be created as part of the structural and behavioural studies and will then be used as file fragments for the identification and reassembly research. The use of a virtual machine will enable a clean baseline of each operating system to be created which can be specifically manipulated for each individual experiment. The results of each experiment can then be compared with the original baseline virtual machine to identify the differences. The use of flat virtual hard disks means the hard disk was equivalent to a standard uncompressed hard drive image and can therefore be examined using tools and methodologies that would be used during a typical analysis.

Procmon [Microsoft, 2012b] will be used during the experiments in Windows systems to log the processes being called; the logs will be stored in CSV format and the results will be examined in Procmon and Excel to identify potentially interesting processes. Procmon results may provide significant insight into the processes being called when thumbnails were being created and displayed.

A hexadecimal editor called Winhex [Fleishmann, 2008] will be the primary tool used in the examination of the virtual hard disks; the Winhex software allows the file system of the disks to be viewed, which will assist in locating and reading artefacts in the live file set. Functionality, such as templates and basic scripting, will be used to assist in ascertaining the structure of the thumbnail cache and related files; once the templates are created they will be applied to

each experiment to identify structural changes to each file. The inbuilt data interpreter allows data to be analysed by selecting a set of bytes and looking through the various values calculated for different data types.

Standard analysis tools such as FTK [Access Data, 2012], Autopsy [Carrier, 2011], and Encase [Guidance Software, 2011] will also be used to examine hard disks to assist with establishing relationships between the thumbnail cache and other system artefacts. The tools will also be used to identify the information displayed about the thumbnail cache and compare it to the actual behaviour replicated during the experiments.

This research also requires the storage of significant amounts of test data to enable training and testing of file fragment identification and reassembly methods. Each virtual hard disk created during the experimentation phase in **Chapters 5 and 6** will be stored on two 2TB hard drives which will become the corpus data for this study. On the corpus drive the data will be stored in its original hard drive form as well as in separate fragments which will be classified by type; each fragment will contain metadata as a header which will store data about the hard drive the fragment came from and its original relative position on the drive.

3.5 Conclusion

This chapter has set out the objectives for this research; the thumbnail cache has been shown to be a source of potential evidence. The increased user awareness in hiding their behaviour makes it more likely that files containing potential evidence are located in unallocated space. In conducting this research it is necessary to be aware of potential evidential constraints which may impact on a potential digital analysis. It is also necessary to ensure any methodology created has adequate logging of its behaviour to enable an analyst to

corroborate the results. Time constraints will impact upon this research; therefore this research will focus primarily on three operating system thumbnail caches. Finally this chapter provided a breakdown of the tools which will be used during this research. The next chapter focuses on the first objective of this research, defining the term evidential value.

4 Defining Evidential Value

4.1 Introduction

This research aims to identify the evidential value of thumbnail cache file fragments identified in unallocated space; in order to evaluate the evidential value it is first necessary to define this term and construct a set of criteria by which thumbnail cache artefacts can be evaluated. This chapter examines the constraints on evidence which may affect the construction of criteria; this is followed by the creation of a set of criteria which will be used to evaluate the results of this research.

4.2 Evidential Constraints

In an ideal world, an analyst would be able to establish the exact series of events that occurred on a computer from the first time it was switched on. Every deleted file would be recoverable and all the necessary evidence to prove or disprove a series of actions had been committed would be retrievable. However there are unavoidable constraints which may impact the evidence presented in court.

This section looks at a cross section of the constraints which impact forensic investigations, and summarises their potential impact on this research. It begins with a discussion on admissibility and legal constraints, this leads into sections on evidence, reliability and procedures. The constraints identified in this section are then used to create a set of criteria for establishing the evidential value of thumbnail cache artefacts in the following section.

4.2.1 Legal Constraints

The law can be defined as “*the rules of a legal system*”[Griffiths, 1979]; since society adapts alongside the development of law it is suggested they may have a symbiotic relationship. For a forensic analyst the legal framework ensures evidence is handled in a fair and appropriate manner. The legal framework also allows an analyst to understand what constitutes the evidence required for conviction.

To be guilty of an offence it is generally necessary to prove that a physical act was committed (*actus reus*), and that there was a guilty mind (*mens rea*) [Reed, 2005]. A guilty mind is generally defined as acting with intent; therefore an analyst is not simply looking for the evidence of the crime, such as abusive images of children, but is trying to establish whether they were intentionally put on the system or are present without the users consent. This highlights the requirement for an analyst to show the system behaviour which led to an artefact being in its current state as well as the artefact which demonstrates an act occurred.

When a user is completing an activity, they will use any technique they feel is appropriate to fulfil their objectives. A forensic analyst must follow appropriate methodologies when looking for evidence of a user’s actions; this ensures any findings can be used as evidence. There are numerous resources on legislation regarding acquiring computer based evidence that document what evidence can be collected and appropriate methods for obtaining it [Thomas, 2004]; Forensic Computing methodologies are discussed in **Chapter 2**. Whilst such legislation constrains the actions of those investigating a case, it also ensures that the evidence being examined after acquisition has not been altered and is likely to be admissible in Court; the methodologies assist in preventing breaks in the chain of evidence that may prejudice the outcome of a case.

In Jurisprudence, admissibility is the term used to describe evidence which is allowed to be presented in court. For a piece of evidence to be considered for admission in court it must be shown to be reliable and relevant to the case [Kenneally, 2004]. The definition of relevance was restated in R v Nethercott [2001]:

“any two facts to which it is applied are so related to each other according to the common course of events one either taken by itself or in conjunction with other facts proves or renders probable the past, present or future existence or non-existence of the other”

In order to show digital evidence is relevant it needs to have probative value; evidence needs to provide information which supports or refutes the case's hypothesis. The ability to refute or accept a hypothesis in a case may rely on a combination of related artefacts which when considered together show an action or series of events which impacts on the likelihood of the hypothesis. In a criminal case it is generally the Judge who is responsible for determining whether evidence meets the criteria for relevance and is admissible [Ramjohn, 2009]. Whilst most evidence which can be shown to meet the definition for relevance is considered admissible there are situations where relevant evidence is excluded; these are known as exclusionary rules. The weight given to each piece of evidence is generally determined by the Jury when considering its importance in relation to the rest of the information in the case [Taylor, 2012]

The *Police and Criminal Evidence Act (PACE)* [1984] section 69 states the situations where evidence found from a computer would be deemed inadmissible in court; these are exclusionary rules of evidence. It states it is necessary to confirm the accuracy of the relationship between the original evidence and that documented in an analyst's case report. This is reflected in the ACPO guidelines [2010] which state it is necessary to minimise alterations

to the evidence and to ensure any action taken to retrieve evidence can be justified. To ensure compliance with the act, any technique used to retrieve the evidence must therefore be justifiable and its full effect on the system understood. The ACPO guidelines [2010] are one of the most important guidelines to ensure the admissibility of Forensic Computing evidence in the UK; these guidelines cover procedures to assist in ensuring admissibility and are discussed in depth in the ***procedural*** constraints section of this chapter.

4.2.2 Evidential Constraints

During an investigation an analyst analyses the evidence to produce a report; the report details any information found that may be relevant to proving or disproving the investigation's hypothesis. In Forensic Computing evidence consists of both physical media and the digital information stored on it.

The structure and format of storage media varies between manufacturers, this affects the methodology used; each device type will need handling in a specific way to ensure the most thorough examination is completed [Meyers, 2004]. If incorrectly identified or handled, data may be lost, and in the worst case the media may become unrecoverable.

Disk corruption may be physical or digital [Casey, 2008], with each requiring the use of suitable methodologies and equipment. It is necessary for the analyst to: identify the condition of the media, take appropriate action to ensure the data is not altered unnecessarily, and ensure the maximum amount of available data is recovered with the available resources. Recovering the data does not guarantee it will be interpretable; for example it may be possible to recover a sector of data, but may not be possible to identify the information it represents.

It has been suggested that to fully understand the behaviour of any part of the operating system a thorough analysis of the source code would be required, and that this is currently impossible [Pan, 2007]. This has led to a requirement for analysts to perform their own experiments in order to understand an artefact and the system behaviour it represents. As well as being aware of the structures used in the storage of information on digital media, it is necessary for digital analysts to understand the software or method they are using to recover and analyse the data [Ball, 2007]. Software used by analysts must maintain the validity of the data by ensuring the results maintain a relationship with the original data; therefore the information analysed is an accurate representation of the state of the information stored on the hardware at the time of acquisition.

4.2.3 Reliability Constraints

Reliability can be measured in terms of the consistency of the results produced. In the case of digital evidence, consistency can be shown to be the ability to repeat a methodology with the original evidence and obtain the same results. Whilst this is generally possible with a dead machine, live forensics methodologies are not usually repeatable as the original evidence is altered in the process of performing the acquisition [Hargreaves, 2009].

In order for evidence to be considered reliable there is a requirement to show a justifiable relationship between the original evidence and the information being presented in Court. This relationship would need to be based on: the scientific analysis of the evidence, the extraction of information, and the analysts performing the actions; the reliability of the evidence could be compromised by any of these components [White, 2010]. Evidence reliability is not limited to the process of handling evidence, but also to the way in which evidence was generated.

To ensure the evidence is accurate and reliable an analyst must only give evidence based on the originally acquired material [Ramjohn, 2009]. Any evidence submitted to the court must have a scientific basis, with appropriate justification for the methodology employed to recover it. A process can be considered to follow the scientific method when a hypothesis is formed based on the problem to be solved and tested through a process of data observation and experimentation [OED, 2011].

Evidence retrieved from digital media may be presumed to be inaccurate if the original media is altered, and this can cause legal problems [Casey, 2007]. Any tool an analyst has used to extract information which they produce as potential evidence must therefore be reliable. It is therefore good practice to validate any findings using multiple tools [ACPO, 2010]. This helps to ensure the reliability of any evidence retrieved by ensuring the same results are produced by multiple tools designed for the same purpose. Checking the results with a second tool or by hand reduces the probability of a mistake occurring during the analysis.

A popular defence to computer crime is the Trojan Defence [Harley, 2010]; the Trojan defence suggests the computer is not working properly. The ability to form relationships between artefacts can assist in establishing whether the system was acting as expected and maximise the corroboration of the information [Sansurooah, 2006]. It is also necessary to establish the reliability of individual artefacts at the structural and contextual levels. By establishing the structure and syntax of an artefact it may be possible to determine information about the system activity which led to its creation [Morris, 2010]; relationships can then be formed with other system artefacts to corroborate the information obtained.

4.2.4 Procedural Constraints

The procedures used by forensic analysts reflect the legal requirements for admissible evidence. Since evidence recovered may be required in legal proceedings, it is necessary to ensure the data are handled in a way which does not compromise its integrity. The primary procedural constraints adopted by most UK analysts are the four principles given in the ACPO guidelines [2010], which state

“Principle 1: No action taken by law enforcement agencies or their agents should change data held on computer media which may subsequently be relied upon in court

Principle 2: In circumstances where a person finds it necessary to access the original data held on a computer or on storage media, that person must be competent to do so and be able to give evidence explaining the relevance and implications of their actions.

Principle 3: An audit trail or other record of all processes applied to computer-based evidence should be created and preserved. An independent third party should be able to examine those processes and achieve the same result

Principle 4: The person in charge of the investigation [the case officer] has overall responsibility for ensuring that the law and these principles are adhered to” [ACPO, 2010: page 3]

Whilst these guidelines are not legally binding, they constitute good practice as they provide a set of general principles aimed at maximising the admissibility of evidence. The need to show a relationship between the original media and the

evidence being extracted is crucial to showing the reliability of analysts findings, the ACPO guidelines provide the principles necessary to ensure the reliability is not compromised.

Principle 1 highlights the importance of ensuring the original data has not been altered, as it allows the methodology used by the analyst to be tested by another analyst. By using an image of the original media and the same methodology it is possible to establish if the result is repeatable.

Comparing the hashes of different sources is one way an analyst can show a relationship between information; another way is by creating an audit trail [ACPO, 2010]. Clearly documenting the steps taken to retrieve the evidence being used in court allows the analyst to show that any information produced in his report as evidence is accurate and reliable; this is reflected by principle three. Audit trails help ensure consistency in the approach taken as the act of documenting actions encourages the analyst to consider his behaviour to ensure it is justifiable.

Following the ACPO good practice guide [2010] will ensure the reliability of evidence is maintained, particularly with regard to being competent and able to justify any action taken. The audit log suggested by the ACPO guidelines in principle 3 allows for results to be reproduced, which is an important scientific method for ensuring the reliability of any evidence found. Principle four stipulates who is ultimately responsible for ensuring the principles are adhered to. In a large case it may be difficult for a single individual to monitor every action being carried out, therefore it is necessary for each analyst to be responsible for their own actions and ensure they follow the principles.

To minimise the possibility of fraudulent evidence being admitted, authentication procedures are required to analyse the chain of evidence [Kenneally, 2004]. The analysis must account for any movement of the evidence, or interpretation of the data. This may be a physical change such as from the custody of an individual to another; or it may be a change in the interpretation of the data, for example locating and extracting deleted files. Justification for any action taken which alters the state of the original evidence needs to be provided to show an analyst understands how the evidence being presented in their report was obtained [National Center for Forensic Science, 2003]. The documentation of these actions shows clearly what has happened to the data and assists with establishing that any evidence produced is an accurate representation of the original data.

To ensure the ACPO guidelines are complied with, it is necessary to understand the consequences of any action performed ensuring the relationship with the original data is maintained. It is also necessary to justify the behaviour of the software being used during examinations, to ensure that any changes made to the evidence do not affect its integrity [ACPO, 2010]. In the UK, the ACPO guidelines are a significant part of a good practice approach to investigating computer crime. However as computer crime is a global problem and may involve several legal jurisdictions in a single case it may also be necessary to consider what constraints affect procedures elsewhere in the world. The USA has produced a set of guidelines based upon their legal requirements [NIST, 2004], which when compared to the UK's guidelines highlight the common theme of ensuring the integrity of the evidence by producing a clear chain of evidence.

4.2.5 Human Behavioural Constraints

A forensic computing analyst examines digital media, from which they produce evidence related to the issue under investigation. Whilst it is not their job to

analyse human behaviour, it can still have an influence over the evidence found and presented in court. This influence can be divided into the actions of three groups; the defendant, the analysts and legal personnel.

A wide variety of crimes can be committed which may involve computer based evidence [Bainbridge, 2007], it is therefore important to understand the crimes being investigated to ensure appropriate evidence is reported should it be found.

Any action on a computer leaves a trace; the amount and location of that information varies, and the average user is not aware of how to find and remove such traces. As with any subject, there is a wide range of ability in computer use, Smith [2008] looks at seven of the common misconceptions in computing, of particular interest is the notion that files are deleted once placed in the recycle bin.

The reason why a crime is committed is as individual as the person committing it; however the underlying motivations are generally similar [Dreyfus, 1998]. An in-depth discussion of criminal reasoning is beyond the scope of this research; however the varying ability of the defendant to manipulate technology in order to commit a crime is an important consideration. Having anti-forensic tools on a user's system may hold significant weight in an investigation as it may show an intent to obfuscate information.

The use of Anti-forensic techniques is increasing [Forte, 2006], these generally involve: zero foot printing, data obfuscation, and data hiding. Tools for anti-forensic practices can be found on the internet along with instructions for their use; however the effectiveness of such tools varies greatly. Lio [2006] identified 4 main goals of anti-forensics, of which the primary aim is to avoid prosecution

for their crimes; therefore anti-forensic tools aim to remove artefacts which relate to system activity.

As well as the technical skills and procedures of analysts, consideration should be given to the support they are given. It has been documented that continual exposure to harmful material, such as images of child abuse, can cause significant stress and discomfort for forensic practitioners [ACPO, 2010]. An individual repeatedly subjected to violent images may find themselves desensitised to their effect [Anderson, 1998], this can lead to a distortion of their morals; reducing their comprehension of the serious nature of the materials they are reviewing. It has been recommended that analysts have access to appropriate psychological support, to ensure their mental wellbeing. Carnagey [2007] has shown that continued exposure to violence can cause desensitisation; therefore it is necessary to ensure analysts minimise exposure to harmful material to minimise desensitisation. Automating activities is one method of limiting exposure to potentially harmful material.

Limited resources also make it desirable to automate tasks, such as fragment identification, than to have each fragment manually examined [Noblett, 2000]. A higher level of accuracy can result from automation, as the large amount of storage media would make it difficult for an analyst to identify all the available fragments for a particular file type.

For this research this section has identified a key requirement of automating the methodology which will assist in minimising the use of resources. The use of anti-forensic tools also leads to a requirement to ensure artefacts recovered are from a system which was working properly.

4.3 Criteria for evaluating the evidential value of thumbnail cache artefacts

This section identifies criteria for evaluating the evidential value of thumbnail cache artefacts. Evidence relating to the thumbnail cache can be initially broken into two kinds of evidence: evidence relating to the activity of the system, and evidence relating to the activity of a user. Both types of activity can assist in creating an event timeline of the overall actions of the system which allows artefacts to be put into context assisting the analyst to evaluate their hypothesis. The sub-sections below describe the criteria for evaluating the evidential value of thumbnail cache artefacts that were determined from the related research discussed earlier in this Chapter; the important points of each set of criteria are highlighted.

4.3.1 Provenance of Artefacts

Data provenance can be shown by ensuring the results can be replicated. Any method used throughout the analysis process should be justifiable and have understandable consequences that the analyst can explain. The process used should be a scientific method and requires clear documentation to provide other sources with the opportunity to replicate the experiments to corroborate the results.

In more detail:

- Results are considered *replicable* when they are documented sufficiently that another analyst could conduct the experiment again and achieve the same results.
- An action is defined as *justifiable* if it can be shown how the results were obtained and the effects on the original data are understood.

- A process is defined as having *understandable consequences* when at each stage of the process the results and their relationship to the original data can be explained.
- The process can be considered to follow the *scientific method* when a hypothesis is formed based on the problem to be solved and tested through a process of data observation and experimentation.
- *Clear documentation* is defined as something which another analyst could read, understand and use to repeat the described processes.
- A result can be *corroborated* when the documentation is clear enough to allow the process to be repeated and therefore the results can be verified.

4.3.2 Interpretation of Results

It is important to establish if the artefacts being analysed were created while the machine was working properly. If an artefact was created during the proper operation of the system it is simpler to establish the behaviour it represents. If the artefact was created outside of the system'

s proper operation then the anomalous behaviour the artefact represents may not be what the analyst expects. A single artefact on its own provides limited information about its meaning; by forming relationships between multiple artefacts it is possible to maximise corroboration of the information available to the analyst. In order to improve the likelihood of all relevant artefacts being recovered it is necessary to ensure that the maximum amount of data is retrieved therefore increasing the likelihood of retrieving artefacts with probative value. Due to the large amounts of data an analyst may encounter it may be necessary to use a documented automated methodology.

In more detail:

- In order to determine an artefact was created while the machine was *working properly* it is necessary to determine the system behaviour which led to an artefacts current state.
- In order to establish the *behaviour* an artefact represents it is necessary to form relationships with other artefacts to identify potential system activity.
- A *relationship* between artefacts can be defined as an association between artefacts; the relationship may provide corroboration or supplementary information that enhances what is known about the system activity.
- *Maximising corroboration* can be defined as verifying and providing additional information the artefact with other alternative sources of information.
- The *maximum amount of data* retrieved is defined as the identification and reassembly of all thumbnail cache file fragments used during the research.
- *Automation* is defined as a scientific process which has been implemented to minimise the interaction of an analyst.

4.4 Process Criteria

This chapter has identified criteria which can be used to evaluate the evidential value of thumbnail cache artefacts; some requirements emerging from the discussion of evidential value suggest constraints outside the scope of this research, but which apply to any investigation in which the results of this research were applied. They are documented below, for completeness.

4.4.1 Provenance of Artefacts

In order to ensure the original data to be preserved the analysis is generally performed on a verified image of the original storage media.

4.4.2 Legal and ethical constraints

Given the definition of relevance restated in R V Nethercott [2001], each collection of artefacts must assist with providing or disproving the central hypothesis of the case; therefore the artefacts must have probative value. The previous sections of this Chapter have shown that some forms of evidence may not be admissible under the exclusionary rules; therefore any methodology created must ensure it does not prohibit the evidence from being admitted. It has been shown that exposure to certain kinds of material can have a negative impact on the welfare of an analyst; therefore any methodology created should attempt to minimise exposure to potentially harmful material.

4.5 Conclusion

In conclusion this chapter has identified potential constraints on the evidence produced by an analyst. This was followed by the creation of a set of criteria for assessing the evidential value of evidence relating to the thumbnail cache; these criteria are used in **Chapter 10** to evaluate the thumbnail cache artefacts identified during this research. The next chapter describes a study of the structure and behaviour of Ubuntu and Kubuntu thumbnail caches which will be used as a baseline for comparison with the artefacts retrieved from unallocated space later in this research.

5 The Structure and Behaviour of the Operating System Thumbnail Cache Implementations used in Ubuntu and Kubuntu

5.1 Introduction

This chapter identifies the artefacts recovered from the operating system thumbnail caches used in Ubuntu and Kubuntu and evaluates their evidential strength against the criteria established in **Chapter 4**.

Kubuntu is an implementation of the Ubuntu Linux distribution that implements the KDE graphical desktop environment rather than Ubuntu's default graphical interface. An open desktop specification for thumbnail caches specifies that each system should implement a single cache which can be utilised by both the operating system and applications [Finke, 2004]. The file managers used in both Kubuntu and Ubuntu base their thumbnail cache implementations on this specification; a file manager provides a graphical interface to the file system. The specification does not specify the user activity which leads to the thumbnail cache being manipulated. Therefore both file managers may implement the specification for the user activity involved in their systems differently; the forensic significance of the artefacts may be significantly different in each case.

This chapter is organised as follows, first the basic methodology employed in this section is described [**Section 5.2**]. This is followed by an overview of the Linux thumbnail caches and a description of the artefacts found on a default installation [**Sections 5.3 - 5.4**]. Next, the structure of the thumbnail cache implementations used in Ubuntu and Kubuntu are identified; followed by an analysis of their behaviour based on user activity [**Sections 5.5 - 5.6**]. In **Section 5.7** the artefacts found on the rest of the system which corroborate thumbnail cache information are identified. **Section 5.8** explores how the

thumbnail cache responds to artefacts being altered, whilst **Section 5.9** identifies methods of forming a relationship between information in the thumbnail cache and an original source file. A brief discussion of artefacts relating to audio and media thumbnails is provided in **Section 5.10**; followed by a discussion of the strengths of the evidential artefacts identified during experimentation in **Section 5.11**. Finally, **Section 5.12** provides a conclusion to this chapter.

5.2 Methodology

In order to compare the structure and behaviour of different implementations of the thumbnail cache it is necessary to establish the methodology which will be implemented during the research. This section describes the overall methodology employed in this chapter; investigations of different aspects require specific experiments which are described in the relevant section below.

First research into existing resources relating to Linux and thumbnail caches are identified and evaluated, which ensures that any methodology created takes into account existing knowledge on the subject. The related reading described in **Chapter 2** identified that the thumbnail cache is stored in a hidden folder in the user's home directory. The main directory is named *.thumbnails* and contains up to three sub-directories entitled *fail*, *normal* and *large*; the directories and files are standard Linux implementations of these constructs. An open desktop thumbnail cache specification exists [Finke, 2004]; however there is no information to suggest the extent to which operating system implements it. The specification provides a starting point for the experiments into establishing the structure and syntax of the thumbnail cache.

In order to establish the structure and behaviour of the thumbnail cache and thumbnail records, it is necessary to perform a variety of experiments to assess

the way data changes. The experiments were performed on virtual machines, each with a 6GB hard drive, and the operating systems were downloaded in an ISO format from the Ubuntu website [Ubuntu: Lucid, 2010]. The installation used the default settings and no additional packages were installed, to replicate a typical user configuration. The virtual machines created were saved and used as baseline machines; each experiment used a virtual machine cloned from a baseline virtual machine. This enabled any changes in the data to be compared with the original configuration. For some experiments, experimental virtual machines were compared with each other as well as a baseline image, for example when looking at how modifying a file affects the data in the thumbnail cache.

The virtual disk images are examined using a hexadecimal editor, and are also mounted to enable the examination of the live system files. The results are also compared with data available on the operating systems, such as source code, the open desktop thumbnail cache specification and Ubuntu forum posts. To permit the automated extraction of artefacts from the thumbnail cache, a Python program will be created to extract any relevant information such as metadata or visual thumbnails which are identified during this research.

5.3 Thumbnail caches

Figure 5.1 shows a screenshot of Nautilus is the file manager used in Ubuntu 10.10, it provides a graphical interface to the system [Ubuntu: Lucid, 2010]. Within the file browser windows, the default setting is to display visual thumbnails of files. However, not all file types display visual thumbnails by default, for example in Ubuntu 10.10 office documents [OpenOffice, 2012]. For files such as PDF and image formats, the thumbnails shown in the browser are of a large enough size to aid the identification of the original file.

The thumbnail cache itself is a hidden directory named *.thumbnails*, which consists of up to 3 files: *fail*, *normal* and *large*; however it is possible for an application or a user to store other data within the *.thumbnails* directory [Ubuntu: Lucid, 2010]. An MD5 hash of the Uniform Resource Identifier (URI) of the original file provides the unique key by which each entry in the cache is named. The subdirectory *normal*, stores standard thumbnail cache records in the form of PNG images with the metadata embedded within each image making up the rest of the entry. The *normal* thumbnail directory contents is defined to be images up to 128 x 128 in size; *large* contains thumbnail graphics up to 256 x 256 pixels in size. The *fail* directory, contains records which are faulty, this may be because the generation of the visual thumbnail was unsuccessful, or due to an error in creating the record.

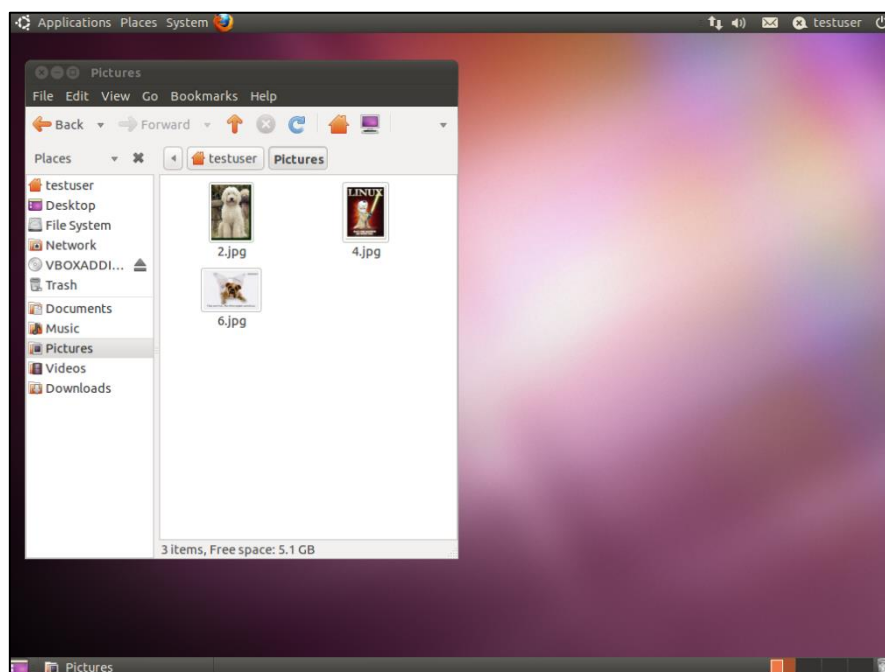


Figure 5-1: Ubuntu 10.10 and the Nautilus File Manager

The directory *.thumbnails* can be found in the user's home directory; therefore each user account has a single centralised thumbnail cache. The directory is hidden, and can contain up to three further directories; *fail*, *large* and *normal*.

5.4 Default Installations

This section describes the artefacts relating to the thumbnail cache which are present on the default installation. An analysis of the artefacts created during installation and their initial state will provide a baseline for comparison with the experiments to be conducted.

After analysis of the baseline virtual machines for both versions it was found that the thumbnail cache is not present. Searches were performed on mounted clones of the baseline virtual disks, to locate the thumbnail cache in the user's home directory, however this proved unsuccessful. A hexadecimal editor was used to conduct searches for 'thumb' and 'thumbnails', and found several python scripts relating to the creation of thumbnails. One interesting result found when searching for 'thumb' was a hit for a file named *ThumbnailFactory.pod*; the main algorithm used for the creation of thumbnail records. It is necessary to know which algorithm is likely to have generated a record to ensure the correct context is applied when extracting data and to ascertain whether the record is valid.

The analysis of the default installation shows that the thumbnail cache is created when it is required by a process and not at the initial installation of the operating system. It also shows that only one algorithm for thumbnail cache generation is installed by default on the operating system; therefore as no additional software will be installed any thumbnails created during these experiments will use the default thumbnail generation algorithm. Any additional thumbnail cache artefacts found during this research will have been added due to the activity in the experiments.

5.5 Identifying the structure

This section identifies the structures used for the operating system thumbnail caches in Ubuntu and Kubuntu. The structures were identified through adding a selection of user files to the virtual machine's hard disk in an attempt to produce visual thumbnails. Five hundred files were copied from a CD, preserving the directory structures, each operating system had the same files loaded. The directory structure is shown in *Figure 5.2*; a naming convention of [Set of directories | Level of directory | Number of directory on that Level] was applied. The structure permitted the evaluation of the behaviour in nested folders. The resulting visual thumbnails were found to be stored in a thumbnail cache; the structures identified in the experiments were analysed to ascertain the thumbnail cache structure.

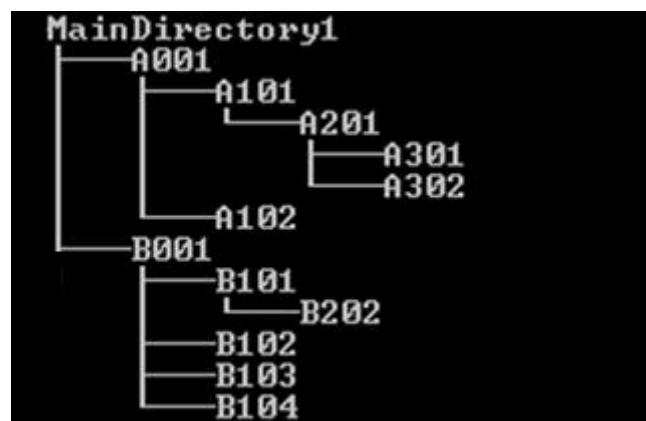


Figure 5-2: Directory structure for experimental test files

5.5.1 The thumbnail cache directory structure

There is generally only one thumbnail cache present on the system which can be accessed by any software as required. During the experiments performed for this research it was noted that the expected thumbnail cache structure shown in *Figure 5.3* was valid.

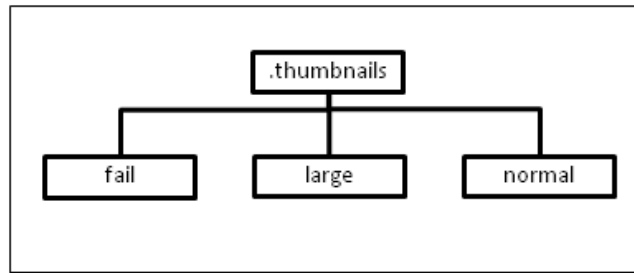


Figure 5-3: The structure of thumbnail caches in Ubuntu and Kubuntu

The hexadecimal below was extracted from an Ubuntu operating system and shows the directory information for *.thumbnails*. In this example, the directory contains a single sub-directory named *normal*. The *normal* directory holds subrecords with visual thumbnails up to 128x128 pixels in size. Up to two further directories may be found in *.thumbnails*: *large* and *fail*. The two further directories do not exist because they are only created when they are required by the system.

```

08219B000   F5 03 02 00 0C 00 01 02   2E 00 00 00 1F 01 00 00   ð      .
08219B010   0C 00 02 02 2E 2E 00 00   F6 03 02 00 E8 0F 0602   ..  ö  è
  
```

06 - Number of characters in text string

02 - Flag type (Determines if it is a file or directory)- A directory

```

08219B020   6E 6F 72 6D 61 6C 0000   00 00 00 00 00 00 00 00   normal
  
```

6E 6F 72 6D 61 6C - Text string denoting name of entry- "normal"

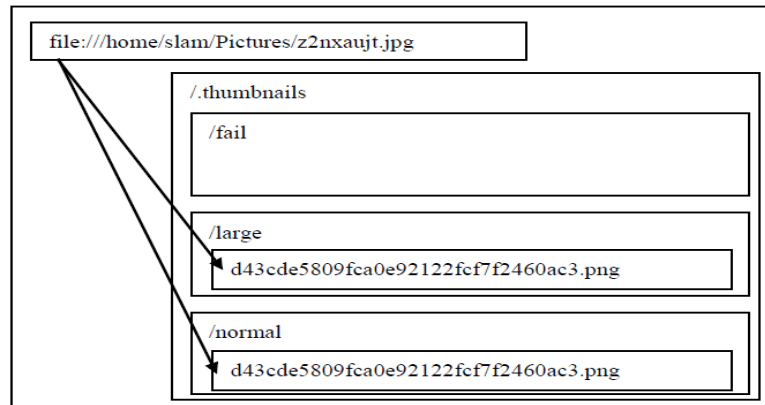


Figure 5-4: A figure showing the thumbnail cache subrecords which make up the complete record for the file z2nxaujt.jpg

The thumbnail cache stores records; a *record* can be defined as a complete set of data relating to an original file. In the case of Ubuntu and Kubuntu, a thumbnail cache record can be defined as the set of files located in *.thumbnails* or any of its subdirectories which have the same filename and extension as each other. A subrecord for these thumbnail caches can be defined as a single file stored in the thumbnail cache. These files are stored in PNG format and can contain metadata about an original file and a visual thumbnail.

If a file existed at the path “file:///home/slam/Pictures/z2nxaujt.png”, and had a visual thumbnail stored in both the large and normal sub-directories, the record for the original file would look like *Figure 5.4*. In the example the record consists of two subrecords; the subrecords form a complete record of thumbnail cache information about the source file.

The following piece of hexadecimal is taken from offset 0x08219C020; it is part of a directory listing for the *normal* sub-directory. The directory contains both standard and incomplete subrecords. An incomplete subrecord is one which is in the process of being completed and therefore information may be missing;

these can be identified by the addition of six further characters after the standard subrecord filename and extension.

```
08219C020  35 34 32 31 66 66 66 65 39 37 32 66 31 31 37 335421fffe972f1173
08219C030  64 30 32 32 33 36 32 34 31 34 33 64 66 62 31 31d0223624143dfb11
08219C040  2E 70 6E 67 2E 57 4E 42  51 33 55                               .png.WNBQ3U

      ASCII  text  string  containing  name  of  an  incomplete  subrecord
      file(5421fffe972f1173d0223624143dfb11.png.WNBQ3U)
```

The filename is the concatenation of an MD5 hash of the source files URI and the extension '.png'. The extra extension '.WNBQ3U' shows this file is incomplete.

```
08219C050  2C 00 240138 39 63 66  38 63 38 31 30 33 34 65, $ 89cf8c81034e

      24 - Size of text string containing the name of the file - 36

      01 - Flag for type (shows if the entry for a file or directory) - file

08219C060  34 31 30 39 34 63 63 64  66 30 62 38 36 62 33 3341094ccdf0b86b33
08219C070  38 63 62 30 2E 70 6E 67  F8 03 02 00 2C 00 24 01 8cb0.pngø  , $

      ASCII      text      string      containing      name      of      file
      (89cf8c81034e41094ccdf0b86b338cb0.png)
```

This file entry is for a complete subrecord as there is no extra extension. Therefore when examining the storage media after each experiment it is possible to distinguish between thumbnail cache sub-records which are complete and those which are not finished. This can assist with an investigation as a sub-record which does not have the extra 6 letter extension should be complete and comply with the expected structure and syntax; an incomplete file may not have all the information and may not be structurally valid.

5.5.2 The structure of a thumbnail Cache subrecord

A thumbnail cache subrecord is stored in the standard PNG format, which is an open specification [Finke, 2004]. The structure can be broken down into a series of 'chunks', each of which has a defined structure.

Each subrecord consists of the following chunks:

A header signature

IDHR (Header chunk)

SBIT

tEXt

IDAT

IEND

A chunk consists of:

4 bytes denoting the size of the data section,

4 bytes denoting the chunks type,

Variable Length data section,

A 4 byte CRC value

Each subrecord has a PNG file extension; the filename is an MD5 hash of the URI of the original source file. The header of a subrecord is in the standard PNG format, each file consists of a single image; therefore each sub-record contains one visual thumbnail. This is followed by the *SBIT* chunk, which defines the significant bits of the colours; generally this is set to 8bit, non-interlaced, with full alpha transparency.

The *tEXt* chunks contain the metadata on the original file and the visual thumbnail; there are usually several *tEXt* chunks in each file. Each *tEXt* chunk holds a single piece of metadata in the form of a key and a value; the key specifies the meaning of the value. The keys may be inbuilt into the PNG format, specific for the thumbnail cache or for the file type. Thumbnail cache specific keys start with 'Thumb', all keys are written in ASCII, and are clearly defined making it simple for a forensic examiner to interpret them; for example 'Thumb:MTime', is the key for the modification date of the source file. The modification time and URI of the source file are generally present in all subrecords. Other information, such as the dimensions of the original file and the number of pages it contained may also be present.

The visual thumbnail itself is stored within one or more *IDAT* chunks, which are compliant with the PNG specification. Finally each subrecord file ends with an *IEND* chunk, which denotes the end of a file.

The following is an example thumbnail cache record viewed in a hex editor; it has been annotated to show the structure and meaning of each byte.

Start of PNG file

```
00000000  89 50 4E 47 0D 0A 1A 0A
          89 50 4E 47 0D 0A 1A 0A - PNG header
```

This is the PNG file signature, it is 8 bytes long. It shows that the remainder of the file is in PNG format, conforming to the chunk structure described above. The signature also shows the file contains a single PNG image, which means that only one visual thumbnail is stored in this file.

Start of IHDR Chunk

The IHDR contains information on the format of the visual thumbnail, including the size in pixels. This information is generally standardised for each thumbnail cache generation algorithm. For analysts the width and height of the image can be used to validate the visual thumbnail stored in the sub-record. The characteristics of the PNG image such as: bit depth, colour type, compression method, filter method and interlace method are standard PNG fields which hold information about how the visual image is processed and stored within the file. This information may assist with identifying characteristics of visual thumbnails generated with different algorithms.

```
00 00 00 0D 49 48 44 52 %PNG IHDR
00 00 00 0D - number of bytes in the chunks data field - 13
49 48 44 52 - Type of the chunk - IHDR - Image header
00000010 00 00 00 80 00 00 67 08 02 00 00 00 47 1D DD € g G Ý
00 00 00 80 - Width of image in pixels - 128
00 00 00 67 - Height of image in pixels - 103
08 - Bit Depth - 8
02 - Colour type - True colour (Each pixel is an RGB triple)
00 - Compression Method used to compress the PNG image data
00 - Filter Method (pre-processing applied to image before compression)
00 - Interlace Method
00000020 A6
47 1D DD A6 - CRC Value for Chunk
```

End of IHDR Chunk

Start of sBIT Chunk

The sBIT Chunk shows the bit depth, throughout this research, only values of '08' for each colour were recorded. This is another standard PNG chunk and

holds information about sample depths; it can be used to recover the original image data if the file becomes corrupt.

```
00 00 00 03 73 42 49 54 0808DB E1 4F E0 | sBITÛáOà
00 00 00 03 - number of bytes in the chunks data field - 3
73 42 49 54 - Type of the chunk - sBIT - Significant Bits
08 - Significant Red Bits
08 - Significant Green Bits
08 - Significant Blue Bits
DB E1 4F E0 - CRC Value for Chunk
```

End of *sBIT* Chunk

Start of *tEXt* Chunk

This *tEXt* chunk provides the width of the visual thumbnail in pixels. The text chunks store information in key pairs, the first string describes the content and the second string holds the value. In this case the key is “Thumb::Image::Width”. The key starts with ‘thumb’ which shows it is a thumbnail cache specific key; this key holds the width of the original image. The value stored in the second string is 787. This information could assist an analyst by providing characteristics of the original source file.

00000030 00 00 00 17 74 45 58 74 54 68 75 6D 62 3A 3A 49 tEXtThumb::I

00 00 00 17 - number of bytes in the chunks data field - 23

74 45 58 74 - Type of the chunk - tEXt - Textual Data

00000040 6D 61 67 65 3A 3A 57 69 64 74 68 00 37 38 37 BE image::Width 787¾

Keyword - Thumb::Image::Width - Thumbnail specific keyword relating to the image width

00 - Null Separator

37 38 37 - Text String - 787

00000050 50 98 74

BE 50 98 74 - CRC Value for Chunk

End of tEXt Chunk

Start of tEXt Chunk

This tEXt chunk provides the height of the visual thumbnail in pixels

00 00 00 18 74 45 58 74 54 68 75 6D 62 P~tEXtThumb

00 00 00 18 - number of bytes in the chunks data field - 24

73 42 49 54 - Type of the chunk - tEXt - Textual Data

00000060 3A 3A 49 6D 61 67 65 3A 3A 48 65 69 67 68 74 00 ::Image::Height

Keyword - Thumb::Image::Height

00 - Null Separator

00000070 36 33 36 B8 90 4B 43

36 38 36 - Text String - 636

B8 90 4B 43 - CRC Value for Chunk

End of tEXt Chunk

Start of *tEXt* Chunk

This *tEXt* chunk provides the URI of the original file to which this record relates, in ASCII text

```
00 00 00 2E 74 45 58 74 54    636, \KC    .tEXtT
```

0 00 00 2E - number of bytes in the chunks data field - 46

74 45 58 74 - Type of the chunk - tEXt - Textual Data

```
00000080  68 75 6D 62 3A 3A 55 52 490066 69 6C 65 3A 2Fthumb::URI file:/
```

Keyword - Thumb::URI

00 - Null Separator

```
00000090      2F 2F 6D 65 64 69 61 2F  4B 49 4E 47 53 54 4F4E      //media/KINGSTON
```

```
000000A0      2F 75 6E 74 69 74 6C 65      64 2E 62 6D 70 88 59 D0      /untitled.bmp^YD
```

Location of original file - file:///media/KINGSTON/untitled.bmp

000000B0 *F0*

```
88 59 D0 F0 - CRC Value for Chunk
```

End of *tEXt* Chunk

Start of *tEXt* Chunk

This chunk stores the last modification time of the original source file

00 00 00 17 74 45 58 74 54 68 75 6D 62 3A 3A ð tEXtThumb::

00 00 00 17 - number of bytes in the chunks data field - 23

74 45 58 74 - Type of the chunk - *tEXt* - Textual Data

000000C0 4D 54 69 6D 65 00 31 32 32 39 34 39 38 38 32 32 MTime 1229498822

Keyword - Thumb::MTime

00 - Null Separator

Last Modification Time - 17/12/2008 07:27:02

000000D0 09 40 36 5F

09 40 36 5F - CRC Value for Chunk

End of *tEXt* Chunk

Start of *tEXt* Chunk

This *tEXt* chunk stores the software used to create the thumbnail cache subrecord. In this instance, it was the Thumbnail Factory, this is the most common method used in Gnome. An analyst can use this information to replicate user activity using the same thumbnail cache generation algorithm to identify actions which may have led to the artefacts current state.

00 00 00 20 74 45 58 74 53 6F 66 74 @6_ tEXtSoft

00 00 00 20 - number of bytes in the chunks data field - 32

74 45 58 74 - Type of the chunk - tEXt - Textual Data

000000E0 77 61 72 65 0047 4E 4F 4D 45 3A 3A 54 68 75 6D ware GNOME::Thum

Keyword - Software

00 - Null Separator

000000F0 62 6E 61 69 6C 46 61 63 74 6F 72 79 6F 44 55 E0bnailFactoryoDUà

Text Sting - GNOME::ThumbnailFactory

6F 44 55 E0 - CRC Value for Chunk

End of tEXt Chunk

Start of IDAT Chunk

This chunk holds the visual thumbnail, in this case, the image is held in a single chunk. In larger visual thumbnails the image data may be held in several IDAT chunks.

00000100 00 00 1B 7D 49 44 41 54 78 9C ED 9D 59 8C 1C C9}IDATxœí\YGEÉ

00 00 1B 7D - number of bytes in the chunks data field - 7037

49 44 41 54 - Type of the chunk - IDAT -

00000110 99 98 FF 88 8C 3C EB 3E BBBB BA FA BE 48 36 49 ™~ý^€<ë>»»°ú¾H6I

... Visual thumbnail ...

00001C80 01 DE 00 5E 47 A8 BA 31 9A

A8 BA 31 9A - CRC Value for Chunk

End of IDAT Chunk

Start of *IEND* Chunk

This is a standard chunk which signifies the end of the file. Therefore all the data for the sub-record should be contained above this point in the file.

```
00 00 00 00 49 45 4E  P ^G""1š  IEN
00 00 00 00 - number of bytes in the chunks data field - 0

00001C90  44 AE 42 60 82  D@B`,
49 45 4E 44 - Type of the chunk - IEND - Image Trailer
AE 42 60 82 - CRC Value for Chunk
```

End of *IEND* chunk

End of PNG file

5.6 Identifying the behaviour

The previous section identified the structure of artefacts stored in Ubuntu and Kubuntu thumbnail caches. This section presents the experiments performed to identify the user behaviour which causes data in the thumbnail cache to be altered. Alteration to the thumbnail cache can be done in three ways: addition, modification or deletion of data. In order to establish when the three types of alterations to the cache occur, experiments were conducted to add files to the system, modify existing files and finally remove files. All files involved in the experiments were user created files, as these are the most likely to be added to the thumbnail cache.

5.6.1 The creation of subrecords

In order to establish when records are created, user files are added to the system. In this sub-section, four methods of adding files were used: USB stick, CD, downloading from email, and creating a file using pre-installed software. In

order to establish if the software used to view the files affects the thumbnail cache, various methods of viewing files were also tried. This sub-section ends with a summary of the user activity which leads to information being added to the thumbnail cache

Using different methods for adding files

Using each of the four methods described above, a set of 10 user generated files were added to the system; the files were: copied from a USB stick, copied from a CD, downloaded from an email, and created using pre-installed software. For each experiment the user generated files were saved to the user's documents folder and then viewed using the file manager to view the visual thumbnails for each file.

In both Ubuntu and Kubuntu, files from the USB stick and the CD had two sets of entries in the thumbnail cache, as the images were cached when the files were browsed on the external and operating system volumes. The thumbnail cache subrecords relating to the files on the volumes contained the volume name as part of the URI. This may assist an examiner as it can provide extra information about the external media; the additional subrecords may also assist in creating an event timeline. Files downloaded from email, or created by pre-installed software only appeared in each thumbnail cache sub-directory once.

A follow-up experiment was performed for the addition of files using external media, with five hundred user files; the user files were stored in various directories and sub-directories created within the user's documents folder. It was noted that using the file browser to view the files for different lengths of time produced different results; the number of subrecords created in the cache appeared to be dependent on the length of time the directory was viewed in the file browser. This is due to the system putting files into a queue for visual

thumbnail generation; if a directory is no longer being viewed in the directory before a file reaches the front of the queue it is removed as it is no longer required. Therefore visual thumbnails and their associated records are only created and modified for the directory whilst they are currently required by the system.

Viewing the files using different methods

An experiment was conducted where 500 JPEG files were copied using the terminal, from a USB stick to the user's documents directory. The documents' directory was then viewed for less than 5 seconds and then closed. The hard disk was then analysed and less than a fifth of the files (89) within the directory had associated records within the thumbnail cache. During the experiment only 16 files were seen in the file manager; however the number of records created in the thumbnail cache was significantly larger than this. A second experiment repeated the method but left the file manager open for 10 minutes. After the second experiment there were records in the thumbnail cache for each of the 500 files; 15 of these records were in the *fail* directory. The results of these experiments show that when the file browser is used, any file which is eligible to have a record in the thumbnail cache is added to the thumbnail generation queue; however files are removed from the queue if the directory is no longer being viewed in the file browser.

A file is considered eligible when the file type is associated with a method which utilises the thumbnail cache; the file type is checked and if a thumbnail method exists for that type then the system uses the method to create or retrieve a visual thumbnail. Experimentation has shown this can lead to some files within a directory not added to the thumbnail cache. For example files created in Microsoft Excel did not create visual thumbnails, without the installation of additional software.

Each piece of software installed by default and listed in the Applications menu on the taskbar on Ubuntu and Kubuntu was examined to ascertain if user generated files could be opened or saved within them. Where this was possible several directories were viewed and the information added to the cache was noted. Each of the directories had not been viewed previously, which assisted in identifying which software used the thumbnail cache.

In Ubuntu, subrecords are generally added to the *normal* directory, with each potential source file viewed in the file browser added to the thumbnail queue. Subrecords may not be created or updated if a directory contains a substantial number of files, which are potential thumbnail cache source files, and the directory is no longer being viewed in the file browser. This is because the files are taken out of the thumbnail cache queue when the directory is no longer being viewed. By default, potential thumbnail cache source files for Ubuntu are standard image formats and PDFs.

Within Ubuntu, GIMP, a pre-installed graphics program can initiate the creation and update of subrecords to the *normal* directory when the user selects 'open image', and potential thumbnail cache source files are viewed as directories are being viewed. F-Spot, a photo management application, also causes subrecords to be updated and created when importing files; however these subrecords are only added to the *large* directory.

In Kubuntu, both the *large* and *normal* directories are frequently used; therefore a greater amount of information is available to a forensic examiner, than that provided by Ubuntu. When the file browser is open, any potential source files are added to the thumbnail queue for both the *large* and *normal* directories. If there is only a single potential source file in the directory, only a *large* record is added on the first viewing. If, within the file browser, there are directories that contain potential source files, these may also be added to the thumbnail queue.

The information panel within the file browser is the reason for the creation of subrecords in the *large* directory, which enables the file to be previewed in a larger size.

In Kubuntu, the image viewer Gwenview adds files to the thumbnail queue, when browsing directories in the main window of the application; however it only uses the *normal* thumbnail cache directory, and therefore the *large* directory is not modified.

Within Kubuntu, it is interesting to note that there are circumstances where thumbnail subrecords are unexpectedly created. Partial subrecords within the cache are automatically generated for a selection of files in the first level sub-directories of a directory being viewed in the file browser. An example is shown in *Figure 5.5*, which shows the Downloads directory being viewed in the file browser. When this directory is opened, each sub-directory is represented by a series of small thumbnails, such as those shown for 'Test Folder'. These small thumbnails represent files present in the sub-directory, and to create them, the system creates partial thumbnails of files within the sub-directory. If the sub-directory is hovered over with the mouse, the mini-thumbnails rotate through any files within the sub-directory that have thumbnail generation methods.

Therefore from each first level sub-directory in a browsed folder, there may be subrecords created for up to the number of files which have thumbnail generation methods, even though the user has not browsed the sub-directory. Interestingly, partial thumbnail subrecords have two properties which allow them to be distinguished from subrecords created using standard methods. Firstly, there is a noticeable lack of metadata within the PNG, including no information on the last modified time and URI for the source file. This means there are two main methods of forming a relationship between a partial subrecord and the source file; the image stored in the subrecord and the MD5 hash of the URI

which is present in the subrecords filename. The second difference in partial subrecords is noticeable in the *IDAT* chunks, where upon comparison with a standard subrecord entry for the same file, it is noticeably shorter as the alpha values are missing.

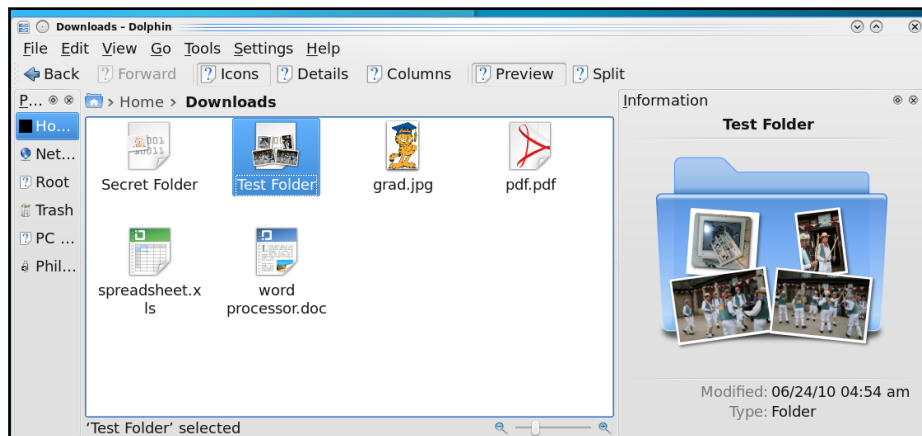


**Figure 5-5: The directory 'Downloads' being viewed in the file browser within
Kubuntu**

If within Kubuntu, the *large* thumbnail cache directory is viewed, the files within it are added to the thumbnail queue for the creation and modification of subrecords within the *normal* directory. These subrecords do not conform to the standard structure; instead they use the same structure as the partial subrecords. Potential thumbnail source files in Kubuntu by default are only image formats. Interestingly, thumbnails for text files are not available in the file browser. If a text file is selected within the file browser, a visual thumbnail is rendered in the information panel, but this is not stored in the cache.

In both operating systems, potential source files viewed on the desktop are added to the thumbnail queue for subrecords to be added or modified in the *normal* directory only. However if the desktop is viewed in the file browser in Kubuntu, subrecords may also be created or modified in the *large* directory.

Circumstances where information is added to the thumbnail cache

A subrecord in the thumbnail cache is created when a file is viewed in thumbnail view, in a browser window, at a URI, for which no subrecord currently exists in the *normal* or *large* directories.

A new record can be created under the following circumstance if the following 3 criteria are met:

Case 1: No record available

- A directory is viewed using the file browser or the file browser within an application when it is in thumbnail mode
- The directory being viewed contains file types which are compatible with the thumbnail cache
- A file exists in the directory which does not currently have a thumbnail subrecord

Case 2: A subrecord only exists in the *fail* directory if the following 4 criteria are met:

- A directory is viewed using the file browser or the file browser within an application when it is in thumbnail mode
- The directory being viewed contains file types which are compatible with the thumbnail cache
- A file exists in the directory which only has a subrecord in the *fail* directory
- The file is processed as if no record information is available

5.6.2 The modification of subrecords

This subsection looks at the modification of information stored within the cache. In order to study the causes of modification to the thumbnail cache, clones of the virtual machines from **Section 5.6.1** will be used. Therefore the experiments start with data within the thumbnail cache; in order to assess how this data is modified the original source files will be modified during a variety of experiments described in this sub-section.

Modifying the contents of a file

A randomly selected group of files on the virtual machine were altered using appropriate pre-installed software. A list of the 500 user-generated files added to the virtual machine was stored in a text file. A small program was written to randomly select 10% of the files; the names of the selected files were stored in a second text file. Each of the selected files was opened using the default editing application for that file type; image files were modified by making the images entirely black, for text based file types the information within the file was copied and pasted underneath the original data. After the files were modified they were saved and the editor closed. Once all modifications had been completed the directories containing the user generated images were then viewed using the file manager. The modification caused the last modified time of the file to be updated. Both on Ubuntu and Kubuntu, this caused the affected subrecords to be altered, to show the current state of the file; all the files showed the modified black images or repeated text in the file browser.

Modifying the files URI

In order to ascertain the effect of modifying the file's URI, the virtual machine was populated with 500 user generated files which were copied off a USB stick and into the user's documents directory. A directory named 'test' was created

on the user's desktop and 5 sub-directories were created within it. Using the same piece of software described in the previous section, 10% of the files were randomly selected; the 50 selected were then further separated into 5 groups by splitting the file list every 10 entries.

The entries in each group were then moved into one of the five sub-directories of the test folder. The terminal was used to move two groups of files, whilst the remaining three groups were moved using the file manager. The file manager was then used to view the test directory and its sub-directories; each sub-directory was left open for 5 minutes. The virtual machine was then analysed. The results show that if a file is moved to a new URI, then a new subrecord is generated for its new location; unless, in Ubuntu a subrecord already exists for a different file at that URI with the same last modified time. On both operating systems, the subrecord for the files previous URI remains on the system, unless it is overwritten by a new file at that URI. This shows that multiple entries in the thumbnail cache may relate to a single file; the entries may assist with building an event timeline for the file.

The deletion of subrecords

From the tests conducted on Ubuntu and Kubuntu throughout this research, no evidence of the deletion of subrecords was recorded; the 53 virtual machines used during this research did not have any deleted sub-records. A further experiment was conducted by deleting the 500 user generated files on the virtual machine and manipulating the clock on the virtual machine to mimic time passing; clones of the virtual disk were taken at weekly intervals for 16 weeks to identify any changes to the information in the thumbnail cache. The test aimed to identify if the thumbnail cache removed content which had not being used recently. A test was also carried out where a USB stick was mounted on the virtual machines, to enable the creation of thumbnail cache records and then the device was dismounted. The machine was then left for thirty days to

ascertain if unused subrecords would be removed. In both experiments the thumbnail cache was not altered; this shows that information in the thumbnail cache may relate to source files which are no longer on the system.

5.6.3 Fragmentation of the thumbnail cache

The data produced during the experiments documented in **Sections 5.5 and 5.6** was analysed to ascertain the fragmentation rate of the thumbnail caches within Kubuntu and Ubuntu. It was noted that the thumbnail cache is generally in a fragmented state; as each individual file which makes up part of the thumbnail cache is stored in a different location on the disk. This is due to the fact it is made up of individual subrecord files, each of which being a standard file on the system can be placed anywhere. Therefore upon examination thumbnail cache subrecords exist throughout the system. This is interesting for an analyst as it provides a greater chance of recovering some information from the thumbnail cache, as even after deletion the files exist throughout the system and it is therefore unlikely they would all be overwritten unless the user was purposefully trying to do so.

5.7 Identifying traces left outside the thumbnail cache

This section describes the corroborating artefacts identified on the rest of the system during the experiments performed to ascertain the structure and behaviour of the thumbnail cache.

A variety of the experiments described in previous sections involved the use of external media, however no information relating to the thumbnail cache could be found on the external media after use. A further experiment was performed where a USB stick containing 500 images was left plugged into the virtual machine for 7 days, with the file browser open. The resulting hard disk and USB stick were analysed, the thumbnail cache on the hard disk contained

information relating to the files on the USB stick; no information relating to the thumbnail cache could be found on the external media. The volume name for the external media is recorded in any subrecords relating to files stored on it; this volume name can also be found on the virtual hard drive both in allocated and unallocated space, showing that it has been mounted on the system. The information found in allocated space shows where the volume was mounted and can provide information about files present on the disk.

As the subrecords are stored in standard files, their file names are present in directory listings for the thumbnail cache sub directories. There is also information on the original source files in the directory listing and within the journal file which can provide corroboration and extra information about user activity.

5.8 Alteration of thumbnail cache artefacts

This section describes the experiments performed to ascertain what would happen if parts of the thumbnail cache evidence were tampered with. The evidence has been divided into three sections: metadata, the visual thumbnail, and the original source file. Each section of the evidence was altered by viewing the virtual hard disk in a hexadecimal editor and editing the information as appropriate.

5.8.1 Metadata

Each *tEXt* chunk within the subrecords was altered during a series of experiments to determine the effect on the artefacts produced. Once the metadata had been altered, the original source files were then viewed in the file browser to see if the subrecord was altered. In both Ubuntu and Kubuntu altering the last modification time chunk without altering the original source file results in the affected subrecord being updated when the source file is viewed

in thumbnail view. This shows that one check used by the thumbnail cache generation algorithms is the last modification time; therefore an analyst can see the last modification date of the file when it was last viewed in a file browser. Altering the URI caused Kubuntu to update the subrecord even when the last modified time was the same; however Ubuntu only updated the subrecord if the last modification time was not equal to the one stored in the thumbnail cache.

5.8.2 Visual thumbnail

In Kubuntu, altering the *IDAT* chunks of a subrecord resulted in the subrecord being regenerated when the original source file was viewed; therefore showing that the system checks the image is an accurate representation of the current state of the original source file. Therefore the system overwrites the subrecord with an updated and accurate image; this shows that in Kubuntu if a file with a tampered subrecord was viewed a valid subrecord would replace it.

In Ubuntu a curiosity was discovered when attempting to alter the visual thumbnail stored in the *IDAT* section of a subrecord. In order to successfully fool the system into displaying an incorrect thumbnail, all other metadata within the subrecord must be accurate. This includes the last modification time, as discussed previously, which must be exactly the same as the source file, or the subrecord is rewritten. It is also important to ensure that each chunk within the subrecord is accurate. For example, if an existing subrecord was being altered it would be necessary to alter the size of each chunk and the CRC data to ensure they matched the new information. If the subrecord uses a valid PNG structure, and the last modification date is equal to the source file then the new image placed in the *IDAT* section would be viewed instead of a representation of the actual file if the CRC was correct.

However, if this ‘tampered’ record is selected, a new visual thumbnail is shown which accurately represents the file. The tampered record is not, however overwritten in the thumbnail cache. Instead the accurate image is generated when required, by the system and does not appear to be stored anywhere. If the browser is refreshed or reopened, the tampered thumbnail is still displayed until the file is selected; the accurate image is only displayed and not saved in the thumbnail cache.

5.8.3 Original source file

If a new file is placed at a URI with an existing subrecord entry in the thumbnail cache, then both Ubuntu and Kubuntu will update the thumbnail cache entry. However, if the associated last modified time in the subrecord is altered to match the new source file, Ubuntu will not update its subrecord entry. Kubuntu appears to perform a greater number of checks on the information in the subrecords and appears to regenerate a record if the visual thumbnail does not accurately reflect the original source files state. Experiments corroborated this finding, as two identical files were viewed, one of which had a tampered subrecord; only the modification time for the subrecord which had been tampered with was updated, suggesting the subrecords are not simply regenerated on access.

5.9 Forming a relationship between the thumbnail subrecords and the source files

Previous sections have established the requirement to put thumbnail cache artefacts in the context of the user actions which generated them. This section looks at ways in which a relationship between a subrecord and an original source file could be formed.

5.9.1 Metadata

There are three distinct types of metadata contained within a subrecord: the URI of the original source file, information on the original source file, and information about the visual thumbnail. The information about the visual thumbnail can assist in checking if the image is reliable, and can also be used in conjunction with the techniques discussed in **Section 5.9.2** to identify visual thumbnails which are the same. The information within the subrecord relating to the original source file can assist with forming a relationship as it can be used to check the source file has the same properties. Finally the URI can assist with forming a relationship as it provides a location in the directory structure for the original source file. The filename of the subrecord is an MD5 hash of the URI, which can be used to corroborate a relationship between the subrecord and the original source file. Checking the source file's URI along with the metadata stored in the subrecord about the original file can provide a stronger relationship between the file and the subrecord, if the two sources of metadata agree.

5.9.2 Images

Each file type which is eligible for visual thumbnail generation uses a specific algorithm; the generation algorithms and the list of associated file types is open source. Therefore it is possible upon knowing the file type of the original source file to use the appropriate algorithm, which would create a visual thumbnail. The *IDAT* chunks of the created thumbnail can be compared with *IDAT* chunks from existing subrecords to identify any matches. Finding a subrecord with a matching image does not show that there is a relationship between the original source file and the subrecord, as multiple copies of a file may exist. It is also possible that the subrecord was created earlier in the source files event timeline and does not accurately reflect its current status; for example, it may not be at the correct URI.

5.9.3 MD5 hashes

In the previous sections it was established that the filename of a visual thumbnail is an MD5 hash of the URI for the original file. The URI is contained within the PNG file itself. For example an original file with a URI of: file:///media/KINGSTON/untitled.bmp would create an MD5 hash filename of 5421ffe972f1173d0223624143dfb11.png. Whilst it is possible to find a subrecord given the URI of a source file, it is not possible to reverse this process as an MD5 hash is a one way function.

5.9.4 Event timeline

Given a file X, Currently located at the URI Y, the thumbnail cache can assist with reconstructing the event timeline of file X; providing sub-records exist within the thumbnail cache relating to the file. This section describes the criteria developed for determining which subrecords should be added to an event-timeline for a given file.

First, if available the original source file and the associated metadata must be extracted, this forms the basis for determining which records should be added to the event timeline for a file. A thumbnail of the file should then be generated using the appropriate algorithm for the source files type; this can be determined by looking through the source code and establishing which algorithm is associated with the particular MIME type.

The following denotes the order in which subrecords should be added to the event timeline. Each subrecord is contained within a PNG file which has file creation and modification times; subrecords may also contain time information within their internal metadata. The order of adding subrecords ensures that the strongest evidence is added first; this is followed by any supplementary evidence which does not conflict with existing data. At each level listed below,

only information which does not conflict with existing information in the event timeline should be added:

1. If a subrecord exists whose name corresponds to an MD5 hash of the source files current URI, it should be extracted and added to the event timeline. **Section 5.9.3** describes the relationships which can be formed between the thumbnail cache and user files based on MD5 hashes.
2. Any thumbnail records, with a hash value for the *IDAT* chunks that matches the hash value for the original source files thumbnail created should be extracted. **Section 5.9.2** describes the limitations of matching visual data from the thumbnail cache with user created files.
3. Any records which contain a URI which ends with the same file name may then be extracted; the same file name may suggest a previous path of the file or that the file has been copied.
4. Any subrecords which have the same metadata information within them as the metadata contained within a generated subrecord of the original source file can then be added. **Section 5.9.1** discusses the limitations of forming relationships between thumbnail cache artefacts and user files based on metadata.
5. Any subrecords of the same file type may then be extracted for hand analysis by an analyst.

The ability to construct an event timeline based on a single subrecord can assist in establishing the user's behaviour; establishing how the file was manipulated by the user can show their intention and assist with proving or disproving the *mens rea* aspect of the investigation [**Chapter 4**].

5.10 Audio and Media thumbnails

This section briefly discusses audio and media thumbnails, which are the two other types of thumbnail used in Ubuntu and Kubuntu systems.

5.10.1 Audio thumbnails

An audio thumbnail can be heard by a user hovering their cursor over an audio files icon; the audio file then begins to play from the beginning and continues until the cursor is moved away from the file's icon. By default both Ubuntu and Kubuntu are able to play audio thumbnails. An audio thumbnail is generated at run-time and is therefore not stored on the system. A collection of 15 audio files were copied from a USB stick on to the virtual machine and on to the user's desktop. Each of the 15 audio files was then hovered over to activate the audio thumbnail. On analysis of the virtual hard disk it was shown that audio thumbnails do not use the *.thumbnails* cache, and do not have any direct connection to the visual thumbnails. The audio information is played as required and plays the contents of the original file; no audio thumbnail information is stored.

5.10.2 Media thumbnails

Media thumbnails are also generated at run-time by the system and allow the user to view a media file from the beginning by hovering over the media files icon. Whilst the thumbnail is the same size as the file icon, the moving images and associated audio can assist a user in identifying files. Ubuntu and Kubuntu do not, by default, implement media thumbnails; however there are various implementation methods available on the Ubuntu community forum. A collection of 15 media files were copied from a USB stick on to the virtual machine and on to the user's desktop. Each of the 15 media files was then hovered over to activate the media thumbnail. On analysis of the virtual hard drive it was shown that these thumbnails are also generated at run-time and do not store

information in the *.thumbnails* cache either. Visual thumbnails for each of the media thumbnails were found in the central thumbnail cache; it is possible to have a visual and a media thumbnail for the same source file. The visual thumbnail represents the first frame of the media file.

5.11 Discussion

In summary, *table 5.1* provides an overview of the main user activity which leads to artefacts being generated within the thumbnail cache, based on a default installation. As can be seen from the table, the reasons for subrecords existing in the two operating systems can be different. Kubuntu, in general provides a greater amount of evidence within the thumbnail cache, as it uses the *large* directory more frequently, which provide larger clearer images of files, which may assist an analyst. It is also evidentially significant that Kubuntu generally has both a *normal* and *large* subrecord for a file viewed in the file browser, which may provide further information about the state of the system. Ubuntu, uses the *normal* directory for the majority of its subrecord storage, and is easier to tamper with; suggesting that artefacts recovered from a thumbnail cache on this operating system would require a greater amount of corroboration from artefacts elsewhere on the system.

In **Chapter 4**, a set of criteria were established to assist in assessing the strength of evidence from a thumbnail cache. These criteria will now be used on the artefacts found on Kubuntu and Ubuntu during the course of this research to establish the strength of evidence recovered.

5.11.1 Provenance of artefacts

This chapter has identified the structure and represented behaviour of the thumbnail cache artefacts in Ubuntu and Kubuntu. An analyst could use the experimental methods described throughout this chapter (for example **Sections**

5.2, and 5.5) to **replicate** the experiments and verify the artefacts presented. The ability to replicate the results based on the descriptions provided in the chapter show the use of **clear documentation**; an analyst would be familiar with this style of documentation. A research paper on this work using the same style of documentation was available to analysts [Morris 2010] and was used in a lecture on thumbnail caches delivered at the Centre for Forensic Computing, Cranfield University in January 2012.

In **Section 5.5** the structures found in thumbnail cache artefacts are identified, this led to the experiments in **Section 5.6** which enabled relationships to be formed between the artefacts and the behaviour which led to their current state. In order to corroborate the information retrieved from the thumbnail cache it was necessary to form relationships with other system artefacts which are in **Section 5.7**. The thumbnail cache artefacts identified during this research are **justifiable** as they are identified using a standard replicable process that enables the verification of the artefacts presented.

The features identified in the thumbnail cache motivate the requirement to identify and provide **corroboration**. For example, if an analyst was investigating claims a suspect had been viewing illegal material and used a file browser, the thumbnail cache would contain a small graphic representation of a source file, and a location it had resided in [**Section 5.5**]. It would also provide information about user behaviour which may or may not have led to the creation of the artefacts in the thumbnail cache. In order to identify the user behaviour associated with the artefacts and verify the information stored in the thumbnail cache it is necessary to form corroborative relationships with other artefacts on the system [**Section 5.6**]. Therefore for an analyst to put the information recovered from the thumbnail cache into context he would need to corroborate the information; this could be achieved by identifying other sources of information about the source file on the system. Whilst Ubuntu relies upon the last modification time for deciding whether to regenerate an image, Kubuntu

uses substantially more checks which make it harder to tamper with the file. Thumbnail cache entries contain CRC checks which can be used to corroborate the contents of each chunk. Finally additional log files such as those related to mounted devices or recently used files can corroborate the information identified in the thumbnail cache by providing additional verification.

Table 5-1: A summary of the user activity which may lead to subrecords being present in the thumbnail cache

		Ubuntu	Kubuntu
Sub-records in <i>/fail</i>		- Sub-record creation halted before creation - The file contained an error - An error occurred during sub-record generation	
Sub-records in <i>/normal</i>	Standard sub-record	- Viewing directories in the file browser - Viewing a directory in GIMP using 'open image'	- Viewing directories in the file browser - Viewing directories in Gwenview
	Partial sub-record		- Viewing a folder for a directory in the file browser - Viewing records in the <i>large</i> directory
Sub-records in <i>/large</i>		- Viewing directories in F-spot	- Viewing directories in the file browser
Sub-records in any directory with an additional six character extension		- Sub-records which are in the process of being generated - The remains of a generated sub-record	

It is necessary to maintain a relationship between the artefacts and the original data to show the thumbnail cache artefacts are ***understandable***

consequences of the method used. For example the research conducted into the reasons subrecords are created in **Section 5.6.1** was broken down into smaller research areas; focused experiments, such as the use of different methods for adding files which is described in **Section 5.6.1.1**, assist in maintaining a relationship between the original data and the artefact. The artefacts identified as a consequence of the user behaviour were documented throughout this chapter and could be applied to artefacts identified during forensic investigations.

Any artefact presented in Court must be shown to have been extracted and interpreted using **a scientific method**. Each artefact identified from the Ubuntu and Kubuntu thumbnail caches can be checked to ensure it has a valid structure and syntax; sub-records use the standard PNG file format. Section 5.8 shows methods in which the thumbnail cache could be tampered with; however the remainder of the system would also require editing to ensure that all relationships between artefacts on the system remained valid. The artefact can provide a set of data which may also assist with identifying or disproving a type of associated behaviour. The ability to corroborate the artefacts with other system data can provide further validation of the interpretation of the data which is presented to the Court.

5.11.2 Interpretation of results

Thumbnail cache artefacts can provide information on potential user activity; in order to put the artefact in the context it is necessary to establish the potential system activity which led to its current state. Therefore the analyst may need to defend that the artefact is a result of the system **working properly**. Some of the thumbnail cache artefacts identified show the creation algorithm which can assist with verifying the artefact and related system artefacts. This research has investigated the manipulation of the thumbnail cache artefacts [**Section 5.8**]; it has shown that parts of the thumbnail cache can be manipulated. However for

any manipulation to appear as standard system behaviour it would be necessary to alter artefacts which have a relationship with the thumbnail cache; further alterations may be required until all the relationships on the system appear to be generated as a result of the system working properly. Given the non-trivial requirement for manipulating the system to ensure all relationships matched the updated information, the task would be difficult and time consuming.

As shown in *Figure 5.4*, a record within an Ubuntu or Kubuntu thumbnail cache may consist of several subrecords; the subrecords in a record are related to the same source file. As shown in **Section 5.6**, there are situations where multiple records may exist for the same source file; this shows that ***relationships*** can be formed between artefacts within the thumbnail cache. Artefacts on the rest of the system may also have a relationship with the information on the thumbnail cache, by providing both corroborating and supplementary information as shown in **Section 5.7**. As identified in **Sections 5.5-6**, it is possible to determine and eliminate likely user behaviour based on the metadata in the subrecords; however by building relationships with other data within the thumbnail cache and the rest of the system it is possible to minimise the potential types of user behaviour an analyst has to investigate the likelihood of occurring.

In order to ***maximise the corroboration*** of thumbnail cache artefacts it is necessary to continue to build relationships with other sources of information until all the relationships between the artefacts involved are identified. Relationships between the thumbnail cache and other artefacts on the system are documented in **Section 5.7**; relationships between the information in the thumbnail cache and the source files are documented in **Section 5.9**. This permits the identification of multiple sources of corroboration and in turn permits the corroboration of those artefacts, enabling the analyst to identify any sources which invalidate the corroboration and investigate their potential meaning.

In Ubuntu and Kubuntu the thumbnail cache is stored in a centralised directory for each user; therefore in order to ***maximise the data*** recovered from the live thumbnail cache it is necessary to extract and examine the thumbnail cache directory. The carving of data relating to these thumbnail caches from unallocated space is discussed in **Chapters 7-9**.

The structure and syntax identified in **Section 5.5** permit the automation of extracting the data and the identification and reassembly of potential file fragments from unallocated space; a tool could be created based upon the identified structure and syntax for file carving.

5.12 Conclusion

This research has shown the structural implementations of the operating system thumbnail caches used by Ubuntu and Kubuntu. Whilst both thumbnail caches have multiple sources of corroborating artefacts their thumbnail cache and the subrecords contained within them, this research has highlighted the significant differences in context between artefacts recovered from the operating systems. Contextual analysis of an artefact can show the user activity which may have led to the artefact being on the system; it has been shown that the availability of corroborating artefacts can have a significant impact on the strength of the evidence. It has also been shown that the Ubuntu thumbnail cache performs less stringent checks on the subrecords to determine if they require updating and therefore it is easy to tamper with the subrecords and mislead a user or an analyst. Audio and media thumbnails do not use the thumbnail cache, however it was determined that a media thumbnail may also have a visual thumbnail.

Finally after an analysis of the research conducted it was determined that given a live cache, Kubuntu can produce stronger evidence from the thumbnail cache;

highlighting the need for understanding the context of any artefacts recovered from a system. This chapter has shown some opportunities for maximising the recovery of evidence but it has not addressed the potential recovery of artefacts from unallocated space; the identification of thumbnail cache data in unallocated space is discussed further in **Chapters 7-9**. The remaining criteria can be met by artefacts from the Ubuntu and Kubuntu thumbnail caches; the scientific based method used to extract thumbnail cache artefacts and related system data provides a wide range of information about system behaviour.

6 The structure and behaviour of the Windows 7 Operating System Thumbnail Cache

6.1 Introduction

The release of Windows Vista prompted speculation from Forensic analysts about the impact of the new Operating system on investigations [Hargreaves, 2008]. One interesting change was the move away from the directory specific *thumbs.db*, to each user having a single centralised thumbnail cache. The release of Windows 7 in 2009 prompted further speculation from analysts; additional changes were introduced to the structure and behaviour of system components such as Windows Desktop Search [Chivers, 2011]. The centralised thumbnail cache format introduced in Windows Vista had also been adapted in the new operating system.

Microsoft operating systems are installed on a significant proportion of user machines and it is therefore unsurprising that currently a significant proportion of forensic analysis involves their operating systems. Due to the likelihood of encountering a Windows based operating system there is significant interest in finding and understanding relevant artefacts within the forensic community.

This Chapter identifies the structure and behaviour of the Windows 7 operating system thumbnail caches; it begins with the methodology employed within this chapter [Section 6.2]. This is followed by the identification of relevant artefacts in a baseline installation of Windows 7 [Section 6.3]. Section 6.4 identifies the structure of the thumbnail cache components. In Section 6.5 the behaviour of the thumbnail cache is identified through experimentation, this is followed by a review of artefacts related to the thumbnail cache which are present on the operating system [Section 6.6]. Section 6.7 looks at the effect of tampering with data contained within the thumbnail cache; Section 6.8 looks at forming a

relationship between information contained within the thumbnail cache and other evidence present on the system. The use of audio and visual thumbnails in Windows 7 is discussed in **Section 6.9**; this is followed by a discussion of the strengths of the thumbnail cache artefacts identified during this research [**Section 6.10**]. Finally the conclusion of this Chapter is presented in **Section 6.11**.

6.2 Methodology

In order to evaluate the structure and behaviour of the thumbnail caches used in Windows 7 it is necessary to establish the methodology which will be implemented during this research. This section describes the general methodology used throughout this chapter; specific experimental methodology is described as appropriate throughout the chapter.

A critical review of existing resources relating to Windows operating system thumbnail caches was conducted in **Chapter 2.4**; this review enables any methodology created for this research to take into account existing knowledge on the subject. The related reading in **Chapter 2** identified that originally visual thumbnails were stored in an alternate data stream for the user generated source file. In Windows XP, the directory specific thumbnail cache *thumbs.db* was introduced which stored visual thumbnails and associated metadata in the OLE2 format. Windows Vista introduced a user specific centralised thumbnail cache as its primary method of storing thumbnail information; the structures of the binary files in the centralised cache were altered again with the introduction of Windows 7. Limited information is available in the public domain about the structure or behaviour of the Windows 7 thumbnail cache.

To establish the structure and behaviour of the Windows 7 thumbnail cache it is necessary to perform a variety of experiments; the experiments establish the

way the data on a storage medium changes based upon user activity. The experiments were performed on virtual machines; a base line virtual machine was created with a 30GB hard drive. Windows 7 Ultimate was installed, using an ISO downloaded from the MSDN website [Microsoft, 2012a]. During the installation, the default settings were selected in order to mimic a typical user system. Once the installation was complete the baseline machine was cloned and examined to determine the initial state of the thumbnail cache and related files, this is discussed in **Section 6.3**. A user account was created, logged on to and then the machine was shut down; this ensured the user accounts default settings, directory structure and files were created. A clone of the baseline was created for each set of experiments to assist in determining the structure and behaviour of the thumbnail cache. This enabled the results of any actions taken to be compared with the baseline image to establish what changes occurred. For some experiments, experimental virtual machines were compared with each other as well as the baseline image; for example during file modification experiments.

The virtual disk images were primarily analysed using a hexadecimal editor, and were also mounted in read only mode to enable browsing of the live file system structure. Procmon was used during each experiment to assist with identifying other system artefacts potentially related to the thumbnail cache. To permit the automated extraction of artefacts from the thumbnail cache a Python program was created to extract any relevant information identified throughout the research.

In summary, a baseline 30GB virtual machine was created and a default installation of Windows 7 Ultimate was installed. The baseline machine was cloned to enable the results of the experiments to be compared with the default system installation. The experiments were primary analysed using a hexadecimal editor and a Python program was created to automate the extraction of the artefacts identified during this research. The following sections

document the state of a default Windows 7 Ultimate installation and the experiments performed during this research.

6.3 Default installation

The baseline virtual machine provides a point of comparison for the experiments conducted during the research as it enables the effects of any actions performed to be identified. In order to identify artefacts relating to the thumbnail cache which were present on the baseline virtual machine a series of exploratory examinations were performed. This section details the methodology employed to identify the location of the thumbnail cache and other related sources of potential interest.

6.3.1 Initial Searches

Firstly a series of initial searches were conducted on the hard disk using a hexadecimal editor to identify locations which may require further evaluation. A search for the term “*thumb*” resulted in several hits within the directory `\Users\Sarah\AppData\Local\Microsoft\Windows\Explorer`, when explored it contained 8 files:

- *ExplorerStartupLog.etl*
- *ExplorerStartupLog_RunOnce.etl*
- *thumbcache_32.db*
- *thumbcache_96.db*
- *thumbcache_256.db*
- *thumbcache_1024.db*
- *thumbcache_idx.db*
- *thumbcache_sr.db*

The six thumbcache files identified represent the centralised implementation of the thumbnail cache used in Windows 7. As the thumbnail cache is stored in a user's home directory it can assist in establishing an event timeline for a specific user area; event timeline reconstruction is discussed in **Section 6.8.4**. The search for the term "*thumb*" also showed the existence of a thumbcache.dll file, which appears to assist with thumbnail record generation; the dll was located at C:\Windows\System32.

6.3.2 Initial state of the thumbnail cache

Upon examination of the thumbnail cache it was identified that the *thumbcache_sr.db* only contains a basic header file by default and therefore gives no initial indication of its role within the thumbnail cache. The files *thumbcache_32.db*, *thumbcache_96.db* and *thumbcache_1024.db* also just contain a header; these files are used to store the various sizes of visual thumbnail. The file *thumbcache_idx.db* has a single entry consisting of 32 bytes which is located close to the centre of the file. The structure of this entry is described in **Section 6.4.2**. Lastly *thumbcache_256.db* has been initialised to a size of 1MB, and contains two identical entries; the free space within the file has not been initialised and may therefore contain data of evidential significance. The two identical subrecords within the cache file do not have visual thumbnails, and instead store the GUID: {645FF040-5081-101B-9F08-00AA002F954E} (see **Section 6.4.4**).

6.4 Identifying the structure

In order to extract meaningful information from the Windows 7 thumbnail caches it is necessary to understand their structure; the structure provides a context for the data. This section describes the structure of the various components of the Windows 7 operating system thumbnail cache.

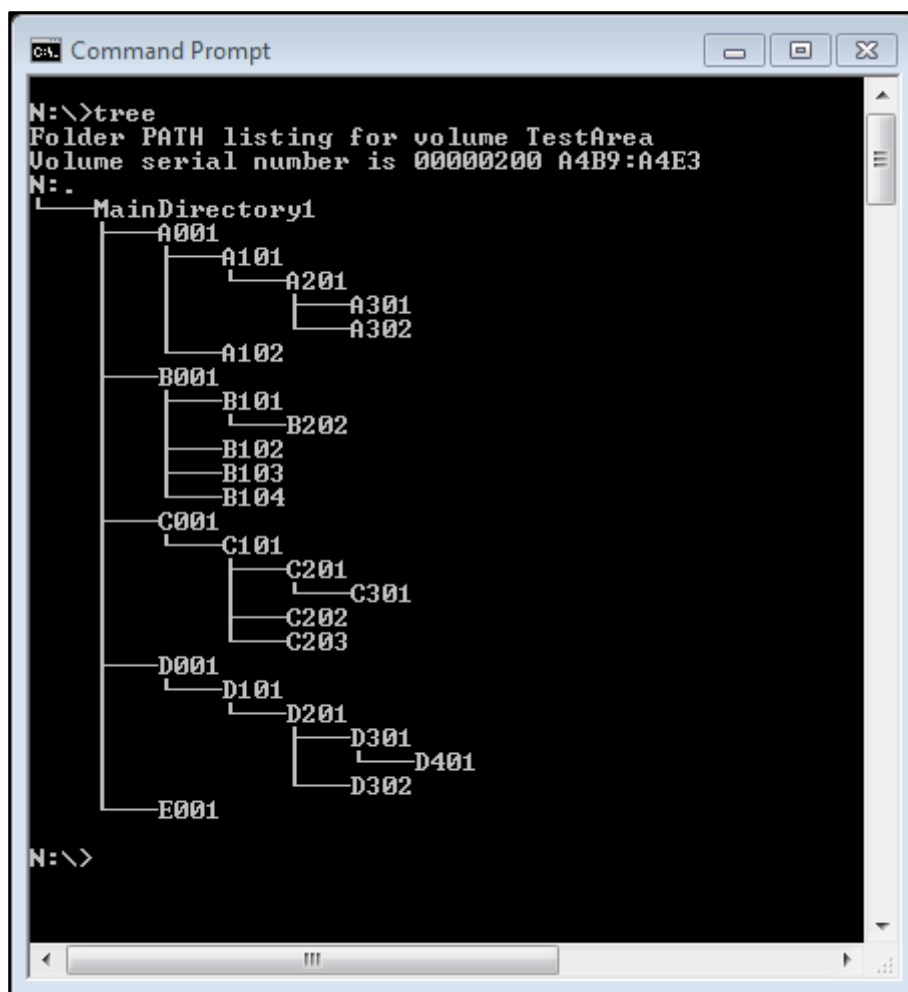


Figure 6-1: Directory structure for experiment test files

6.4.1 Methodology

In order to identify the structure of the Windows 7 thumbnail cache a clone of the Virtual Machine Image discussed in the **Section 6.3** was used. The directory containing the thumbnail cache was extracted from the baseline image, to allow further examination and provide a baseline.

Experiment Set 1 involved the addition of 500 different user generated files to the system. The addition of files to the system aimed to populate the thumbnail cache with data; the data could be used to ascertain the structure and syntax of the thumbnail cache files. In order to ensure a variety of file types were included

10 potential user generated file types were selected; the file types selected were: JPG, BMP, GIF, DOC, DOCX, XLS, PPT, PPDF, MOV and, MP3. For each type 50 files were selected from the corpus of files collated before commencing this research.

A python program was created to randomly select 50 files of each file type; these files were then split between 25 directories. The directory structure is shown in *Figure 6.1*; a naming convention of [Set of Directories] Level of directory | number of directory on that level] was applied. The structure permitted the evaluation of the behaviour in nested folders; directories displayed in thumbnail view may include thumbnails for sub-directories. The main directory 'MainDirectory1' and its contents were copied onto the virtual machine from a USB device using the command prompt. Ten clones of the machine were made in its current state.

- Clone 1-1: Each directory was viewed in thumbnail view for 30 seconds starting with the A directories.
- Clone 1-2: Each directory was viewed in list view only
- Clone 1-3: The A directories were viewed in thumbnail view for 30 seconds each.
- Clone 1-4: Directories on levels 0 and 1 for each set were viewed in thumbnail view for 30 seconds each.
- Clone 1-5: All directories were viewed in thumbnail mode, but for less than 5 seconds per directory.
- Clone 1-6: A single file in each directory was selected with a single click before proceeding to the next directory; at least two of each file type was selected overall.
- Clone 1-7: The previous experiment was repeated with the preview pane turned on.

- Clone 1-8: Directories at level 0 were viewed in thumbnail mode. The preview pane was on and each directory was selected with a single click.
- Clone 1-9: The directory E was viewed in thumbnail mode.
- Clone 1-10: Every directory was viewed in thumbnail mode with the preview panel on; every file was individually selected for 5 seconds with a single click.

Experiment Set 2 used 5 clones of one of the virtual machines created in experiment set 1 (Experiment Set 1: Clone 1). This meant that at the start of this experiment each of the 25 directories had been accessed in Windows Explorer thumbnail view. This set of experiments involved modifying files either by modifying their metadata or contents; the modification of this information allows changes in the data already stored in the thumbnail cache to be identified.

- Clone 2-1: Each file in the A directory set was renamed to start with 'AAA' using the command line; the directories were then viewed in Windows Explorer thumbnail view.
- Clone 2-2: The A directory set was moved to an attached clean USB storage device using the command line and then viewed in thumbnail view in their new location.
- Clone 2-3: As the previous experiment except the files were moved using Windows Explorer.
- Clone 2-4: Keeping the directory structure intact, the files from directory set B were deleted; the directories were then reviewed in thumbnail view.
- Clone 2-5: As per the previous experiment, except only one file from each sub-directory was deleted.

The structures were reverse engineered based upon previous experiments in Windows Vista and the experiments documented above. The two experimental sets permitted the structure of the thumbnail cache to be reverse engineered; this provided the information documented in the following sub-sections. An analysis of the data collected was performed using a hex editor, and by comparing the different thumbnail caches based upon the actions that were taken it was possible to deduce the associated structure. Partial thumbnail cache structures have been presented on some websites [Noxa, 2008; Parsonage, 2012]. In contrast with prior works the present research adopts a systematic methodology including experiment planning and quantitative analysis of efficiency and effectiveness of the existing approaches to thumbnail fragment detection and reassembly. Any structural and syntactical information identified during this research will be added to the thumbnail cache extraction program to enable artefacts to be automatically extracted from Windows 7 thumbnail caches.

This section begins with an overview of the thumbnail cache directory structure; this is followed by an overview of the structures contained within the six files used in the thumbnail cache.

6.4.2 The thumbnail cache directory structure

In **section 6.3** the operating system thumbnail cache implemented in Windows 7 was found to reside in a user's home directory. Windows 7 has a centralised thumbnail cache for each user rather than the directory specific cache system used in Windows XP. This directory was shown to contain six thumbnail cache files: an index and five further files. The index holds records, which provide pointers to the location of subrecords in the remaining files; the structure of the thumbnail cache is shown in *Figure 6.2*.

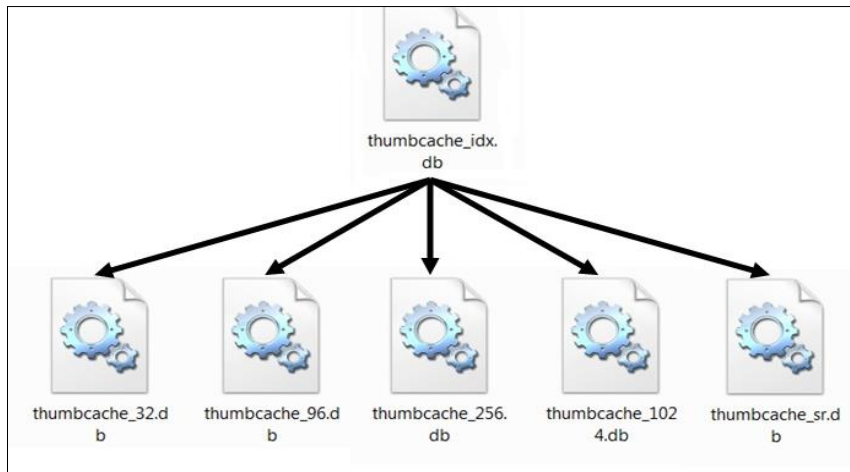


Figure 6-2: The centralised thumbnail cache structure

The rest of this section identifies the structures present within the six thumbnail cache files and identifies the records and subrecords which exist within them.

6.4.3 thumbcache_idx

The file *thumbcache_idx.db* provides an index to the information stored within the thumbnail cache. The index uses a hash table to store the records. At any point in time each source file is represented by a single record in the index; however the source file may have other records in the thumbnail cache which have a relationship with a previous state of the file. A record contains pointers to the locations of the associated subrecords. The file has a standardised structure for records with each one using thirty two bytes; the structure determined during the experiments documented earlier in this section is broken down in *figure 6.3*.

The unique identification string consists of sixteen hexadecimal characters which are known as the *ThumbCacheID*, this string can also be found in the record relating to the original source file within *Windows.edb*. The flags are related to the file type; this was established through matching the records with the original source files. The remaining twenty bytes store pointers to the subrecords for the original source file; each pointer identifies the starting position in bytes from the beginning of the relevant file. If the pointer is set to *0xFFFFFFFF* there is no associated subrecord in the file. This was determined

by extracting the images from each of the thumbnail cache files and noting the first byte of the subrecord; these positions were then compared with values within the thumbnail cache index file.

Thumbcache_idx record structure:

- 8 bytes: Unique Identification string
- 4 bytes: Flags representing file type
- 4 bytes: Pointer to location of subrecord in *thumbcache_32.db*
- 4 bytes: Pointer to location of subrecord in *thumbcache_96.db*
- 4 bytes: Pointer to location of subrecord in *thumbcache_256.db*
- 4 bytes: Pointer to location of subrecord in *thumbcache_1024.db*
- 4 bytes: Pointer to location of subrecord in *thumbcache_sr.db*

Figure 6-3: Thumbcache_idx record structure

The *thumbcache_idx.db* record structure is different to the one used in Windows Vista, which contained records which were forty bytes in size. The structure of Windows Vista records was determined during preliminary research for this thesis using the same methodology. The last modification time of the source file has been removed, which accounts for the eight byte difference in record size.

Within the index file, space for one hundred and one records is allocated at a time; the addition of records is covered further in **Section 6.5** Below is an example annotated record from the Windows 7 Thumbcache_idx.db:

Start of Record

87 F7 6A 62 4B 7B C1 36 02 80 00 08 FF FFFFFFFF 00 B3 05 00

87 F7 6A 62 4B 7B C1 36 - Thumbnail Cache Record ID

02 80 00 08 - *Flags denoting type of file*

FF FFFFFFFF - Offset of subrecord in thumbcache_32.db

00 B3 05 00 - *Offset of subrecord in thumbcache_96.db*

C7 E6 02 00 FF FFFFFFFF

C7 E6 02 00 - Offset of subrecord in thumbcache_256.db

FF FFFFFFFF - *Offset of subrecord in thumbcache_1024.db.db*

FF FFFFFFFF - Offset of subrecord in thumbcache_sr.db

End of record

6.4.4 thumbcache_32, 96, 256 and 1024

The centralised thumbnail cache system implemented in Windows 7 allows for a maximum of four differently sized images to be stored for each thumbnail source file. If the thumbnail cache is used as intended the images stored are up to a maximum of 32x32, 96x96, 256x256 and 1024x1024 pixels respectively. Each of the four sizes has its own file into which subrecords are stored; the structure of a subrecord can be divided into two separate sections: the header and the image. The subrecord header section contains the metadata for the subrecord and has a clearly defined structure. This is followed by the image section which follows the standard structure for the image file type implemented (JPEG, BMP or PNG image formats). The structure for the subrecord header is defined in *figure 6.4*.

Subrecord header:

4 bytes – Subrecord signature

4 bytes – Complete size of subrecord in bytes

8 bytes – Thumbnail cache ID

4 bytes – Size of Identifier value in Unicode, during experimentation only a value of 32 bytes was recorded for visual thumbnail subrecords.

4 bytes – Size of padding after identifier value

4 bytes – Size of the data section of the subrecord

4 bytes – Unknown but during experiments always 00 00 00 00

8 bytes – Data checksum (CRC 64 value)

8 bytes – Header checksum (CRC 64 value)

Figure 6-4: Subrecord header

The Windows 7 subrecord header is 32 bytes, which is 8 bytes smaller than the Windows Vista implementation; this is due to the removal of 8 bytes representing the file extension in Unicode. The Windows Vista structure was determined during the preliminary experiments performed before this research using the same methodology described in this Chapter for the identification of the Windows 7 thumbnail cache structure. The checksum values provide a method of authenticating the values in the subrecord. Below is an annotated example subrecord which was created during this research.

00000100 **FF D8**FF E000 104A 46 49 46 0001 010100 00ÿØÿà JFIF

FF D8 - JPEG file signature

FF E0 - Application use marker

00 10 - Length of APP0 field

4A 46 49 46 00 - Identifier - "JFIF"

01 01 - Version number

01 - Unit - dots per inch

00 00 - Horizontal resolution - 0

00000110 00 0000 00 FF DB 00 43 00 05 03 04 04 04 03 05 ÿÛ C

00 00 - Vertical resolution - 0

00 - Horizontal pixel count - 0

00 - Vertical pixel count - 0

...(JPEG image data continues)

000009D0 EB ED 43 E6 E8 75 D1 8E 05 41 7B 46 EE 7F **FF D9**ëíCæèuŃŽ A{FíÿÜ

FF D9 - JPEG file footer

End of subrecord

The subrecord above shows a standard visual thumbnail subrecord from the *thumbcache_256.db* file; the image data contained within the subrecord is in standard JPEG format. Whilst the thumbnail cache predominately stores visual thumbnails, the *thumbcache_256.db* also holds non-standard subrecords. Below is an extracted subrecord which does not contain an image, instead it stores data in a Unicode format, in this particular case it is a GUID.

```

14F8CE010      43 4D 4D 4D 80 00 00 00      CMMME
      43 4D 4D4D - Subrecord identifier
      80 00 00 00 - Size of subrecord in bytes

14F8CE020      E8 4E B8 F9 51 BC 24 09 50 00 00 00 00 00 00 00      èN,ùQ¼$ P
      E8 4E B8 F9 51 BC 24 09 - Thumbcache ID
      50 00 00 00 - Size of identifier
      00 00 00 00 - Size of padding

14F8CE030      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 - Size of subrecord data
      00 00 00 00 - Unknown but always 00 00 00 00
      00 00 00 00 00 00 00 00 - Checksum for data section

14F8CE040      1D FE 28 91 9F 41 8B 4D 3A 00 3A 00 7B 00 36 00      p('YA<M: : { 6
      1D FE 28 91 9F 41 8B 4D - Checksum for the header section

14F8CE050      34 00 35 00 46 00 46 00 30 00 34 00 30 00 2D 00      4 5 F F 0 4 0 -
14F8CE060      35 00 30 00 38 00 31 00 2D 00 31 00 30 00 31 00      5 0 8 1 - 1 0 1
14F8CE070      42 00 2D 00 39 00 46 00 30 00 38 00 2D 00 30 00      B - 9 F 0 8 - 0
14F8CE080      30 00 41 00 41 00 30 00 30 00 32 00 46 00 39 00      0 A A 0 0 2 F 9
14F8CE090      35 00 34 00 45 00 7D 00      5 4 E }

```

The subrecord above contains the GUID String“: : {645FF040-5081-101B-9F08-00AA002F954E}” in the identifier field within the subrecord header. This GUID is the value used for the recycle bin and relates to the associated icon located on the desktop.

The following two subrecords show the storage of allocated drive letters and available networked machines. Various devices were attached to the virtual machine to ascertain which drive letters were stored within the thumbnail cache. It was determined they are stored for all allocated drive letters which can be seen in the left hand pane of the standard Windows File Explorer, including portable storage devices; this is discussed further in **Section 6.6**. By adding, removing and modifying the names of network locations available to the virtual machine it was determined any network locations available when the Windows File explorer is being used are stored. This includes the name of the machine on which the thumbnail cache has been created.

```
Start of subrecord
00000110          43 4D 4D4D 36 00 00 00          CMMM6
00000120  87 05 EA 36 8A 72 17 75  06 00 00 00 00 00 00 00  + ê6Šr u
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000140  C9 20 C6 9D 50 B9 16 12  43 00 3A 00 5C 00          É Æ P¹  C : \
```

This subrecord relates to the drive allocation C:

```
End of subrecord

Start of subrecord
000001B0          43 4D 4D4D 48 00          CMMMH
000001C0  00 00 F6 48 09 04 67 C9  7F DA 18 00 00 00 00 00  öHgÉ Ú
000001D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001E0  00 00 44 1E BA 3B 95 FA  E2 9D 5C 00 5C 00 53 00  D °;•úâ\u \ S
000001F0  41 00 52 00 41 00 48 00  2D 00 44 00 45 00 4C 00  A R A H - D E L
00000200  4C 00          L
```

This subrecord relates to the network place \\SARAH-DELL

End of subrecord

6.4.5 thumbcache_sr

Whilst this file is present in both the Windows Vista and 7 implementations of the operating system thumbnail cache, experimentation has yet to identify its purpose. The only recorded data for this file is in the standard header.

6.5 Identifying the behaviour

The thumbnail cache can exhibit three kinds of behaviour towards the information stored in it: information can be added to the thumbnail cache, it can be modified or it can be deleted. This section identifies the behaviour the Windows 7 thumbnail cache exhibits when a user interacts with the operating system. For the experiments in this section both Microsoft Office and Adobe PDF viewer were installed on the baseline machine described in **Section 6.3**; the default settings were selected during their installation. The software was installed to allow the artefacts created in the thumbnail cache following the use of common user file formats to be examined during the experiments.

6.5.1 The creation of records and subrecords

When a user interacts with a system, there are several ways in which they can create a new file; for example: they can use an application, they can move a file from a storage device, or they can download a file from the internet. They can also view the files using a variety of methods such as in an application or in a file browser. As discussed in **Chapter 2**, establishing the appropriate *mens rea* occurred is important in legal proceedings; therefore it is necessary to understand how the artefacts came to be in their current state as this may help to establish the suspect's *mens rea*. This sub-section identifies the behaviour which results in records and subrecords being added to the thumbnail cache.

6.5.1.1 Using different methods for adding files

When a file is created in an application such as Paint or Microsoft Word a file can be saved to a storage location, allowing the created file to be accessed again at a later date. An experiment was conducted where 50 files were created on the virtual machines using the installed office applications and the pre-existing software: notepad, and paint. The files were saved using the software

to a variety of locations on the storage device; the thumbnail cache was then viewed to ascertain what changes occurred. It was noted that when a file is created in this way a record relating to the file was not added to the thumbnail cache. The created source files were then viewed in thumbnail view using the file explorer, or a dialogue Window used for opening and saving documents. The analysis of the thumbnail cache showed that viewing the visual thumbnails in this way led to the creation of a thumbnail record for each potential thumbnail cache source file.

Figure 6.5 shows several files being viewed using the standard file browser (Windows Explorer). In the first experiment the directory was opened and the files were viewed without any further interaction; this resulted in each potential user created thumbnail cache source file having subrecords added into the *thumbcache_256.db* and *thumbcache_96.db* file. The system created images such as the one used for the directory 'Not Viewed' were only added to the *thumbcache_96.db*. In the second experiment when the size of the icons used was increased from medium to large icons, the resulting additions to the thumbnail cache did not change. The same directory was then viewed using the extra-large icons; this resulted in the directory visual thumbnail also being added to the *thumbcache_256.db* file.

In the third experiment the file '*Goat.jpg*' was selected using a single click, which led to the file and its metadata populating the preview and information panels. On analysis of the thumbnail cache it was recorded that subrecords were added to the *thumbcache_1024.db* file when a source file is selected and previewed in the preview pane of the file browser. A further experiment was performed with an image file of 1000 x 1000, and a second image of 200 x 200 pixels being selected in the same way; on selection of the files the images populated the preview and information panels. However only the 1000 x 1000 pixel image had a corresponding subrecord added to the *thumbcache_1024.db* file; this suggests a subrecord is only added if the image being previewed is

greater than the maximum size image permitted in the next smallest thumbnail cache file. Throughout the experimentation in this chapter it has been noted that image files, PDF files and directories viewed in thumbnail mode consistently add records to the thumbnail cache; however visual thumbnails are not always stored for Microsoft Office files. This may be due to the addition of a checkbox to enable the saving of a visual thumbnail in the Save as Dialog box in Microsoft Office 2007 as shown in *figure 6.6*.

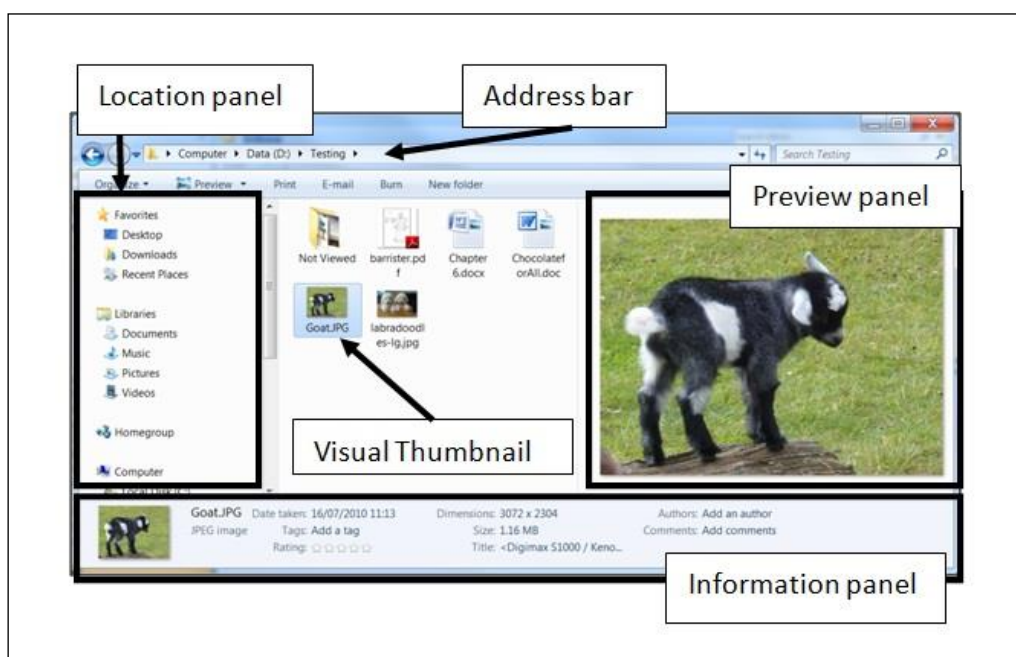


Figure 6-5: The creation of subrecords in thumbcache_1024.db

When files are viewed on moveable storage devices in thumbnail mode, entries are added to the thumbnail cache if the device is indexed by the system. A series of experiments were conducted where a directory containing 100 user generated files were moved from a USB storage device to the virtual machine. In the first experiment the storage device was attached to the virtual machine and the directory was viewed using Windows Explorer, the drive was indexed by default. The directory and its contents were then moved from the USB

storage device to the User's "My Documents" Folder using Windows Explorer; the directory's contents were then viewed in its new location.

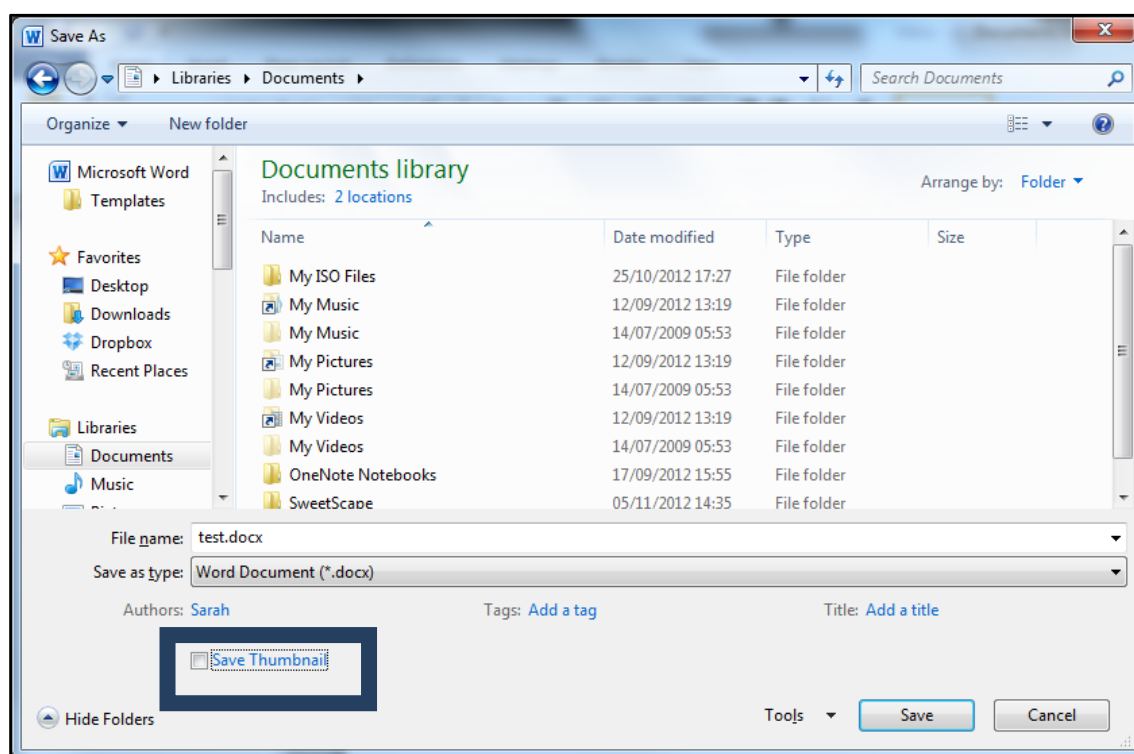


Figure 6-6: Introduction of a save thumbnail option

The results of the experiment have showed that when a file is moved from an indexed storage device to a hard disk drive, 2 records are added for each potential thumbnail cache source file. The first record represents the file on the storage media, the second represents the file on the virtual machine; therefore whilst two records are created only one has a relationship with the current state of the file. The addition of two records based on the same file due to the movement between indexed media may also help with event reconstruction. A subrecord containing the drive allocation letter was also added in the thumbcache_256 file. This shows that thumbnail cache entries for portable media may be present in the thumbnail cache and may provide corroboration that a device has an association with a particular machine or user. When the

experiment was repeated with a non-indexed external storage media records relating to the user files were not added to the thumbnail cache; instead a *thumbs.db* file was created. The subrecord containing the drive allocation letter was still added to the thumbcache_256 file.

In order to ascertain the artefacts present in the thumbnail cache if a user views files in network storage the set of 100 user generated files were stored on a second virtual machine in a shared directory. The machine formed part of the network with the experimental virtual machine; the files in the shared directory were viewed using Windows Explorer from the experimental virtual machine and the results were analysed. The results showed that network storage viewed in thumbnail cache mode does not create entries in the thumbnail cache; however a *thumbs.db* file was created in the shared directory whilst it was being viewed using the experimental virtual machine.

A further experiment was conducted by moving the files from the shared directory onto the experimental virtual machine and into the current user's "My Documents" Directory. The files were moved using Windows Explorer and viewed in their new location. On analysis of the thumbnail cache a single set of records was present for each file viewed. The results of this experiment showed that visual thumbnails of the files were not present in the virtual machine until the files were moved to a local drive; this was due to the network drive not being indexed by the virtual machine's operating system.

In **Section 6.4.1** an experiment was described where a collection of 25 directories were viewed (Clone 1-1); when the thumbnail cache was analysed after this experiment it was noted that the subrecords were stored in the order the directories had been viewed. New subrecords are appended to the end of the file. This experiment was expanded on by repeating the experiment, but leaving 24 hours between viewing each directory. On analysis of the thumbnail

cache it was noted that the subrecords relating to the user generated content viewed did not alter their relative position.

Any subrecords added to a thumbnail cache file are appended to the end of that file; however the index file is a hash table, where entries are assigned a place in the table based on the eight byte thumbnail cache ID number using the following calculation[Personal Communication from H.Chivers, 2010].

$$(2^{32} X + Y) \bmod Z = \text{Position in hash table within } \textit{thumbcache_idx.db}$$

Where

X is the first 4 bytes in little endian of the thumbcache ID

Y is the second 4 bytes in little endian of the thumbcache ID

Z is the number of positions currently in the hash table

The index allocates one hundred and one spaces in the thumbnail cache when it is initialised. A series of experiments were performed where 1000 user created files were added to the virtual machine in sets of 10; each group was then viewed using Windows Explorer. The thumbnail cache was then viewed to ascertain the size and growth of the thumbnail cache. The results showed that for index sizes fewer than five hundred and five records, each time a clash occurs in the hash table, another one hundred and one spaces are added to the table. For larger index sizes the number of new records added each time increases but is always a multiple of one hundred and one. The hash values for all the existing records are then recalculated to account for the new table size.

Any record added to the thumbnail cache contains associated data in the Windows.edb file; the record within the windows.edb contains a thumbcache ID which allows a relationship between a source file and a thumbnail cache record

to be shown [Douglas, 2009]. *Figure 6.7* shows a file 'Cake.JPG', and shows the structure of the relationship between the files.

It is important to note that records within the thumbnail cache may not be directly associated with a visual thumbnail; these records may instead hold information stored in Unicode in the identifier section of the subrecord header. During experiments to determine the behaviour of visual thumbnail subrecords, non-standard subrecords were observed in the *thumbcache_256.db* file, analysis showed these subrecords to contain: GUIDs relating to entries in the Windows registry, the letters associated with allocated drives, and the names of network places.

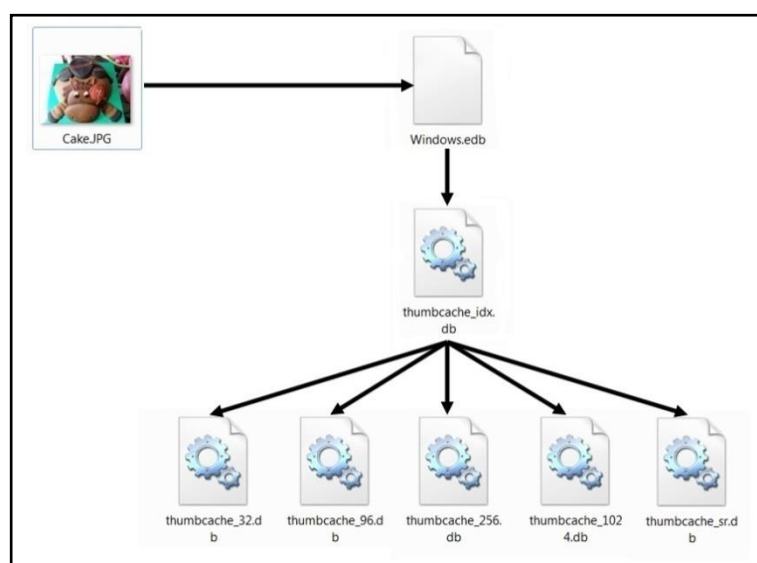


Figure 6-7: The relationship between a file and a record in the thumbnail cache

The thumbnail cache subrecords relating to drive letters and network places are generated when the file browser is used. This was identified by using a clean virtual machine to open Windows Explorer using the Computer shortcut on the desktop; this screen was left open for 5 minutes and then the virtual machine was shut down. The thumbnail cache was then analysed. As can be seen in *Figure 6.8* the right side of the file browser has icons for drives, folder locations

and network places. As these items use default icons, only the standard metadata in the subrecord header is stored in the thumbnail cache. New subrecords are added when locations are added to the right side panel, either by the system or a user and are visible in the file browser.

Subrecords relating to locations which are no longer present do not appear to be removed. This was shown by continuing with the experiment over a 12 week period where 8 devices were originally on the same network as the virtual machine; every 4 weeks a device was removed from the network; Windows Explorer was then viewed to confirm the device was no longer available. At the end of the experiment the thumbnail cache was analysed and subrecords relating to all 8 original devices and the virtual machine were found. This may be a useful source of information for analysts as it can provide a list of drives which have been allocated and machines which have been networked to the users machine; it also provides the name of the users machine as this is shown in the network places list within the panel. Further investigation of drive allocation and network place subrecords is discussed in **Section 6.6**.

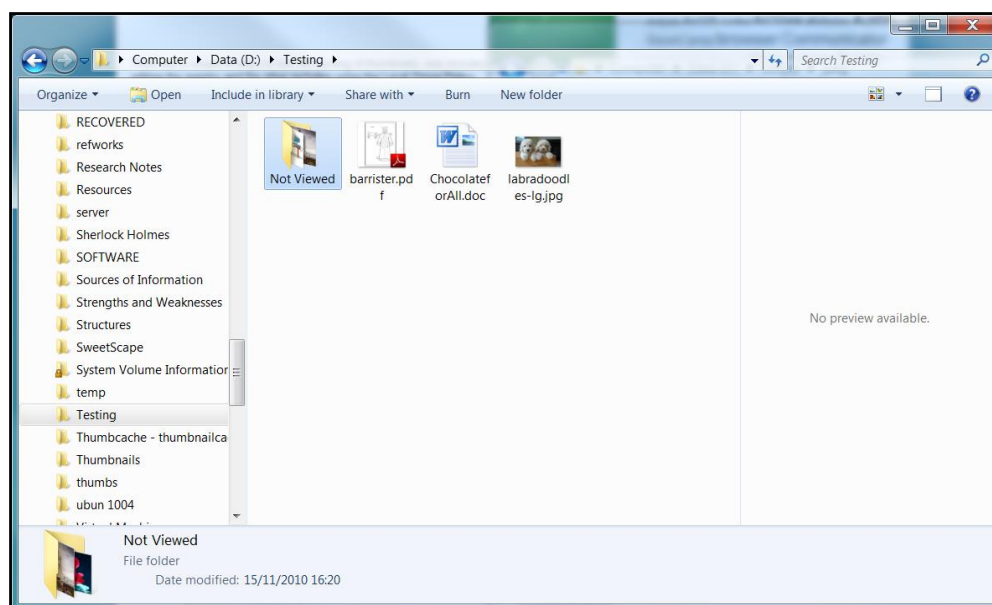


Figure 6-8: A screenshot from a test to see if records were created for files which had not been viewed

6.5.1.2 Circumstances where information is added to the thumbnail cache

A record is added to the thumbnail cache when no matching record currently exists for a given URI and the file at the URI is a potential thumbnail cache source file. For a record to match, the file must have the same last modification time as recorded in Windows.edb.

A new record can be created under the following circumstance:

No matching record found within the current *thumbcache_idx.db*

- A directory is viewed using the file browser, or the file browser within an application using thumbnail mode
- The directory being viewed contains file types which are compatible with the thumbnail cache
- A file exists within the directory which does not currently have a thumbnail cache record, or a subrecord of the appropriate size.

6.5.2 The modification of records and subrecords

A user can modify a file in three distinct ways: by modifying the contents of the file, altering the file's URI, or deleting the file. This sub-section discusses the experiments performed to ascertain the behaviour of the thumbnail cache when the file is modified; **Section 6.5.3** discusses the deletion of files.

6.5.2.1 Modifying the contents of a file

In order to ascertain the behaviour of the thumbnail cache when a user modified a file, the baseline virtual machine created in **section 6.4** for experiment set 1 was cloned. The virtual machine contained 500 files, 20 Microsoft Office files and 10 image files were modified using their default applications. The files were all located in the file browser, and then selected to open, each file was then

modified, saved and closed. The thumbnail cache was examined after the file was closed and no related changes to the thumbnail cache were observed.

The modified files were then viewed using a variety of methods, including through the open command in their default application and through the file browser. The results showed that when a file was viewed in thumbnail view, the visual thumbnails associated with the source file were updated to reflect the current state of the file. During the experiments it was noted that the *thumbcache_1024.db* image, where present was not consistently updated when the subrecord was viewed.

6.5.2.2 Modifying the files URI

If a source file changes its corresponding URI, a new record is generated in the thumbnail cache and no alteration is made to the existing record. This effect was recorded during an experiment described in **Section 6.4.1** where a file was placed at a URI and viewed to ensure a record was added in the thumbnail cache. Once the record was stored in the thumbnail cache, the URI of the file was changed; this involved either altering the file's name or the entire path. The file was then viewed at its new URI and the effect to the thumbnail cache was analysed.

When the file name was changed or the file was moved to a new location on the storage media no effect was noticed in the thumbnail cache. When the file was moved to an external storage device a new record was created in the *thumbcache_idx.db*, with subrecords also being created. Aside from the different thumbcache ID value, the record and subrecords were identical. This suggests that it may be possible to create a false relationship between artefacts in the thumbnail cache and a source file. This could be achieved by an analyst forming a relationship between a user generated file and an entry in the

thumbnail cache based solely upon the visual thumbnail and the file looking similar. The strength of a relationship based upon the image data is discussed in **Section 6.8.2**.

Interestingly, if the file is returned to its original storage media and URI, the existing records are reused if the last modification time is equal to the one currently stored. Whilst experimenting with changing the name of a directory, which had a folder thumbnail, it was noted that this caused regeneration of the folder image. The new visual thumbnail used different images within the folder; however when the directory was changed back to the original name, the old visual thumbnail was displayed. This occurred even when the files relating to the folder image were removed from the directory. It was also noted that having multiple file browsers open with the directories being experimented on in view caused all information to be regenerated at each change, which meant old records were not used.

In *figure 6.8* a directory is being viewed using Windows Explorer, the directory contains a sub-directory named 'Not Viewed'; which is represented by a folder and a selection of thumbnails. Experimentation has shown that the thumbnails used to represent the contents of a directory are selected when the thumbnail is first generated. If the images used in the thumbnail are removed from the directory, the directory thumbnail is not always updated, which may lead to an inaccurate representation of the directory's current contents. Only files which are used for the directories visual thumbnail record have records created for them; therefore there may be records in the thumbnail cache for files which the user has not viewed. Since each user area has its own centralised thumbnail cache, the thumbnails used in the directory visual thumbnail must represent source files which have been present in the directory at some point.

It is also interesting to note that the two images being displayed for the directory are not identical, this is because they are both being taken from different subrecords for that entry in the thumbnail cache and the subrecords were generated at different times and were invoked by different processes which were unaware of other subrecords for the thumbnail cache entry.

6.5.3 The deletion of records and subrecords

There are several options built into Windows 7 to allow the removal of the thumbnail cache; firstly the cache could be deleted by the user. However if the cache is simply deleted, the cache will start again, this may remove traces of a file which is no longer on the system. Since a file leaves multiple traces on a system, experimentation has shown that with the removal of the thumbnail cache, traces of a file can still be found in Windows.edb. The disk clean up wizard provides an option to remove the thumbnail cache, this simply deletes the current thumbnail cache; a new cache would be created next time it was called.

The local group policy editor has an option to “turn off the caching of thumbnails in hidden *thumbs.db* files”, suggesting that Windows 7 can utilise both the directory specific and centralised forms of thumbnail cache. During the experiments conducted in this research, *thumbs.db* files were identified in directories which were accessed over a network or on partitions which were not set to be indexed by the operating system.

Experiments showed that if a file is deleted from the system, it is not automatically removed from the thumbnail cache, instead the record remains; the cache also does not remove records which have not been accessed for some time. However it is possible to alter the default settings within Windows 7, to limit the size of the thumbnail cache, which experiments have shown causes

the cache to remove inactive records when it reaches its maximum size. The cache has been shown to create a sub-directory within the explorer directory named *ThumbCacheToDelete*; this directory is where outdated versions of thumbnail cache files are placed prior to deletion. Moving the thumbnail cache files allows the system to regenerate new files; this is particularly used when the index requires extra space. During this research no method tested has led to the secure deletion of thumbnail cache files, therefore once deleted the files are available in unallocated space.

6.5.4 Fragmentation of the thumbnail cache

The allocation of a predetermined amount of space each time the thumbnail cache files appears to minimise fragmentation, by ensuring a collection of records are stored together. However the individual files tend to be fragmented in clusters the size of the file space allocation unit. Therefore as the files are unlikely to be contiguous, the grouping of several records or subrecords is likely to assist with identification.

6.6 Identifying traces left outside the thumbnail cache

This section identifies the artefacts present on a Windows 7 system outside the thumbnail cache. The thumbnail cache works in conjunction with the rest of the operating system, and also interacts with user files. Therefore it is necessary to identify any correlation between artefacts within the thumbnail cache and those left on the remainder of the system.

The experiments were designed to mimic typical user behaviour to identify information which changed on the remainder on the system that could be related to changes within the thumbnail cache. During the experiments potential thumbnail cache source files were added, modified and deleted both on the

main and external storage devices. The resulting changes to each virtual machine were analysed using a variety of tools, including Encase and WinHex.

Table 6-1: A summary of the user activity which may lead to subrecords being present in the thumbnail cache

		Windows 7
Records in thumbcache_ idx .db		• A URI with a visual thumbnail is viewed in thumbnail view • A URI which represents a shortcut is viewed in thumbnail view
Subrecords in thumbcache_ 32 .db		• Generally system icons or directories viewed in thumbnail view • Visual thumbnails of user generated content selected and viewed in the detail panel of Windows Explorer
Subrecords in thumbcache_ 96 .db		• Directories viewed in thumbnail view • Visual thumbnails of user generated content viewed in thumbnail view
Subrecords in thumbcache_ 256 .db	Standard subrecord	• Visual thumbnails of user generated content viewed in thumbnail view from indexed drives • Directories viewed in thumbnail view • Visual thumbnails of images used in directory visual thumbnails
	Partial subrecord	• GUIDs relating to system icons viewed in thumbnail view • ID values relating to icons for shortcuts viewed in thumbnail view • When viewed in Windows Explorer navigation or main panels: • Letters relating to allocated drives • Network place names
Subrecords in thumbcache_ 1024 .db		• Visual thumbnails of user generated content viewed in extra-large thumbnail view from indexed drives • Visual thumbnails of user generated content selected and viewed in the preview panel of Windows Explorer
Subrecords in thumbcache_ sr .db		No information added during experimentation
<i>thumbs.db</i>		Created on non-indexed drives when user files are viewed in thumbnail view

6.6.1 Windows.edb

The database for Windows Desktop Search stores a wide range of information including comprehensive metadata about user generated files [Chivers, 2011]; therefore the database can potentially be used to provide extra and corroborating information on a user file which may be of interest to an analyst. Interestingly, the Windows Desktop Search database contains a field which holds the thumbnail cache ID [Douglas, 2009]; this field can be used to form a relationship between a file indexed in the database and a record in the thumbnail cache. Other metadata stored in Windows.edb may also assist in strengthening the relationship between a source file and the thumbnail cache for example Exif data. However, there may not be a record in the database relating to the file or thumbnail cache ID the analyst is looking for; Chivers proposes several approaches to recovering deleted records in the database, which may recover the record the analyst requires [2011].

6.6.2 Registry

There are records stored in the Windows 7 thumbnail cache that do not contain visual thumbnails [**Section 6.4.4**; Morris, 2011]; some of these records contain a GUID. The GUID can represent system icons found on the user's desktop; the relationship can be identified by searching for the GUID within the registry. *Figure 6.9* shows the results of searching for a GUID which relates to the Recycle Bin; by selecting the default icon directory shown in *Figure 6.10* it is possible to identify the location of the icons being used. In this example the icons are stored in *imageres.dll* in positions 54 and 55 depending on the state of the recycle bin; an example contents from an *imageres.dll* file is shown in *Figure 6.11*.

In order to identify the icon used on the desktop it is necessary to identify the contents of file containing the icon as the result may be different to the standard icon, for example the user may be using a customised theme. Restore points

can identify when the icons appeared on the desktop; this is discussed further in **Section 6.8**. Generally system icons are added when a user first logs into their user area, however it is possible to manually add system icons later. This was established by examining the baseline machine described in **Section 6.3**, where an account had been created and logged on to in order to complete the creation of the account. On examination of the thumbnail cache GUID records were identified which were searched for in the registry hives. The GUIDs were found to correspond with “*My Computer*” and the “*Recycle Bin*”; the two system icons present by default on the user’s desktop.

Further experiments were performed on the machine by adding 100 JPEG files a day over a 10 day period from a USB device to the user’s documents directory. After the first day the system icon for the “*Control Panel*” was added to the desktop; this resulted in the corresponding GUID being added to the thumbnail cache after the 100 visual thumbnails. Interestingly there was no associated record in the thumbnail cache index file for the subrecord containing the *Control Panel* GUID. After the experiment the virtual machine was analysed and the *Control Panel* GUID was found to have moved above the records for the images added in Day 1. The subrecords containing GUID’s generally move to the top of the thumbnail cache file and do not always contain a reference in the index file. System icons may also be removed by a user; however the reference to the icon appears to remain in the thumbnail cache which may assist in identifying the behaviour of the user.

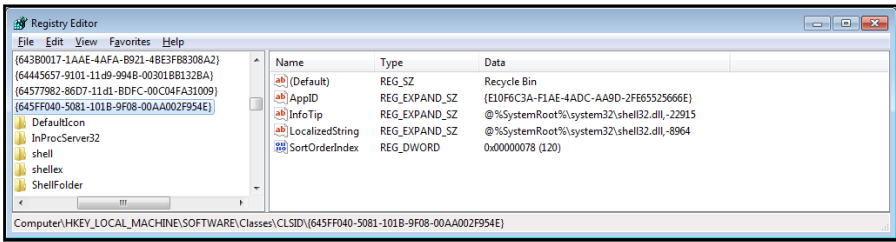


Figure 6-9:Searching for a GUID

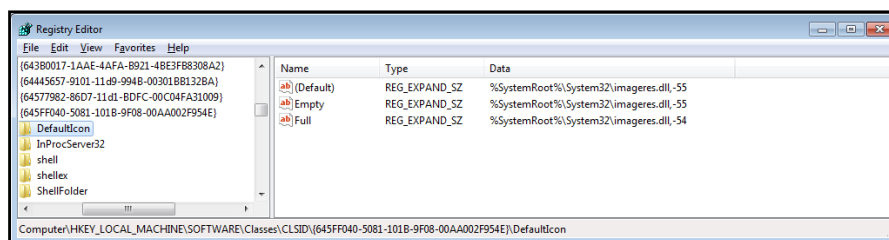


Figure 6-10: Identifying the Default Icon

6.6.3 Shortcuts

Shortcuts on the system can also have visual thumbnails; these can be system, application or user created icons. *Figure 6.12* shows an example of a shortcut icon for the directory 'Forensic Software'; the icon is not stored in the thumbnail cache, however a reference to the icon is. This was identified by using the virtual machine Clone 1-1 as described in **Section 6.4**; 20 shortcuts were created on the system, including the one for the 'Forensic Software' directory. The shortcuts when viewed in Windows Explorer were associated with an icon; when the thumbnail cache was examined new entries were found in the thumbcache_256 file. The new entries contained a hexadecimal character string and no visual thumbnail. The results show that the reference is a subrecord without an image, the name field stores a 30-32 hexadecimal character string in Unicode. For software shortcuts, there is generally a registry key which stores the default icon to be used, which like the desktop icons could be extracted by the analyst.

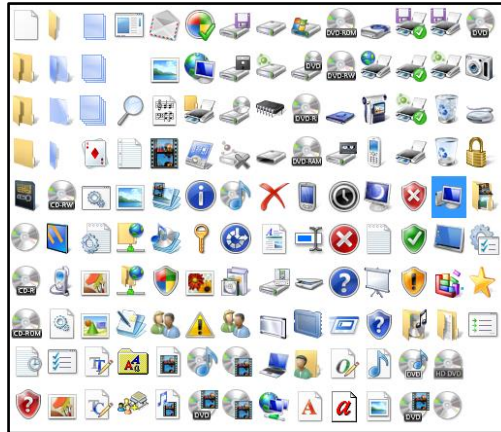


Figure 6-11: An example contents from an imageres.dll

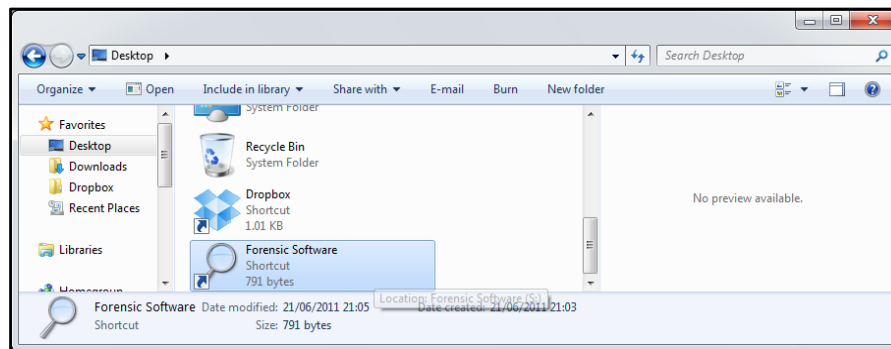


Figure 6-12: A Shortcut Icon in Windows Explorer

6.6.4 External Media

Visual thumbnails may be stored in the centralised cache for any potential source file which is stored on an indexed volume; during the experiments documented in this chapter the external media were automatically indexed when connected to the system. An experiment was performed using the baseline virtual machine where 10 external USB devices were each attached. Each USB device contained 25 of the files chosen during the experiments performed in **Section 6.4**. First the Computer icon was selected in the navigation pane of Windows Explorer to expand the list of currently available drives; on examination each of the 10 USB device drive allocations were found in the thumbnail cache. The contents of each drive were then viewed in

thumbnail view for 30 seconds. Upon examination records in the thumbnail cache were present for each of the files on the external media even though the files had not been present on the main storage media.

Part of the algorithm for the generation of the thumbnail cache ID uses the volume ID; the generation of the thumbnail cache ID is discussed further in **Section 6.8.3**. Therefore by generating an ID for a file it may be possible to show the volume the file resided on. However, it would be time consuming to attempt to reverse engineer the ID given the number of variables involved in its creation.

6.6.5 Encrypted Containers

Encrypted containers such as those used in TrueCrypt [2013] can be mounted as a volume by a user to enable the content to be accessed. Once the content is mounted it is possible to index the container the same as a normal volume. In order to ascertain whether any information from the TrueCrypt drives was stored in the thumbnail cache 3 clones of the baseline virtual machine were created.

- Experiment 1: A 10GB NTFS TrueCrypt volume was created on the user's desktop. The file structure and associated files documented in **Section 6.4** were copied into the volume. The contents of the drive were viewed in Windows Explorer thumbnail view, a file in each sub-directory was selected using a single click.
- Experiment 2: A 2GB FAT TrueCrypt volume was created on the user's desktop. The file structure and associated files documented in **Section 6.4** were copied into the volume. The contents of the drive were viewed in Windows Explorer thumbnail view, a file in each sub-directory was selected using a single click.

- Experiment 3: Experiments 1 and 2 were repeated, but with the option to index the mounted drives selected.

In experiment 3, as expected both drives created records in the centralised thumbnail cache for each of the files viewed. In experiments 1 and 2, a *thumbs.db* file was created in each directory viewed on the drive. The drive letter assigned to the volume had an associated record in the centralised thumbnail cache; no other identifiable information about the volume was stored in the central thumbnail cache. In experiment 2 it was found that the selection of records within a FAT32 TrueCrypt volume by a user results in the creation of corresponding records within the thumbnail cache.

6.7 Alteration of thumbnail cache artefacts

This section investigates the strength of the thumbnail cache with respect to tampering; three different pieces of data relating to the thumbnail cache are manipulated during these experiments to attempt to mislead a user or analyst.

6.7.1 Metadata

It is difficult to alter the metadata of a thumbnail cache record in order to mislead the user; the thumbnail cache is closely linked to data within the Windows.edb file as well as the original source file. An experiment was performed where the data in a record was switched with another existing record. This meant although the records maintained their position in the index, they pointed to the wrong locations in the subrecord files. Such a modification also required the thumbnail cache ID in the subrecords to be altered, as well as the checksum for the subrecord header. Despite changing all known relevant metadata, the experiments failed to get an incorrect subrecord image to be displayed, with each attempt resulting in the correct subrecord being recreated and appended to the end of the appropriate file. This shows comprehensive

checks are performed on the associated metadata during the thumbnail image generation algorithms.

6.7.2 Visual thumbnail

When a user is viewing a file in thumbnail view, the corresponding visual thumbnail is requested from the cache; if it exists the visual thumbnail is displayed, otherwise one is created. This was observed during the experiments described in **Section 6.4.1** where files were added to the system. Therefore the ability to modify the visual thumbnail may mislead the user by misrepresenting the file's contents. Several experiments were performed to attempt to get the system to display an inaccurate visual thumbnail; these are described below.

First just the image section of a thumbnail subrecord was modified; the image was replaced with another pre-existing visual thumbnail. It was determined that the checksum for the image would also require modification to match the updated information; once the subrecord had been modified the relating source file was viewed in the file browser. The result proved interesting as initially the modified visual thumbnail was displayed; however this was replaced by the actual visual thumbnail for the file within a few seconds.

A second test showed that if the directory was only viewed quickly, for example a user upon entering the directory selects a sub-directory to view, that the subrecord entry remains in its altered state. This suggests that it may be possible to mislead a user if they were browsing through directories without giving the thumbnail cache an opportunity to authenticate the results. It also suggests that a tampered record which has not been viewed would remain in the tampered state, and if appropriate metadata such as the checksum was altered it would appear to be a valid entry. However whilst it is possible to tamper with the visual thumbnail, it would be infeasible to edit all additional artefacts

which have a relationship with the thumbnail cache record; therefore if tampering had occurred an analyst would identify conflicting artefacts.

6.7.3 Original source file

In **Section 6.5** it was shown that the modification of the contents of a user file within the virtual machine would result in the corresponding record in the thumbnail cache being updated the next time it was viewed in thumbnail view. A further experiment was performed where the last modified date of 6 files were altered; the times were moved backwards by 24 hours so that the modification time of each file appeared a day earlier. The files were then viewed in Windows Explorer thumbnail view; there was a noticeable delay in displaying the visual thumbnails whilst they were regenerated. If the last modification date in the original source file is altered to a time before that stored in Windows.edb, then the thumbnail subrecord is regenerated next time it is called.

It is possible to modify the contents of a file within a hexadecimal editor, therefore ensuring the files metadata remains accurate. This was identified by editing a JPEG image using WinHex version 15 running from a portable storage device attached to the virtual machine; the image was replaced by one of the same size but with noticeable visual differences. The results showed that this form of modification does not lead to the records within the thumbnail cache being updated; as the checks performed on the data show the subrecord is accurate.

6.8 Forming a relationship between the thumbnail subrecords and the source files

The previous section discussed ways in which the information in the thumbnail cache could potentially be manipulated; the section highlighted the importance of relationships to provide corroborating information and make unidentifiable tampering of data harder. This section discusses the artefacts that can assist in

showing a relationship between information contained with the thumbnail cache and a thumbnail cache source file. This is achieved by evaluating the metadata and images both within the cache and source file; finally the use of event timeline reconstruction in forming a relationship is considered.

Establishing a relationship between a file created by a user and the thumbnail cache can assist in corroborating the information identified; it can also assist in forming an overview of the user's interaction with a file. This section describes the ways in which a relationship can be formed between a thumbnail cache record and a user generated file.

6.8.1 Visual Thumbnail

A visual thumbnail provides a snapshot of at least part of a file at a point in the file's history; if the visual thumbnail relates to the current state of the file then a visual comparison could be performed to identify if there is any similarity in its content to a user file. If the visual thumbnail and a user file appear to contain the same data then there may be a relationship between the two sets of information; an analyst cannot be certain the two sets of data are related. For example, there may be multiple copies of a file on the user's system, in which case all the copies may visually match with the same subrecord however only one copy would have a relationship with it.

The visual thumbnail may be compared to part of a user file using computer vision techniques; if the two images are the same size then a one-to-one mapping could be applied [Dufournaud, 2000]. However given it is likely that the two images are different sizes and may also be in different image formats the problem involves a one-to-many mapping; the increased requirement for the calculation of mappings may mean the technique is too resource intensive for large data sets and would therefore require the data set to be pre-processed.

Like the visual inspection by an analyst a matching algorithm can only narrow down the potentially related subrecords.

The visual thumbnail may not represent the last state of the file, in Windows 7 each record in the thumbnail cache may contain up to four different sized visual thumbnails [Morris, 2011]; it is possible that each of the visual thumbnails may represent different states of the file. Therefore the state represented by the visual thumbnail may not match the last state of a file. This would prevent a visual match of the data. It would therefore be necessary to examine each visual thumbnail for a record to maximise the chance of identifying a visual match between the data in the record and the file; it may also assist in identifying previous states of the file which may provide useful information for an analyst.

6.8.2 Metadata

The removal of time stamps from the thumbnail cache in Windows 7 makes it difficult for an analyst to establish the time a record was last updated within the thumbnail cache [Morris, 2011]. However it is possible to identify a time period during which a record was added to the thumbnail cache; previous versions of the thumbnail cache files are stored in Windows 7 by the Volume Shadow Service. *Figure 6.13* shows the windows restore points for an Explorer directory. Since each previous version has a date attached it is possible to work out a time frame during which a record was added. Selecting each previous version of the directory stored by the Volume Shadow Service will show the thumbnail cache files in an earlier state; by comparing the information within the versions of the thumbnail cache it is possible to identify a basic order in which records were added.

The type of a file can be identified through the flags stored in the thumbnail cache; each type of file has a specific sub-set of flags; these flags are found in the main record for each file located in the *thumbcache_idx.db* file [Morris, 2011]. The flags can be used to identify the type of file which relates to the record, as the file type may not be obvious from a visual inspection [Section 6.4.2].

6.8.3 Thumbnail cache ID

Each record contains a unique thumbnail cache ID which is made up of sixteen hexadecimal characters. In order to ascertain the information which was used in the formula for creating the thumbcache ID a series of tests were conducted. The tests were conducted to investigate the possibility of a simple relationship that allowed a file path of MFT ID to be determined from the thumbnail cache ID. Each test was performed in a clone of the baseline virtual machine with a set of 20 user created files stored in the current user's *"My Documents"* directory; the images were viewed using Windows Explorer in order to create a baseline thumbnail cache. The files selected were: 2 BMP, 2 DOC, 2 DOCX, 2 GIF, 5 JPEG, 3 PDF, 2 PPT, and 2 XLS.

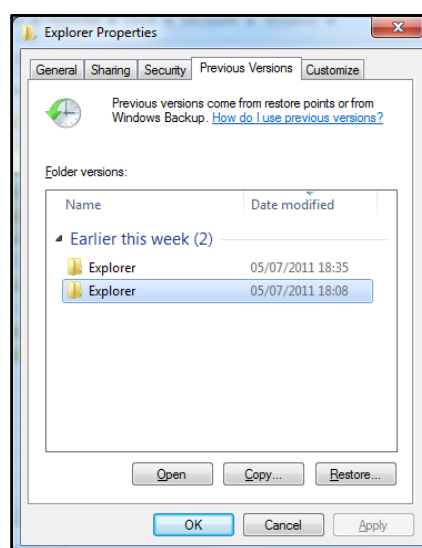


Figure 6-13: Restore points for the directory Explorer

In each experiment a single variable was altered on all of the files; for example variables included: changing the file path, changing the volume the file was stored on, changing dates/times of both the system and the files. After each test the resulting thumbnail cache was extracted and compared with the original thumbnail cache; therefore any changes made to the thumbnail cache ID values could be identified. The results of this testing have shown this ID is comprised of data relating to the user file; it has been identified that the thumbnail cache ID is related to the volume, MFT data and the type of the file.

This suggests that there is no straightforward way of recovering the file path from the thumbnail cache ID; however, in **Section 6.6.1** the thumbnail cache ID is used to form a relationship through the Windows Desktop Search database.

6.8.4 Event timeline

It is possible to identify the order user created files were added to the thumbnail cache by identifying their position in the individual thumbnail cache files. **Section 6.6.4** identified that it is possible for subrecords which do not contain visual thumbnails to move closer to the top of thumbnail cache files over time. **Section 6.5.1** described an experiment which suggested that new subrecords containing visual thumbnails were appended to the bottom of the thumbnail cache files; they also maintained their relative positions in the thumbnail cache overtime. Therefore for user generated files the related subrecords in the file appear in the order they were created by the system. This can assist with creating an event timeline as a subrecord can only be created for an original source file which has a relationship with the operating system at the time of creation. A relationship between the original source file and the operating system is defined as the file being viewable within the operating system. In order to add extra data to the event timeline it is possible to add metadata recovered from the subrecords, windows.edb, and the user created source file.

The event timeline created could then be placed in context with other system information to provide an overview of the systems activity.

6.9 Audio and Media thumbnails

As discussed in **Chapter 2**, there are three kinds of thumbnail used in modern computing: audio, media and visual. This chapter has so far focused on the visual thumbnails; this section describes the application of audio and media thumbnails within Windows 7.

6.9.1 Audio thumbnails

Audio thumbnails, as defined in **Chapter 2** are representations of audio files, which allow a user to preview the contents. In Windows 7, an audio file can be identified by the visual thumbnail used for the file; this can be either the generic audio visual thumbnail or album art with an icon indicating the software associated with the file. *Figure 6.14* shows the two different types of audio visual thumbnail and describes their individual characteristics.

The audio thumbnails are activated by selecting an audio file in the file browser; the file then appears in the preview panel with an option to play the audio track. If the file is deselected, the file will be removed from the preview panel and will cease to play. *Figure 6.15* shows the file browser with an audio thumbnail selected; the audio file can only be played from the beginning within the preview panel. An experiment was performed where 10 audio files were copied into the current user's *My Documents* directory in a virtual machine. The audio thumbnail for each of these 10 thumbnails was then accessed; the first audio thumbnail was accessed for 10 seconds, with each subsequent audio thumbnail adding 10 seconds to the amount of time they were played for. The results of

this experiment have shown that the audio thumbnails do not appear to be stored on the system; however the visual thumbnails associated with an audio file are stored within the central thumbnail cache.

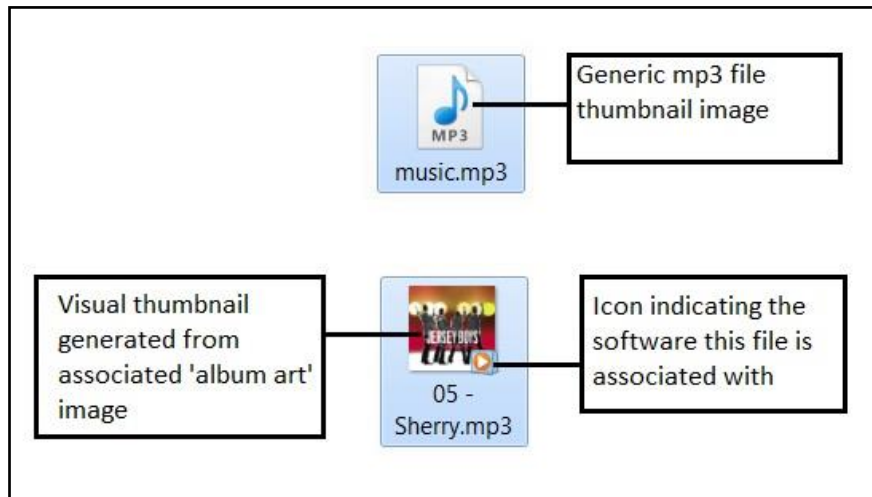


Figure 6-14: Audio Thumbnails

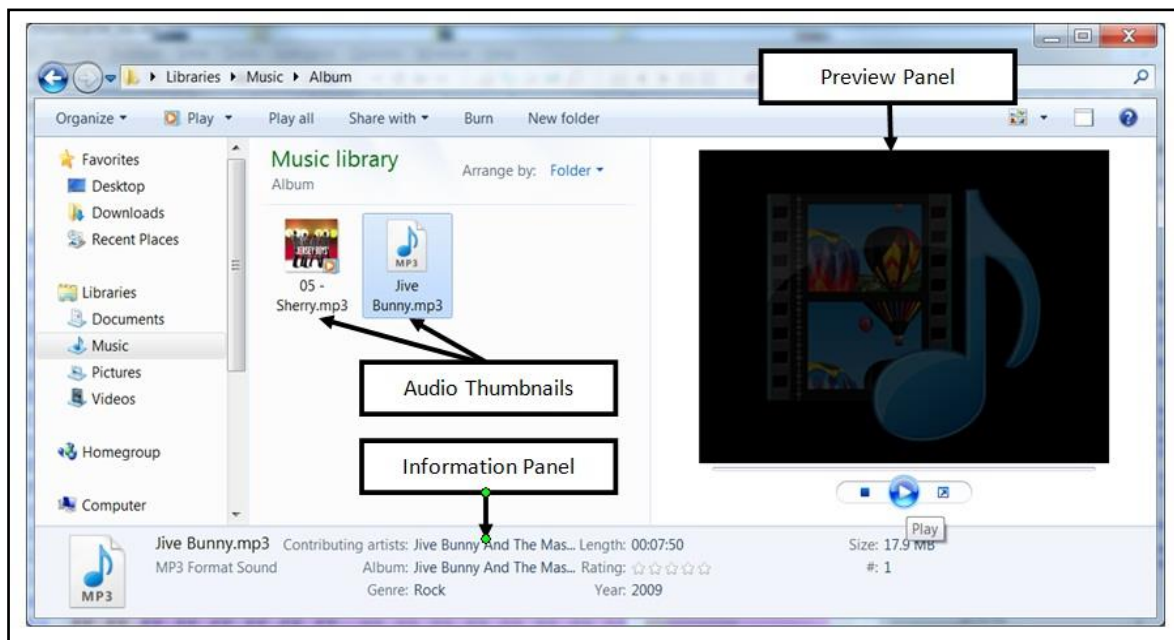


Figure 6-15: The use of audio thumbnails

6.9.2 Media thumbnails

In Windows 7, multimedia (media) thumbnails have a common structure, which can be seen in *figure 6.16*. The film strip border on the visual thumbnail identifies the file as having a media thumbnail. The media thumbnail preview of the file can be accessed when the file is selected in the file browser (*figure 6.17*). As with audio thumbnails, upon selection of the file, the media thumbnail can be seen in the preview pane. If the file is deselected, the media thumbnail will cease to play; should the file be reselected, the media thumbnail can only be replayed from the beginning.

An experiment was performed using a set of 10 media files copied into the current “*My Documents*” directory of a virtual machine. Each of the media thumbnails was accessed; the first media thumbnail was accessed for 10 seconds, with each subsequent media thumbnail adding a further 10 seconds to the access time. The results have shown that the media thumbnails are created at runtime and are therefore not store on the system. However as with audio thumbnails, the visual thumbnail is stored in the cache.

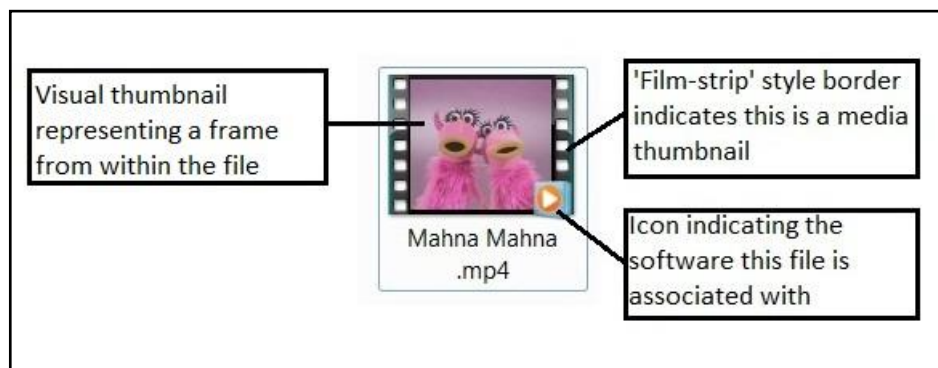


Figure 6-16: The structure of a media thumbnail

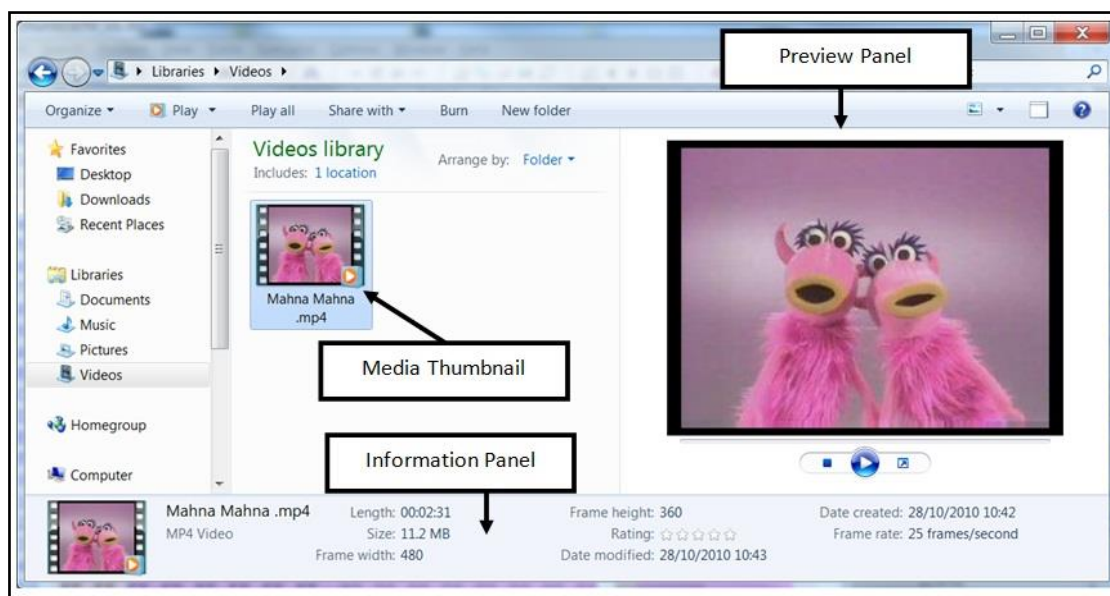


Figure 6-17: The use of media thumbnails

6.10 Discussion

In summary, *table 6.1* provides an overview of the main user activity which leads to artefacts being generated within the thumbnail cache, based on a default installation. The table shows the actions which lead to the creation, modification, and removal of records and subrecords from within the thumbnail cache structure. The table shows both the behaviour for the standard records which are associated with visual thumbnails and the non-standard entries. As can be seen from the table a wide variety of user activity can lead to information being added or updated within the thumbnail cache; however the only actions identified during this research which led to the removal of information were the user choosing to delete the thumbnail cache either manually or through a system cleaning menu.

This section discusses the potential evidential value of Windows 7 thumbnail cache artefacts. The discussion is based around the evidential strengths identified in **Chapter 4**; this begins with looking at the criteria for establishing

the provenance of artefacts. This is followed by the criteria for assessing the interpretation of results.

6.10.1 Provenance of Artefacts

This chapter has identified the structure and represented behaviour of thumbnail cache artefacts identified in Windows 7. An analyst could use the experimental methods described throughout this chapter (for example **Section 6.2**) to **replicate** the experiments and verify the artefacts presented. The ability to replicate the results based on the descriptions provided in this chapter show the use of **clear documentation**; an analyst would be familiar with this style of documentation.

The thumbnail cache structure is identified in **Section 6.4**; this led to experiments to determine the types of behaviour which can lead to the addition, modification, or deletion of data from within the thumbnail cache [**Section 6.5**]. This information was corroborated by forming relationships with other artefacts in the system and with the user generated source as described in **Sections 1.6 and 1.8**. The thumbnail cache artefacts identified during this research are **justifiable** as they are identified using a standard replicable process that enables the verification of the artefacts presented.

The features identified in the thumbnail cache motivate the requirement to identify and provide **corroboration**. For example, an analyst may be investigating claims that the suspect was unaware of terrorist material on their computer and that it was automatically downloaded. The thumbnail cache may contain several different sized visual thumbnails, metadata about the original file, and in conjunction with other system information it is possible to corroborate the path of the original file. If a record is present for any of the material in the thumbnail cache the analyst can determine potential user activity which may

have occurred; it could also determine which user activity could not have resulted in the current state of the thumbnail cache record [**Section 6.5**].

In order for an analyst to further reduce the list of potential user behaviour which led to the thumbnail cache's current state it is necessary to form corroborative relationships with other system information such as those described in **Section 6.6**. Therefore an analyst needs to identify corroborating artefacts to put the information from the thumbnail cache into context; corroborating artefacts relating to the user generated source file can be identified on the system.

A relationship must be maintained between the results and the original data in order to show thumbnail cache artefacts identified are ***understandable consequences*** of the methods used. For example in **Section 6.6.1** a series of experiments are described which aim to form a relationship between a user file and the thumbnail cache; focused experiments such as the experiments relating to each main type of data stored within the thumbnail cache assist in maintaining a relationship between the original data and the artefact. The artefacts identified as a consequence of the user behaviour were documented throughout this chapter and could be applied to artefacts identified during forensic investigations.

In order to present an artefact in Court it is necessary to show that the artefact was identified and put into context using a ***scientific method***. The structure and syntax of any thumbnail cache artefacts identified by an analyst can be checked to ensure they are valid; the structures are documented in **Section 6.4**. Once verified the artefacts can then be placed in context by identifying relationships with other system artefacts. Putting the artefacts in context can assist an analyst in showing or disproving that a type of associated behaviour took place.

6.10.2 Interpretation of Results

An analyst may be presented with a scenario where doubt is cast upon whether the system was ***working properly*** at the time of the alleged incident. The manipulation of thumbnail cache artefacts was evaluated in **Section 6.7**. The section showed methods of verifying and corroborating the artefacts in the thumbnail cache to ascertain the likelihood they were created whilst the system was working normally. It is a non-trivial task to alter all the relationships within the system which relate to the thumbnail cache; therefore it is likely an analyst would find some information to cast doubt on the manipulated data. The experiments performed in **Sections 6.4 - 6.6** were based on a system functioning normally and document the expected results of certain behaviours. By establishing relationships between the thumbnail cache and the rest of the system it is possible to analyse the information in the thumbnail cache in the context of the general system activity; therefore the ***behaviour*** an artefact represents can be established [**Section 6.5**].

Thumbnail cache artefacts can be corroborated by forming ***relationships*** with other system artefacts [**Section 6.6**]. Artefacts such as a GUID within a non-standard thumbnail cache entry can be difficult for an analyst to place into context without supplementary information. However if a relationship is formed between the thumbnail cache entry and the registry it is possible to determine the meaning of the GUID and create further relationships to provide additional information. Relationships between thumbnail cache artefacts and the original source file can also assist an analyst; the potential system behaviour identified from thumbnail cache and corroborating artefacts can be linked to a specific file.

By forming relationships with other sources of information and continuing to build relationships until all the links between artefacts are identified it is possible to ***maximise the corroboration*** of thumbnail cache artefacts. The thumbnail

cache artefacts can be corroborated by forming relationships with other system artefacts and the original source file [Sections 6.6, 6.8]; this shows that multiple sources of corroboration for thumbnail cache evidence can be identified and in turn those sources can be corroborated.

In Windows 7 there is a centralised thumbnail cache for each user which contains six binary files; therefore in order to *maximise the data* recovered from the live thumbnail cache it is necessary to extract the six binary files and examine them. Windows 7 also uses the *thumbs.db* format, which are directory specific thumbnail cache files. In a live file system these can be identified by searching for the filename *thumbs.db*; they can then be extracted and analysed. The carving of data relating to the Windows 7 centralised thumbnail cache from unallocated space is discussed in **Chapters 7-9**.

The thumbnail cache structure and syntax identified in **Section 6.4** permit the *automation* of extracting data from the file formats; they also permit the identification and reassembly of file fragments from unallocated space.

6.11 Conclusion

This chapter has examined the structure and behaviour of the operating system thumbnail cache implemented within Windows 7. Within Windows Vista, the thumbnail cache implementation changed significantly from Windows XP by using a centralised thumbnail cache for each user, this implementation was continued in Windows 7 with slight differences in the structure of records and subrecords. The thumbnail cache contains visual thumbnail records alongside GUID information, active drive letters and network places. Records are added to the cache, with a new subrecord being added to the bottom of the thumbcache files. Some records are deleted periodically, however this is application specific. The subrecords contain checksums to assist in authenticating the data;

however it is still possible to tamper with parts of the thumbnail cache without it detecting an error. Finally audio and visual thumbnails are generated at run-time and experiments have shown aside from their visual thumbnail they leave no trace on the system.

7 Identification of thumbnail cache fragments

7.1 Introduction

In **Chapters 5 and 6** the structure and behaviour of the operating system thumbnail caches found in Ubuntu, Kubuntu and Windows 7 were identified; the results of this research provide a baseline which shows the artefacts recovered from complete thumbnail caches and related system artefacts. This chapter expands upon the previous research to explore the identification of thumbnail cache file fragments from unallocated space. It was established in **Chapter 1** that the increase in user awareness of both forensic practices and technology would make it easier for a user to attempt to hide their activity. If a user attempts to hide their activity it is likely the thumbnail cache would be deleted and therefore may exist either in whole or in part in unallocated space at the time of analysis. Also, as established in **Chapter 6** when files in the Windows 7 centralised thumbnail cache are updated, the old file is given a temporary name and deleted; therefore traces of these temporary files may also be found in the Master File Table (MFT) and unallocated space.

The growing size of storage media and the possibility that only fragments remain of a piece of important evidence suggest that analysts may need to investigate the contents of unallocated space. It is too resource intensive for an analyst to investigate unallocated space in depth for every case. However the ability to identify fragments and reassemble them using an automated method may provide an analyst with further information about the state of the system with minimal resource cost. The likelihood of potentially relevant thumbnail cache artefacts being in unallocated space suggests research into identifying thumbnail cache file fragments would be relevant and may assist in automating a potentially lengthy process. The process developed will also show that the approach is generalisable as it will have to be adaptable to different

implementations of the thumbnail cache; therefore the approach used in this research could enable the identification of other file types.

This chapter investigates methods for identifying thumbnail cache file fragments. **Section 7.2** discusses traditional data carving techniques; recent data carving research is discussed in **Section 7.3**. In **Section 7.4** a methodology for the research in this chapter is established; the information to be carved is described in **Section 7.5**. This is followed by the four most common types of identification method being adapted and implemented for thumbnail cache file fragment identification [**Sections 7.6 – 7.9**]. **Section 7.10** describes the results of the implemented file fragment identification methods; a comparative discussion of the four techniques follows in **Section 7.11**. A discussion of this research is provided in **Section 7.12**; finally this chapter is concluded in **Section 7.13**.

7.2 Traditional Data Carving

Both NTFS and FAT store information showing a relationship between individual clusters and their contents. The references in the file system provide a starting point to identify the clusters used to store a file. Fellows [2005] discusses the recoverability of deleted files when the directory structures such as the MFT are available; however the paper has little discussion on the recoverability of potential evidence from such files when the MFT information relating to a cluster is no longer available. There may be information on a device which does not have any file system references but is pertinent to the investigation; the file system references may not always be available, for example if the device has been formatted.

Without the file system references, recovery becomes more complex; this creates a requirement for file carving. File carving is the technique of extracting

a file from a storage device based upon its structure. It is commonly used when the directory information is unavailable, to extract as much of the file as possible. In the case of deleted files this may not be the complete file as it may have been partially overwritten [Sammes, 2007]. The following sub-sections discuss traditional data carving approaches; in **Section 7.2.1** the brute force approach to data carving is described. This is followed by a description of structural and syntactical data carving approaches in **Section 7.2.2**.

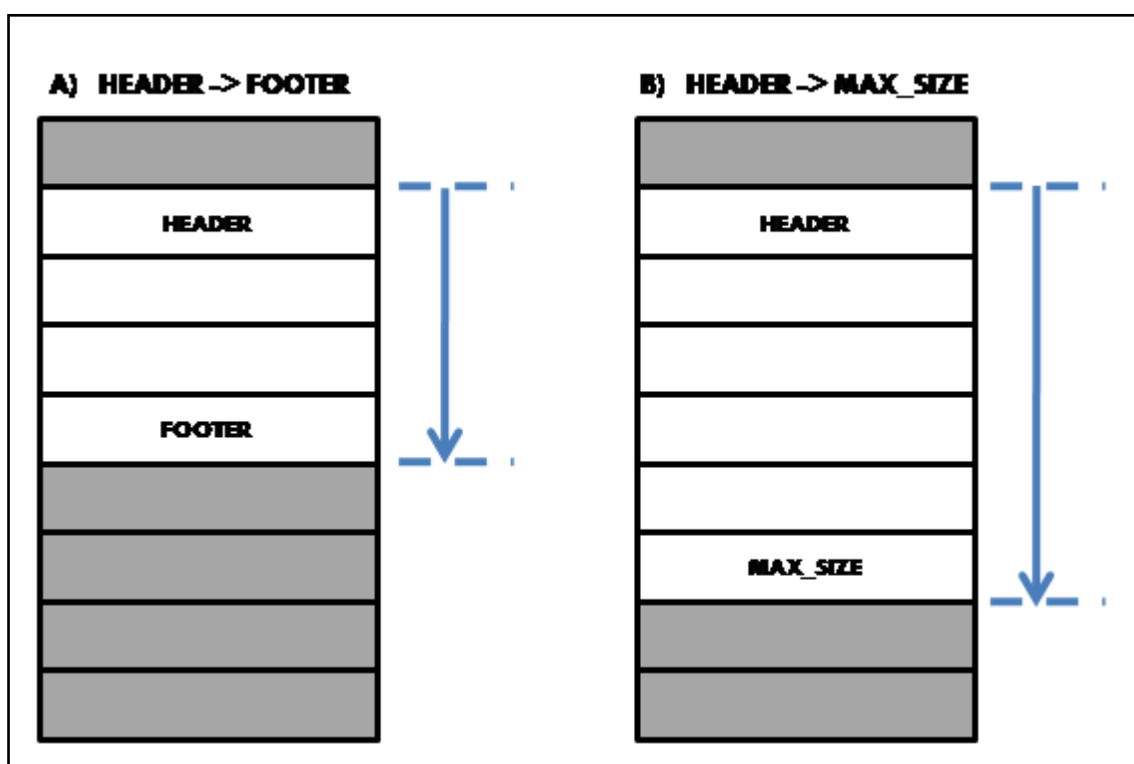


Figure 7-1: Traditional file carving from clusters of unallocated space on the storage media

7.2.1 Brute Force Approach

File carving, or file recovery is the process of retrieving complete or partial files from the storage media when there is no file system information available. Early file carving methods were based on identifying the header and footer of a file and then carving out all the information between these two points; this provides

valid results only when the file is: contiguous, not fragmented, and has a footer [Calhoun, 2008]. As can be seen in *figure 7.1* once a file signature is identified the file is carved until a footer is identified. If the file does not have a footer then it is possible to carve the file by extracting a predetermined number of bytes which is larger than the expected file size. The maximum size is determined by the analyst based upon the file type. Forensic tools such as Winhex provide an expected file size for file types which do not contain a footer; for example for thumbs.db or registry hive files.

7.2.2 Structural and Syntactical Analysis

The *brute force* approach, described above, relies on simple sequences to identify information; by using additional information about the file type it is possible to improve the accuracy of the identification. In *Figure 7.1* the approach relies upon a footer signature. This type of file carving may not identify the whole file. For example PDF files have an ambiguous footer signature which occurs in multiple positions within the file. In order to carve out a complete PDF file an offset larger than the potential size of the PDF file is selected; from here the offsets back towards the PDF header are checked for the footer signature. Identifying the file signature and carving the data that follows it will only retrieve a complete file if the file is contiguous; therefore fragmented files will only be partially recovered which may result in potentially relevant artefacts not being recovered [Pal, 2009].

Another method for identifying artefacts is to search for keywords, for example using a hex editor. It is also possible to search for a structure using regular expressions [Fleishmann, 2008]. Whilst the file signatures would be found in the first cluster a file occupies, later clusters would not contain this information; if the cluster containing the header was overwritten it may be difficult to identify the remaining clusters.

If potential evidence is identified in unallocated space using a keyword search it is possible to identify the potential file type by applying structures of known file types to the information and validating the syntax. This approach uses the *Brute force* approach to identify potential candidate information in unallocated space. In order to extract information from the fragment a template containing the structure can be applied to the information; the template extracts and interprets the information [Fleishmann, 2008]. The identified keyword is used to ascertain the type and position of the structure. The information can then be extracted from the file using the context and checked to ensure it follows the expected valid syntax.

7.3 Data Carving Research

To encourage the development of file carving techniques within the forensic community, the organisation *Digital forensic research workshop* [DFRWS] ran challenges [Dickerman, 2006], which created a surge of interest and new techniques for file recovery, several of which are discussed later in this chapter. After the focus of the challenge moved the research continued; however tools still struggled to deal with fragmented or incomplete files.

Cohen [2007] suggests that identifying positive and negative constraints on files, to establish whether a fragment belongs to the correct file type, can improve efficiency and give a greater level of accuracy. Most algorithms implicitly use this approach to rule out fragments that are not of the correct type. His descriptions of carving PDFs, Zip files, Mpegs and email formats highlight the requirement to understand the structure of the file being carved; this ensures the resulting file can be validated

7.3.1 Statistical Methods

If document fragments can be identified as belonging to the same file it is possible to reassemble them using statistical modelling based upon the expected context of the file [Shanmugasundaram, 2003]. The fragments are added to a graph with the probability of adjacency to other fragments being calculated and used as weights; the reassembled document is therefore equivalent to picking the sequence of fragments with the highest probability of being a structurally and syntactically valid file. Currently this is possible only for files where all the fragments are available, making it useful in cases where the directory structure is missing, but would not provide assistance reassembling incomplete files. This algorithm can currently only find a solution when it is given an input that consists solely of fragments from the file to be reassembled [Memon, 2006]; it is unclear how fragments could be sorted into individual files during an investigation to provide the necessary input. This research was conducted by looking at a variety of file types including logs and binary files. This implied the authors wanted to determine the effects of carving multiple file types that may be relevant to an investigation rather than just focusing on a specific structure such as OLE2 or jpeg.

Fragmentation of a file occurs when it is not possible to store the file in contiguous sectors, such as when storing a large file. A fragment may be part of a fragmented file or as little as a single cluster recovered from unallocated space. Research has suggested that file fragmentation is not present on all drives that an analyst would come across [Garfinkel, 2007]; this is in line with the fact that different users have different needs from their computers and are therefore likely to store varying amounts of information.

Garfinkel [2007] collected a series of hard drives from places such as eBay and whilst examining them found that from the 51 NTFS file systems he examined approximately 12.2% of files were in at least 2 fragments, with 6.6% being in 5

or more fragments, showing that fragmentation is likely to be encountered on NTFS systems, this is in contrast to FAT systems where almost 97% of files were found to be contiguous. Approximately 44% of the total drives in the corpus contained no fragmented files, however the total number of files found from these drives accounted for less than 1% of the files recovered from the entire corpus, suggesting the more files found on the drive the greater the chance of fragmentation occurring.

Garfinkel [2007] makes an interesting statement when he discusses the probability of encountering fragmentation is related to the individual type of the file; user generated files such as word documents were found to have higher fragmentation rates, possibly due to the tendency to work on such files over a prolonged period. These results show that whilst tools such as Disk Defragmenter exist it is still likely an analyst will find fragmented user files on the disk. This suggests that the file types an analyst is likely to be interested in are those most at risk of fragmentation; this suggests that fragmented file carving may be necessary to recover relevant artefacts. Since this study was conducted, the use of computers has altered; users are now storing vast quantities of media such as music and videos on their machines. The media can take up a significant amount of space; whilst the size of storage media is also increasing, the large volume of data being stored still suggests a greater chance of encountering fragmented files. Since the thumbnail cache stores information relating to user files, it is likely to become fragmented if enough user files are added that generate thumbnail cache content. This is particularly likely if the additions are done over a prolonged period as the additional clusters required to store the new subrecords may not be available contiguously on the storage media.

Fragments can be classified based on pattern recognition statistics; a prototypic histogram is used in a supervised learning environment to attempt to model file types, and research has shown this to be reasonably successful with a low error

rate for HTML and JPEG files [Veenman, 2007]. Veenman also establishes the need to combine such techniques with pre-existing carving techniques in order to maximise the information being carved, which may take a greater length of time to run on a given storage device. To date, such techniques have been relatively separate in their research and implementation, and whether developing a tool encompassing several methods would decrease the error rate on more complex file types is still unknown.

Calhoun [2008] looked at predicting the file types of fragments using linear discriminant and longest common substrings. It was found that using longer byte sequences for statistical analysis would improve the prediction rate of the linear discriminant method; this may be due to the fact that information is normally stored in several bytes. Veenman [2007] also looked at statistical methods for identifying fragments but used data surrounding the fragments being tested to provide a context. Providing information which places a fragment in context of its location and surrounding information is useful during an analysis as it may assist with identifying the likely type of smaller fragments and providing an overall timeline of the activity on the machine.

7.3.2 Neural Networks

Whilst tools currently used by analysts, such as Encase and FTK, rely on header and footer information for the majority of their carving, a criminal may alter such information to prevent automated software finding the files. If these tools were the only method of investigation on a case such evidence may not be discovered. *Neural Networks* are adaptive and research has shown they may identify tampered documents which are otherwise overlooked. Harris [2007] investigated the possibility for recovering binary file types, and whilst his results showed that such networks could identify several important file types, a number of factors may improve upon his research. Firstly, Harris used back propagation to adjust the weights on the network, and whilst this is a common algorithm for

adjusting weights, techniques such as those utilised by Fogel [2001], using evolutionary algorithms based on a Darwinian approach, could also have been employed and may have increased the adaptability of the network [Russell, 2003]. Harris also used a training set that was ten times larger than the testing set. Decreasing the size of the training set and altering the proportions of each file type may have increased the recognition for the file types, preventing the network from becoming over trained, losing the ability to generalise to new files.

7.3.3 Recovering deleted information from other operating systems

Whilst analysts are most likely to come across Windows operating systems, they may also see Mac and Linux based systems too [Burghardt, 2008]. Deleted files in OS X have been the subject of research, by establishing if any given sector which is known to contain a deleted file has been overwritten; it is possible to calculate the chance of recovering useful evidence from the file. This is a useful technique, as the method identifies where blocks containing deleted information should be and in comparing them to the volume bitmap it can see if they have been overwritten. The technique is also limited in that it relies upon a complete and accurate entry in the journal; a log of file changes on the disk, and would therefore not provide assistance in cases where this is corrupt.

7.4 Methodology

The aim of this chapter is to evaluate methods to identify thumbnail cache file fragments. The methods described earlier in this research were based upon the identification of file fragments which consisted of multiple clusters. This research focuses on identifying single fragments of thumbnail cache files, without using any information outside the fragment to assist with its identification; it has already been established that a single fragment may be all that remains of a file. Therefore an important consideration for this research is the identification of methods which could be adapted to identify single fragments of information without any supplementary information about the system.

Having described current methods for the identification of deleted file fragments it is possible to take a single method with reasonable performance statistics, adapt it for fragment identification and apply it to the thumbnail cache file type. However this method would establish only the ability of the method being implemented to identify thumbnail cache file fragments, whereas a comparative study of the most commonly implemented methods would assist in implementing a single method with higher rates of identification for thumbnail cache files. A comparative study also prevents bias on the results of this research which the use of a single method may give and provides an opportunity to evaluate the performance of the methods on a single file type. Therefore the results on the file type can be accurately compared, highlighting the strengths and weaknesses of each approach.

Trialling a variety of methods against the thumbnail cache may improve the accuracy of recovery and allow the corroboration of results. Based on the review of literature conducted earlier in this chapter the four types of methods chosen for evaluation in this research are: *Brute force*, *Structural and Syntactical*, *Statistical*, and *Neural Network*. Of each type a single method will be implemented; the specific implementations are discussed in later sections; each method will then be evaluated against the same test data set.

In order to identify thumbnail cache fragments using the four methods identified it is first necessary to determine the data set which will be used to train and test the ability of each method; this is discussed in **Section 7.5**. This is followed by a description of how each of the identified methods was adapted to enable single fragment identification; the sections also describe how each method was applied to the thumbnail cache file formats [**Sections 7.6 – 7.9**]. The results of each method are then described in **Section 7.10**; after the results the methods are compared and evaluated in **Section 7.11**. Finally a discussion of the

research in this Chapter is provided in **Section 7.12**, and the chapter conclusion is provided in **Section 7.13**.

7.5 Information to be carved

In previous chapters Ubuntu, Kubuntu and Windows 7 thumbnail caches have been researched; as a result of this research a corpus of data based on these three operating systems has been collected. The data corpus also contains a wide selection of user generated files which were collected for use during the previous experiments. This section describes the generation of the corpus and how it has been constructed for this research.

7.5.1 Corpus Generation

In order to find a method for identifying thumbnail caches the methods implemented on this chapter need to be tested on a set of data which is representative of real problems and independent from the methods created. This led to the creation of three data sets which were used during this research to test each identification method.

Data Set 1

In order to create the first data set all the live files were extracted from virtual machines created in **Chapters 5 – 6**; the live files were classified into sub-directories of the corpus by file extension. For Windows 7 this generated 4147076 individual files with a total size of approximately 500 GB; for Ubuntu 2197329 individual files with a total size of approximately 140GB. A further 250 GB of user generated content was added to the corpus; these 1095696 files were collected specifically for this research and contained a wide variety of user generated data. The files were either created using software tested during the

previous experiments or downloaded from file type specific searches made using Google.

Data Set 2

The second data set represented a real world case with a larger range of file types and usage. The live files were taken from both machines and organised in directories by extension. The data set which contained 1.3TB of files taken from the main computer used for the 18 month period before this research was also collected; as the computer had significant use over this period it had completed a large variety of tasks and contained a significant amount of user data. In addition in order to study the growth of the thumbnail cache in Windows 7 during this research, the thumbnail cache directory was made read only; this led to a thumbnail cache with a total size of 23 GB and 827 temporary thumbnail cache files being created. An 80 GB Ubuntu machine used during the same time period was also included in this data set; the machine had a similar range of usage and contained significant amounts of user generated data.

Data Set 3

The final data set consisted of the free space from all the storage devices used in Data Sets 1 and 2. This unknown data provides a second set of test data where little is known about the contents before the identification methods are used. The free space contains unknown file types which may generate false positive results from the identification methods. It also provides an opportunity to identify any thumbnail cache file fragments in unallocated space on the systems analysed during this research. In Windows 7 the free space from data set 1 was 490 GB, and set two provided 687 GB of data. The data collected from the free space would have been too resource intensive to analyse by hand, and is therefore 1177 GB of unknown data.

7.5.2 File Fragment Classification

In order to identify thumbnail cache file fragments each type of fragment requires a classification; each class represents a file fragment with different characteristics. Initially classifications for each operating system were considered individually; after refining the classifications for Windows 7 fragment identification 4 classes of fragment were created:

W1: The fragment is part of the *thumbcache_idx.db* file

W2: The fragment is part of an image thumbcache file and contains metadata

W3: The fragment is part of a thumbcache file and only contains image data

W4: The fragment is not part of the Windows 7 thumbnail cache

Visual thumbnails are generally stored in one of three image formats: JPEG, PNG and BMP; BMP is generally only used to store visual thumbnails in the smaller cache files and therefore it is most likely that image-only data fragments will only be present for JPEG and PNG image formats. Since PNG files are present in the Linux thumbnail caches this research will focus on JPEG images for W3.

In **Chapter 5** it was established that visual thumbnails in Linux are stored in individual PNG files and contain thumbnail specific metadata. For this research, this led to the creation of three classifications for Ubuntu and Kubuntu fragment identification:

L1: The fragment is part of a PNG visual thumbnail and contains metadata

L2: The fragment is part of a visual thumbnail and only contains image data

L3: The fragment is not part of the PNG thumbnail cache record

There is a possibility that a storage device may contain multiple operating systems, therefore it is appropriate to search for both Linux and Windows thumbnail cache fragments on a device. Whilst each operating system is stored in separate partitions, it is possible that this information may not be available on a damaged or partially overwritten device; therefore the area an analyst may have to file carve during an investigation may contain artefacts from multiple operating systems. It was also noted that the PNG file format is used to store visual thumbnail is associated with user generated files in all the operating systems being investigated. These considerations led to the classifications being merged to produce 6 classes of files for this research.

H1: *Thumbcache_idx.db* file

H2: Image thumbcache file and contains metadata

H3: Image thumbcache file and only contains JPEG image data

H4: Linux PNG visual thumbnail and contains metadata

H5: PNG visual thumbnail fragment and only contains image data

H6: Not part of a Windows or Linux thumbnail cache

7.5.3 Preparing the File Fragments

The aim of this research is to identify thumbnail cache file fragments; however this is rather broad and requires the terms *fragment* and *thumbnail cache file* to be further defined. For the purposes of this research a *fragment* will be considered to be a cluster of data; for this research a cluster has been defined

as 4096 bytes. The cluster size selected is the default cluster size currently adopted by Windows 7 installations using NTFS and Ubuntu installations using EXT4. As has been previously determined in **Chapter 2** there is no standard implementation of an operating system thumbnail cache and therefore trying to identify any *thumbnail cache file* fragments would be a broad aim. For this research it has been decided that each type of thumbnail cache structure identified in **Chapters 5 and 6** will be dealt with as a separate classification for identification; the structures for these implementations of the thumbnail cache are described in **Appendix A**.

In order to prepare the file fragments for this research, each of the files in the corpus was split into 4096 byte chunks; each chunk was put into a separate file with a header to enable it to be identified. The 64 byte header contained:

Signature ["FRAGMENT"]

Individual frag id [4 bytes]

Path of original file [44 bytes]

Place in file [4 bytes]

Classification [4 bytes]

Each file was named with the unique fragment identifier and an extension of "frag". Given the large number of fragments created by this process it was decided to store only 1000 fragments per sub-directory; separating the files into batches made it easier to navigate the files manually.

Table 7-1: Breakdown of file fragment types per classification

	Data Set 1	Data Set 2
H1	3218	3277
H2	768039	672922
H3	6774078	5971550
H4	63385	11677
H5	62429	10668
H6	225625589	357223017
Total Files	233296738	363893111

7.5.4 Distribution of File Types

The distribution of file fragment types in the 2 known data sets is shown in *Table 7.1*. This is followed by a breakdown of file fragment types which have similar characteristics to the thumbnail cache classifications [*Table 7.2*]; these file types are most likely to be falsely classified as belonging to the thumbnail cache.

Table 7-2: Breakdown of potentially similar file fragment types in H6

	Data Set 1	Data Set 2
Thumbs.db	972	1349884
Bmp	3999477	3895491
JPG	7167119	8244521
PNG	8855007	4436326
Office Files (OLE2)	6931152	5945872
PDF	5212690	3925449
TXT	5364586	2005622
Other H6 Files	188094586	327419852
Total H6 Files	225625589	357223017

7.5.5 Training and Testing Sets

The three data sets contain a wide variety of file types which may be found on user systems during an investigation. As it is not feasible to produce a data set which contains every type of file fragment which may occur on a user's system, a data set needs to be selected which can provide a wide variety of file types which are likely to be on a system under investigation. The corpus created for this research contained a wide variety of system and user generated file fragments. In order to evaluate the identification methods created during this research the data corpus is split into training and test sets.

The training set is used to assist each method with modelling based upon fragments with known classifications. The training set was made up of

approximately half the data in Data Set 1; using a random number generator half the fragments from each file type were selected. For the thumbnail cache, half the records for each classification of thumbnail cache data were selected. Where the number of fragments in a category was odd, the number of selected fragments was rounded up.

For methods 1 and 2, the training phase focused on determining which rules for classifying fragments were accepting false positives. After each time the methods were run, the logs were analysed to ascertain which rules were falsely classifying fragments. For example, was a specific structure used to classify a fragment also being detected in other file types? Once the rules generating false positives were identified the individual method could be adapted to strengthen the rule. This provided an opportunity to refine the classification rules based on the thumbnail cache information identified in **Chapters 5 and 6**. For the statistical and *Neural Network* methods, the training is described in their individual sections.

After the models had been trained using the training set data, the models were tested against the testing set. The testing set comprised the remainder of the files from Data set 1, and all the files in Data sets 2 and 3. The files in Data sets 1 and 2 had known classifications. In data set 3 the number of fragments in each classification was unknown; therefore the only information provided by each models for this data set was the number of correctly identified thumbnail cache fragments and the number of false positive identifications.

7.6 Brute Force Approach

This approach uses a list of known hexadecimal strings which can be used to classify potential files; if a potential fragment contains one of the known strings it is put in the relevant classification. In Windows 7, thumbnail cache files begin with the file signature IMMM or CMMM; the record signature CMMM also appears at the start of each record contained in the Image files. The keyword IMMM would identify the first cluster of the index file; given the structure of the records in the index any further clusters belonging to the index file could not be identified in this manner. Searching for the keyword CMMM may identify a significant number of thumbnail cache file fragments; the keyword CMMM should be present in most of the fragments which satisfy **H2**. If the metadata for a subrecord is split over a cluster boundary then a fragment may contain metadata without a header; the fragment may still contain an image file signature so checks could be performed for JPEG, PNG, and BMP file headers. The image data only fragments do not contain any keywords and therefore cannot be identified using this method.

Visual thumbnails in the Ubuntu and Kubuntu thumbnail cache files are stored in individual PNG files and therefore contain standard PNG identifiers such as chunk names [**Chapter 5**]; the files also contain thumbnail specific metadata chunks with identifiers that could be searched for. With the establishment of keywords and file signatures [*Table 7.3*] which could be used to search for thumbnail cache file fragments a simple program was created which takes a fragment of data at a time and examines it for the identified keywords. Any fragment which contained a keyword was then extracted and stored in a directory for the relevant classification; the classifications relating to each keyword are shown in *table 7.3*.

Table 7-3: A list of thumbnail cache keywords

Key Words Identified in Thumbnail Cache Files		
Ubuntu / Kubuntu		
Keyword	Hexadecimal	Classification
tEXtThumb	0x744558745468756D62	H4
ThumbnailFactory	0x5468756D626E61696C466163746F7279	H4
PNG	0x504E47	H2,H4
IDAT	0x494444154	H4
IEND	0x49454E44	H4
Windows 7		
Keyword	Hexadecimal	Classification
IMMM	0x494D4D4D	H1
CMMM	0x434D4D4D	H2
JFIF	0X4A464946	H2
BM6	0x424D36	H2
PNG	0x504E47	H2, H4
thumbcache	0x7468756D626361636865	H1, H2

7.7 Structural and Syntactical Approach

The *Structural and Syntactical* method relies on the information gathered about the different implementations of the thumbnail caches researched in **Chapters 5 and 6**; in which the structure of the files are identified along with the valid syntax for each part of the structure. In order to automate the *Structural and Syntactical* method, a program was written which begins by checking if the fragment starts with a known file signature. If the file signature is 0x434D4D4D or 0x494D4D4D then it may be part of a thumbnail cache file; if the fragment starts with a known file signature that does not belong to a thumbnail cache file then the fragment is automatically classified as **H6**. This is followed by the checks described in the sub-sections below starting with those for the **H1** classification; processing for each classification stops as soon as a check is failed. If all checks for a classification are completed successfully the fragment is classified as that type; if the fragment fails to be classified as **H1 – H5** it is automatically classified as **H6** with no further checks. The following sub-sections describe the characteristics of each classification of fragment which were used for validation in this method.

7.7.1 H1:

The index files used by the Windows 7 caches (*thumbcache_idx.db*) store 32 byte records with the first complete record starting at byte 24 of the cluster; a cluster can hold up to 127 complete records and may hold a file header and incomplete records. *Figure 7.2* shows the 4 possible structures of an index file fragment; the first complete record starts 24 bytes into the fragment. Each record should either be empty or complete; therefore the first check to identify if a fragment belongs to an index file is to apply the basic structure to the fragment. It was also shown that the 4 bytes of a record which show the position of the subrecord in the *thumbcache_sr.db* file should either contain all 0xFF or 0x00.

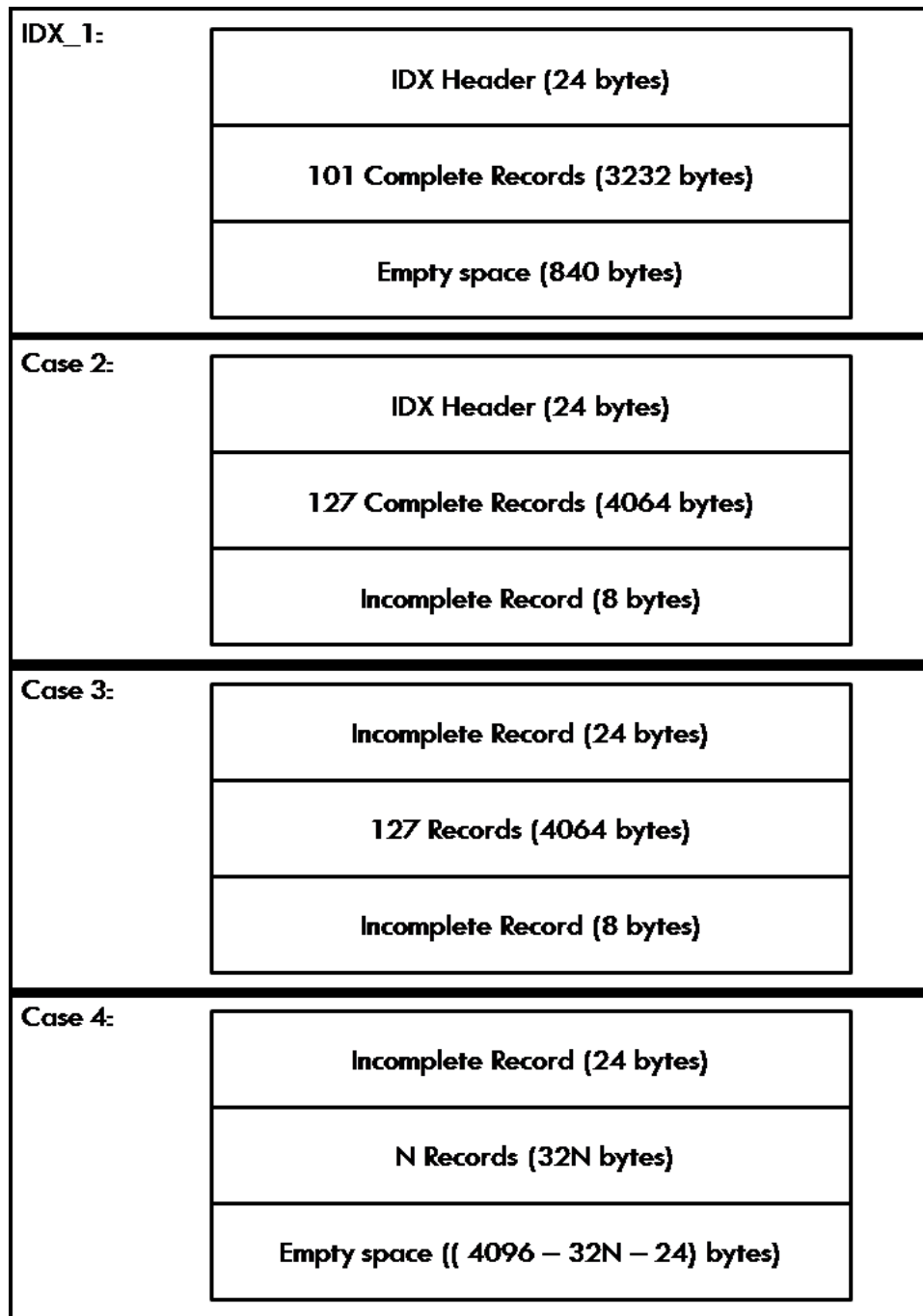


Figure 7-2: Classifications for thumbcache_IDX fragments

The next check to identify an index fragment involves the thumbcache ID values stored in each record; In **Chapter 6** it was established the index stores records in a hash table, the formula for working out the position of a record in a hash table is:

$$((X * 2^{32}) + Y) \text{ Mod } Z = \text{Position in hash table within } thumbcache_idx.db$$

Where

X is the first 4 bytes in little endian of the thumbcache ID

Y is the second 4 bytes in little endian of the thumbcache ID

Z is the number of positions currently in the hash table

[Chapter 6]

This formula can also be used to establish the possible table sizes for an index file fragment by substituting possible values of Z, which must be multiples of 101, into the equation to identify which, if any, values of Z solve the equation. Any value for Z which is valid for all the thumbnail cache ID's in the fragment can be considered a potential table size. Once a possible table size is established it can be verified by substituting it into the equation for other records in the same fragment. The identification of a valid possible hash table size also shows that the thumbcache ID values are syntactically valid and provides a substantial indication that the fragment belongs to a *thumbcache_idx.db* file.

After checking the thumbcache ID value, the remainder of the record structure is checked to ensure that the fragment follows the remainder of the structure and syntax identified in **Chapter 6**. The structure is followed in order starting with checking the flags are valid; if the record is complete the flags cannot consist solely of either 0xFF or 0x00. The remaining positions are then checked; these cannot be equal to 0 unless the record is empty. If a record fails any of the checks then the fragment is tested to see if it belongs to **H2**.

7.7.2 H2

For the identification of **H2** fragments a similar structural and syntactic validation is performed to the one used for **H1** identification. In the first instance fragments which contain the subrecord signature 'CMMM' are identified; the first instance of this signature within the fragment is used for classification. In this case the subrecord structure which begins with CMMM could start at any offset within the fragment; once the CMMM identifier is found each part of the subrecord structure can be validated for syntax against the structure in **Appendix A**. It is possible that a fragment may start part way through a subrecord header; this would mean that the subrecord signature may not be present. Given the fragment size of 4096 if the subrecord header was split over two fragments and the subrecord contained a visual thumbnail then the second fragment should contain an image file signature. If an image file signature is found the structural checks can be performed backwards heading towards the start of the file in order to identify if the fragment is likely to be part of a thumbnail cache image file.

In the third case the subrecord header is still split over 2 fragments, but the subrecord does not contain a visual thumbnail. In this case it is either immediately proceeded by another subrecord or the end subrecord; the header in these subrecords should ensure the fragment is subjected to the classification checks for this category. At the end of each thumbnail cache image file there is an end subrecord which does not have the same valid range of syntax as other subrecords. The last subrecord begins with the subrecord signature and 4 byte integer which stores the size of the subrecord. This is followed by 32 bytes of 0x00, and an 8 byte thumbnail cache ID value. If the last subrecord is stored in a fragment which contains other fragment headers then the last fragment will not be used for the classification checks. However if the last subrecord contains the only subrecord header in the fragment then the identified structure would be subject to classification checks.

The cases are checked in order from 1-4; for each case if any of the checks are failed then the case is rejected and the next case validation checks are performed. If the fragment fails the checks for all 4 cases the fragment is rejected for **H2**; the fragment is then tested for the **H3** classification.

7.7.3 H3

The **H3** classification relates to a JPEG thumbnail cache image fragment which does not contain metadata. Such fragments may not contain any information which an analyst could classify by a quick visual inspection. The JPEG file format is a binary file type that stores information in chunks; each chunk has an identifying marker to denote its type. The start of a visual thumbnail may occur in a fragment which also contains subrecord metadata; in this case the fragment would be categorised as **H2**. If the subrecord header is not in the same fragment as the start of the visual thumbnail, then the JPEG file signature 0xFFD8 would be identified; the data preceding the JPEG file signature could be checked to ascertain if it meets the structure and syntax of an image thumbnail cache file.

The first 623 bytes of each JPEG visual thumbnail from the thumbcache_256 files examined during this research was found to be the same; the first 623 bytes of JPEG visual thumbnails from the thumbcache_1024 files examined during this research were also found to be the same as each other. Whilst each size of JPEG visual thumbnail shared a common 623 bytes at the start there was a 3 byte difference between the two thumbcache files. Therefore 2 sequences of 16 bytes were selected from the common information; the sequences were searched for in the first 623 bytes of each fragment. If a sequence was identified, then the surrounding area of the fragment was checked to ensure it contained the expected sequence. The sequences and the relative offsets from the start of the visual thumbnail are shown below:

Relative offset: 48

0B 09 0C 11 0F 12 12 11 0F 11 11 13 16 1C 17 13

Relative offset: 592

E3 E4 E5 E6 E7 E8 E9 EA F2 F3 F4 F5 F6 F7 F8 F9

Later parts of the visual thumbnail occurring in image only fragments could not be identified using this structural and syntactic method as only the start of image markers were identifiable; however out of context it would be difficult to validate a fragment which appeared to contain a start of sequence marker. Such fragments are more suited to pattern matching techniques such as those used with the *Bayesian* and *Neural Networks* later in this chapter. If any of the checks identified in this section fail, the classification checks for **H4** begin.

7.7.4 H4

The **H4** classification relates to Linux PNG thumbnail cache files containing metadata; the first fragment of each file is likely to contain a substantial amount of identifiable artefacts which indicate the file is part of a thumbnail cache. For example **Chapter 5** describes the thumbnail cache specific chunks which generally contain the word 'thumbcache' and could be used to separate standard PNG files from PNG thumbnail cache files. The chunk format is described in **Chapter 5**; there is a CRC value at the end of every chunk which can be used to validate the content and the structure.

Other fragments belonging to a PNG thumbnail cache file may contain the IDAT PNG chunk marker; on analysis of the files found in **Chapter 5** it was noted that the IDAT marker occurred in every second fragment of a file. If the marker is found the data around it can be checked for structural and syntactic validation to enable it to be classified as **H4**. However these fragments can only be validated

to show they belong to a PNG file; this may not be a thumbnail cache file as they both have the same structure and syntax for the IDAT chunks.

7.7.5 H5

The **H5** classification relates to a PNG thumbnail cache image fragment which does not contain metadata. Such fragments do not contain any information which an analyst could determine by a quick visual inspection. The average PNG thumbnail cache file would be held in between 20 – 40 fragments. The first, last and every odd fragment contained within the file contain PNG markers that would be identified and checked for **H4**. The fragments which belong to this classification would just contain binary information, and like **H3** are more suited to pattern matching techniques for their classification. If a fragment is checked for **H5** classification it is automatically rejected and passed to **H6**.

7.7.6 H6

The combination of the classifications in **Section 7.5** to take into account the possibility of a storage device containing artefacts from multiple operating systems led to the creation of 6 classifications. For all fragments tested, each 4096 file fragment is tested for each classification of fragment in the order **H1-H6**; should a fragment fail to meet any of the first five classifications it is automatically allocated to **H6** and determined not to be a thumbnail cache file fragment. This is because it has failed to meet the expected structure and syntax for any of the thumbnail cache file fragment types.

7.8 Statistical Approach

The statistical technique chosen was a *Bayesian Network*, this enables the method implemented in the previous section to be expanded and developed with the aim of increasing the identification of thumbnail cache file fragments.

The *Bayesian Network* takes into account the probability that an artefact or set of artefacts appearing in a fragment relates to its classification. In order to construct the *Bayesian Network* originally each artefact relating to each hypothesis was shown in the network; however this led to the development of a complex network which required refinement.

7.8.1 Construction

In order to construct a *Bayesian Network* it was first necessary to establish characteristics which have a relationship with each of the classifications; the complete *Bayesian Networks* for the identification of thumbnail cache file fragments are shown in *figures 7.3 – 7.5*. As can be seen from the figures a selection of structural, syntactical and frequency based statistics were used to identify fragments; the related probability tables and a short description of each node can be found in **Appendix B**. It is interesting to note that in the original *Bayesian Network* all nodes were directly linked to the hypothesis nodes; however upon examination of the statistics it was noted that some artefacts have a greater impact on the validity of a hypothesis when combined together leading to the layers in the network. For each node with dependencies the probability table is created using the formula below.

$$p(X | Z) = \frac{p(X) p(Z | X)}{p(Z)}$$

In this research the *Bayesian Network* nodes will have Boolean outcomes; a node can resolve to either True or False. The equation provides the probability of a node resolving to True given the data that directly effects it within the network. X represents the node itself, while Z relates to any nodes directly influencing X. For this research, the syntax $p(X | Z)$ means the probability of X being True given Z is True). In the *Bayesian network* nodes can have multiple

parents; in this case Z would represent the state of the parent nodes. For example if a node had two parents, Z could represent one of four states ([True, True], [True, False], [False, True], [False, False]).

After the construction of the probability tables for each node the network could be used for to classify each fragment. In order to calculate the probability tables in **Appendix B** the training data was used. For each value with no parents the probability of the value occurring was established by calculating the number of fragments the value occurred in compared to the number without the value. For values with parents the probability of each set of parent states occurring while the node value was True was compared with the probability of the parent states occurring with the node being False. Therefore the probability of a set of states occurring in a thumbnail cache file fragment was compared with the probability of the states occurring in a fragment which does not belong to the thumbnail cache.

7.8.2 Using *Bayesian Networks* for Classification

The *Bayesian Network* shown in figures 7.3 – 7.5 and associated probability tables in **Appendix B** can be used to classify fragments. A program was constructed which identified whether each of the conditions of the independent nodes were True or False. This information was then used to calculate the probability of each of the classifications being true. A classification was deemed to be True if the probability of it being True, given the data, was above 0.75. If multiple classifications resolved to True, the one with the highest probability was accepted as the fragment classification.

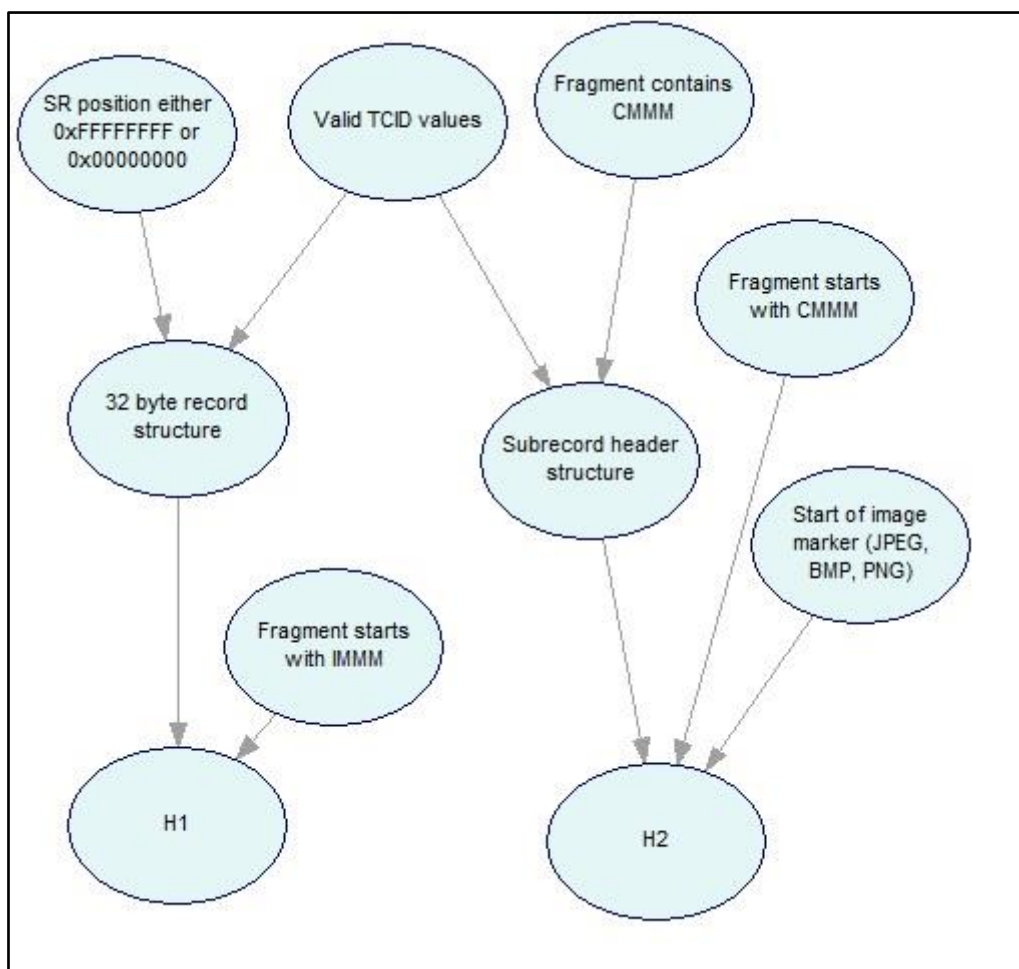


Figure 7-3: Bayesian network for H1 and H2 classification

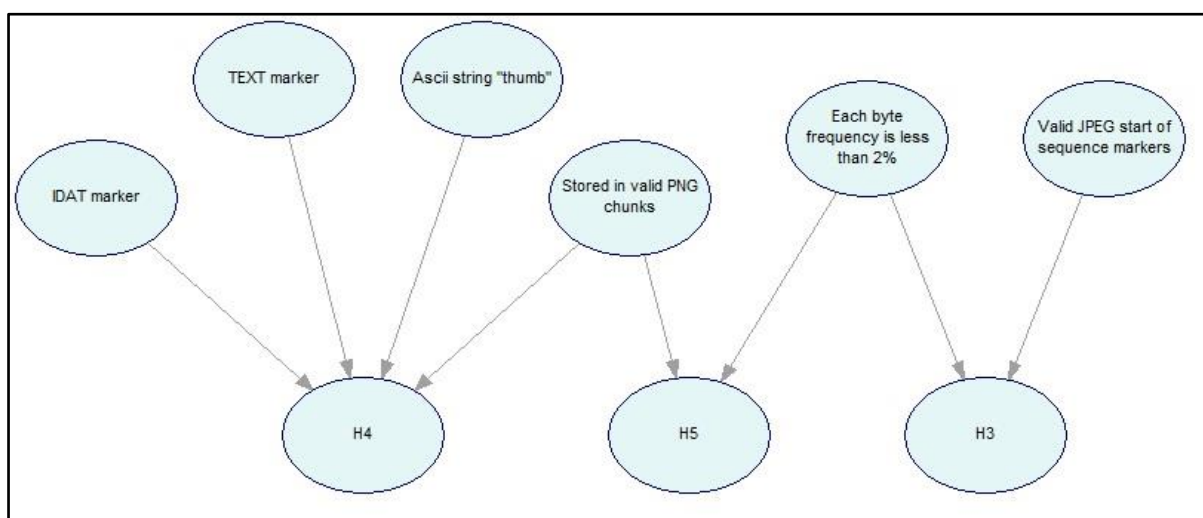


Figure 7-4: Bayesian Network for H3, H4 and H5 classification

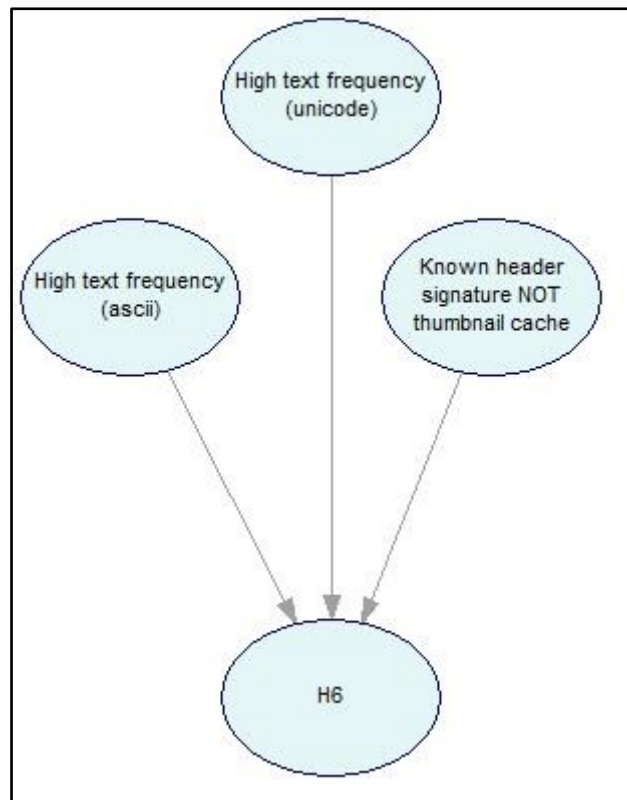


Figure 7-5: Bayesian Network for H6 classification

7.9 Neural Network Approach

The *Neural Network* method was implemented using a feed forward *Neural Network* with a single hidden layer. The network consisted of 17 input nodes which use the same input as the nodes identified in the *Bayesian Network*. The hidden layer consisted of 11 nodes, with 6 output nodes representing the various classifications. A sigmoid transfer function and back propagation were used for learning. The training set comprised of half of Data Set 1 as described in **Section 7.5**. The learning rate was initially set at 0.5 and training was set to stop when the error rate was 0.01 or 25000 iterations were reached. The remaining fragments in the corpus were used for testing the network. The implemented network was designed and written completely for this research; this enabled the method to be adapted to the specification of this research.

7.10 Results

This section contains the results from the last iteration of each method on the training set, followed by testing on the unused part of Data set 1, then against Data set 2 and Data set 3; as described in **Section 7.5** some of the methods were altered during iterations to take into consideration the results of the previous attempt and improve the methods accuracy.

7.10.1 Method 1: *Brute Force* approach

Table 7-4: Method 1 Results for the Training Set (Half of Data Set 1)

		Classification of each fragment by Method 1(Percentage)							
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	8	0	0	0	0	1601	1609
		%	0.4972	0.0000	0.0000	0.0000	0.0000	99.5028	100
	H2	No.	0	378980	0	0	0	5040	384020
		%	0.0000	98.6876	0.0000	0.0000	0.0000	1.3124	100
	H3	No.	0	0	0	0	0	3387039	3387039
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H4	No.	0	0	0	31693	0	0	31693
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	0	31215	31215
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H6	No.	32615718	3781375	0	5246610	0	71169092	112812795
		%	28.9114	3.3519	0.0000	4.6507	0.0000	63.0860	100

Table 7-5: Method 1 Results for the 1st Testing Set (Half of Data Set 1)

		Classification of each Fragment by Method(Percentage)							
Actual Fragment Types (Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	25	0	0	0	0	1584	1609
		%	1.5538	0.0000	0.0000	0.0000	0.0000	98.4462	100
	H2	No.	0	376150	0	0	0	7869	384019
		%	0.0000	97.9509	0.0000	0.0000	0.0000	2.0491	100
	H3	No.	0	0	0	0	0	3387039	3387039
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H4	No.	0	0	0	31692	0	0	31692
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	0	31214	31214
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H6	No.	33254086	2550791	0	5696645	0	71311272	112812794
		%	29.4772	2.2611	0.0000	5.0496	0.0000	63.2120	100

Table 7-6: Method 1 Results for the 2nd Testing Set (Data Set 2)

		Classification of each Fragment by Method(Percentage)							
Actual Fragment Types(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	24	0	0	0	0	3253	3277
		%	0.7324	0.0000	0.0000	0.0000	0.0000	99.2676	100
	H2	No.	0	665706	0	0	0	7216	672922
		%	0.0000	98.9277	0.0000	0.0000	0.0000	1.0723	100
	H3	No.	0	0	0	0	0	5971550	5971550
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H4	No.	0	0	0	11677	0	0	11677
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	0	10668	10668
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H6	No.	54874388	3924175	0	3566289	0	294858165	357223017
		%	15.3614	1.0985	0.0000	0.9983	0.0000	82.5418	100

Table 7-7: Method 1 Results for the unknown data set (Data Set 3)

Classification of each Fragment by Method(Percentage)							
	H1	H2	H3	H4	H5	H6	Total
No.	2	572	0	49	0	308544299	308544922
%	6.482E-07	0.000185386	0	1.588E-05	0	99.99979808	100

7.10.2 Method 2: *Structural and Syntactical* approach

Table 7-8: Method 2 Results for the Training Set (Half of Data Set 1)

Classification of each Fragment by Method(Percentage)									
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	1609	0	0	0	0	0	1609
		%	100.0000	0.0000	0.0000	0.0000	0.0000	0.0000	100
	H2	No.	0	384020	0	0	0	0	384020
		%	0.0000	100.0000	0.0000	0.0000	0.0000	0.0000	100
	H3	No.	0	0	51	0	0	3386988	3387039
		%	0.0000	0.0000	0.0015	0.0000	0.0000	99.9985	100
	H4	No.	0	0	0	31693	0	0	31693
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	0	31215	31215
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H6	No.	0	0	3583560	630989	0	108598246	112812795
		%	0.0000	0.0000	3.1766	0.5593	0.0000	96.2641	100

Table 7-9: Method 2 Results for the 1st Testing Set (Half of Data Set 1)

	Classification of each Fragment by Method(Percentage)								
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	1609	0	0	0	0	0	1609
		%	100.0000	0.0000	0.0000	0.0000	0.0000	0.0000	100
	H2	No.	0	384019	0	0	0	0	384019
		%	0.0000	100.0000	0.0000	0.0000	0.0000	0.0000	100
	H3	No.	0	0	76	0	0	3386963	3387039
		%	0.0000	0.0000	0.0022	0.0000	0.0000	99.9978	100
	H4	No.	0	0	0	31692	0	0	31692
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	0	31214	31214
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H6	No.	0	0	3981698	631732	0	108199364	112812794
%		0.0000	0.0000	3.5295	0.5600	0.0000	95.9105	100	

Table 7-10:Method 2 Results for the 2nd Testing Set (Data Set 2)

	Classification of Fragment by Method(Percentage)								
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	3277	0	0	0	0	0	3277
		%	100.0000	0.0000	0.0000	0.0000	0.0000	0.0000	100
	H2	No.	0	672922	0	0	0	0	672922
		%	0.0000	100.0000	0.0000	0.0000	0.0000	0.0000	100
	H3	No.	0	0	73	0	0	5971477	5971550
		%	0.0000	0.0000	0.0012	0.0000	0.0000	99.9988	100
	H4	No.	0	0	0	11677	0	0	11677
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	0	10668	10668
		%	0.0000	0.0000	0.0000	0.0000	0.0000	100.0000	100
	H6	No.	0	0	8244521	321139	0	348657357	357223017
		%	0.0000	0.0000	2.3079	0.0899	0.0000	97.6022	100

Table 7-11: Method 2 Results for the unknown data set (Data Set 3)

Classification of each Fragment by Method(Percentage)							
	H1	H2	H3	H4	H5	H6	Total
No.	71	874	2736	945	0	308540296	308544922
%	2.30112E-05	0.000283265	0.000886743	0.000306276	0	99.9985007	100

7.10.3 Method 3: Statistical approach

Table 7-12: Method 3 Results for the Training Set (Half of Data Set 1)

Classification of each Fragment by Method(Percentage)									
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	1567	0	0	0	0	42	1609
		%	97.3897	0.0000	0.0000	0.0000	0.0000	2.6103	100
	H2	No.	0	384020	0	0	0	0	384020
		%	0.0000	100.0000	0.0000	0.0000	0.0000	0.0000	100
	H3	No.	0	0	1540682	0	0	1846357	3387039
		%	0.0000	0.0000	45.4876	0.0000	0.0000	54.5124	100
	H4	No.	0	0	0	31693	0	0	31693
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	21866	9349	31215
		%	0.0000	0.0000	0.0000	0.0000	70.0497	29.9503	100
	H6	No.	105821	2931512	6589250	404	4747223	98438585	112812795
		%	0.0938	2.5986	5.8409	0.0004	4.2081	87.2584	100

Table 7-13: Method 3 Results for the 1st Testing Set (Half of Data Set 1)

		Classification of fragment by method(Percentage)							
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	1581	0	0	0	0	28	1609
		%	98.2598	0.0000	0.0000	0.0000	0.0000	1.7402	100
	H2	No.	0	384019	0	0	0	0	384019
		%	0.0000	100.0000	0.0000	0.0000	0.0000	0.0000	100
	H3	No.	0	0	2115649	0	0	1271390	3387039
		%	0.0000	0.0000	62.4631	0.0000	0.0000	37.5369	100
	H4	No.	0	0	0	31692	0	0	31692
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	0	0	21433	9781	31214
		%	0.0000	0.0000	0.0000	0.0000	68.6647	31.3353	100
	H6	No.	128791	2849734	6019246	91641	4622362	99101020	112812794
		%	0.1142	2.5261	5.3356	0.0006	4.0974	87.9262	100

Table 7-14: Method 3 Results for the 2nd Testing Set (Data Set 2)

		Classification of each Fragment by Method(Percentage)							
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	3261	0	0	0	0	16	3277
		%	99.5117	0.0000	0.0000	0.0000	0.0000	0.4883	100
	H2	No.	0	672922	0	0	0	0	672922
		%	0.0000	100.0000	0.0000	0.0000	0.0000	0.0000	100
	H3	No.	0	0	3346051	0	3820	2621679	5971550
		%	0.0000	0.0000	56.0332	0.0000	0.0640	43.9028	100
	H4	No.	0	0	0	11677	0	0	11677
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	4322	0	5489	857	10668
		%	0.0000	0.0000	40.5137	0.0000	51.4529	8.0334	100
	H6	No.	1	3	9879115	7509961	40845608	298988329	357223017
		%	0.0000	0.0000	2.7655	2.1023	11.4342	83.6979	100

Table 7-15: Method 3 Results for the unknown data set (Data Set 3)

Classification of each Fragment by Method(Percentage)							
	H1	H2	H3	H4	H5	H6	Total
No.	65	864	9193	3490	51532	308479778	308544922
%	2.10666E-05	0.000280024	0.002979469	0.001131116	0.01670162	99.9788867	100

7.10.4 Method 4: *Neural Network* approach

Table 7-16: Method 4 Results for the Training Set (Half of Data Set 1)

Classification of each Fragment by Method(Percentage)									
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	529	0	6	0	5	1069	1609
		%	32.8776	0.0000	0.3729	0.0000	0.3108	66.4388	100
	H2	No.	0	236144	0	0	0	147876	384020
		%	0.0000	61.4926	0.0000	0.0000	0.0000	38.5074	100
	H3	No.	0	0	2396283	0	84557	906199	3387039
		%	0.0000	0.0000	70.7486	0.0000	2.4965	26.7549	100
	H4	No.	0	0	0	31693	0	0	31693
		%	0.0000	0.0000	0.0000	100.0000	0.0000	0.0000	100
	H5	No.	0	0	259	0	26322	4634	31215
		%	0.0000	0.0000	0.8297	0.0000	84.3248	14.8454	100
	H6	No.	637	186	9056634	1485	7596948	96156905	112812795
		%	0.0006	0.0002	8.0280	0.0013	6.7341	85.2358	100

Table 7-17: Method 4 Results for the 1st Testing Set (Half of Data Set 1)

		Classification of each Fragment by Method(Percentage)							
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	472	0	1	0	6	1130	1609
		%	29.3350	0.0000	0.0622	0.0000	0.3729	70.2300	100
	H2	No.	0	306495	0	0	0	77524	384019
		%	0.0000	79.8125	0.0000	0.0000	0.0000	20.1875	100
	H3	No.	0	0	2502056	0	94470	790513	3387039
		%	0.0000	0.0000	73.8715	0.0000	2.7892	23.3394	100
	H4	No.	0	0	0	31691	0	1	31692
		%	0.0000	0.0000	0.0000	99.9968	0.0000	0.0032	100
	H5	No.	0	0	3809	0	22929	4476	31214
		%	0.0000	0.0000	12.2029	0.0000	73.4574	14.3397	100
	H6	No.	822	128	9348868	1268	7201152	96260556	112812794
		%	0.0007	0.0001	8.2871	0.0011	6.3833	85.3277	100

Table 7-18: Method 4 Results for the 2nd Testing Set (Data Set 2)

		Classification of each Fragment by Method(Percentage)							
Actual Fragment Type(Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	1226	0	3	0	9	2039	3277
		%	37.4123	0.0000	0.0915	0.0000	0.2746	62.2215	100
	H2	No.	0	395123	0	0	0	277799	672922
		%	0.0000	58.7175	0.0000	0.0000	0.0000	41.2825	100
	H3	No.	0	0	3991699	0	105340	1874511	5971550
		%	0.0000	0.0000	66.8453	0.0000	1.7640	31.3907	100
	H4	No.	0	0	0	11673	0	4	11677
		%	0.0000	0.0000	0.0000	99.9657	0.0000	0.0343	100
	H5	No.	7	0	630	0	8755	1276	10668
		%	0.0656	0.0000	5.9055	0.0000	82.0679	11.9610	100
	H6	No.	1373	2007	14580167	1875	18408479	324229116	357223017
		%	0.0004	0.0006	4.0815	0.0005	5.1532	90.7638	100

Table 7-19: Method 4 Results for the unknown data set (Data Set 3)

Classification of each Fragment by Method(Percentage)							
	H1	H2	H3	H4	H5	H6	Total
No.	28	253	9028	932	60974	308473707	308544922
%	9.07485E-06	8.19978E-05	0.002925992	0.000302063	0.01976179	99.97691908	

7.11 Comparison of Classification Methods

The methods described and implemented throughout this chapter have different strengths and weaknesses. Each of the four methods adapted and implemented during this research has shown to assist with the identification of thumbnail cache file fragments. *Table 7.20* shows the success and false positive rate for each method tested on data set 2. The table shows that the *structural and syntactical* method achieved 100 % success with no false positives for categories **H1** and **H2**; this method also achieved the highest success rate for **H4**. The highest success rate for **H3** was achieved by the statistical approach; the *neural network* achieved the highest success rate for **H5**.

The *Brute Force* approach provides a quick way of implementing a system to identify information; it relies on unique sequences which can be used to classify information. The sequences are identifiable to an analyst by a brief examination of the file type. The *Brute Force* method can only identify fragments containing known sequences; this relies on the analyst selecting a sequence which occurs in every instance of the classification they are attempting to identify. Depending on the frequency of the keyword, it is possible to attract a substantial amount of false-positives; whilst the amount of information is still reduced from the original data set the analyst may need to use other methods to further refine the potential information.

Table 7-20: Percentage Success and False Positive rate for each method for Data Set 2

		Brute Force	Structural and Syntactical	Statistical	Neural Network
H1	Success	0.73	100.00	99.51	37.41
	False Positive	15.08	0.00	0.00	0.00
H2	Success	98.92	100.00	100.00	58.71
	False Positive	1.08	0.00	0.00	0.00
H3	Success	0.00	0.00	56.03	66.84
	False Positive	0.00	2.30	2.76	4.07
H4	Success	100	100	100	99.96
	False Positive	0.98	0.09	2.06	0.00
H5	Success	0.00	0.00	51.45	82.06
	False Positive	0.00	0.00	11.23	5.09

The *Structural and Syntactical* approach requires a more in depth understanding of the file structure and valid syntax. The approach provides information which can be used in conjunction with the *Brute force* approach to further validate potential fragments and reduce the number of false positives. The use of this method requires a stable structure and any changes in both structure and potential syntax to be identified and added to the model; if the changes are not added then the method will not be able to identify those fragments. The structure and syntax is generally developed using known information; it is possible that the known data set which is used to create the model does not contain every variation of that fragment type and this may also lead to information being missed.

Both the *Brute Force* and *Structural and Syntactical* approaches performed badly when attempting to classify image only fragments; such fragments are

more suited to pattern recognition techniques. Methods such as the *Bayesian* and *Neural Networks* take in a variety of input and aim to put a fragment into the category it most likely belongs to. The techniques could not improve upon the *Brute force* and *Structural and Syntactical* approaches for identifying structured information with metadata; the pattern recognition techniques improved the identification of image-only data. Whilst given infinite resources it is possible for an analyst to examine and classify each fragment individually for **H1**, **H2**, and **H4**, it would be difficult for an analyst to identify fragments belonging to **H3** and **H5**. Such fragments which would be difficult for a human to identify have unique patterns which a machine can use to learn which classification is the most likely for a given input.

The *Bayesian* approach takes the likelihood of a node occurring in both the classification and in other circumstances into consideration to build a statistical model based upon the training data. The use of statistical modelling allows a fragment which does not perfectly fit the *structural and syntactical* model to be classified; this allows for variations outside the observed data. The *Bayesian network* is designed using expert knowledge about the domain to construct a model to form relationships between characteristics. The *Neural network* uses the same input information as the *Bayesian network* but allows the machine to interpret the relationship between features and classification. This suggests that the *neural network* has the capability to assign weight to relationships differently to an analyst which may assist with classification.

7.12 Discussion

The hypothesis of this research focused on the evidential value of thumbnail cache file fragments identified in unallocated space. The potential size of unallocated space on a storage device makes the individual analysis of each cluster infeasible; therefore a method for the automated identification of information was required. At the time of conducting this research there was little

information available about single fragment identification; particularly in cases where the fragment classification was based solely on the information within the fragment. This suggested that an existing method for the identification of larger chunks of data had to be selected and adapted for single fragment identification.

If a single method was selected and adapted then the research would be evaluating the ability of the adapted method to identify thumbnail cache file fragment. Therefore a comparative study was selected to attempt to maximise the identification of thumbnail cache file fragments. The study provided an opportunity to evaluate a variety of adapted methods on different thumbnail cache structures; this highlighted the strengths and weaknesses of using each approach.

The results of this research on data sets 1, 2 provided 100% identification of fragments in **H1** and **H2** with no false positives using the *structural and syntactical* approach; **H3** produced the least accurate results with a high rate of false positives and negatives. The evaluation of file fragment identification methods produced inaccurate results when identifying fragments which solely contained JPEG data; this is partly due to the fact it is difficult to distinguish between a JPEG fragment which belongs to the thumbnail cache and a JPEG fragment which belongs to any other file. For this research it was decided that it would be best to ensure all the JPEG fragments from the thumbnail cache were identified; whilst attempting to minimise the false positives; this proved difficult and most JPEG fragments were classified as thumbnail cache image fragments. Therefore further research is required into methods of identifying JPEG fragments in order to improve the accuracy of their carving.

The tables of results in **Section 7.10** show the actual fragment classification against the classification given to the fragment by each method for Data sets 1

and 2. This is possible because the fragments in this data set were taken from known live files; the files were split into fragments and each given a header with metadata to ensure a relationship with the original file was maintained. Data set 3 was a collection of the unallocated space from the machines used to provide the original two data sets.

Given the size of Data set 3 it was not feasible to individually analyse each cluster to ascertain its actual classification; therefore the only information provided after parsing this data set is the classified files. This data set represents a real world example of unallocated space with which to test the methods and identify information from an unknown collection of data. The data set also provided an opportunity to analyse the unallocated space for thumbnail cache artefacts; the results of this research show that the methods classified some information found in unallocated space as belonging to the thumbnail cache. On examination of the fragments identified from Data Set 3, information was recovered from deleted user accounts, and thumbnail cache files which were known to have been deleted from the systems.

This research has highlighted the increased identification for classifications of fragments which have been studied and can be formally described; therefore to identify fragments accurately it is necessary to understand the specific characteristics of the file type. The ability to accurately identify multiple types of thumbnail cache file fragments suggests that the approach documented can be generalised to other types of file.

None of the methods tested during this research were able to identify all the thumbnail cache file fragments without false positives; classifications **H1** and **H2** were identified with no false positives using the *structural and syntactical* approach. The lack of a completely successful method suggests there is room for improvement in the identification of single thumbnail cache file fragments. A

hybrid approach may improve the accuracy of identification, and would provide an opportunity to enhance the identification of **H3** fragments; *Table 7.20* shows **H3** had the lowest overall success rate during this research with the highest success rate for this classification being 66.84%. The results tables in **Section 7.10** show that for most of the methods and data sets, the **H3** fragments consistently have low rates of identification.

7.13 Chapter Conclusion

This chapter described research into the identification of thumbnail cache file fragments; this was achieved by implementing a variety of the most common file carving methods and evaluating their performance on a corpus. If a single file carving method had been implemented the results of the identification would have only provided an evaluation of the method itself; therefore using several methods provides the opportunity to enhance identification. The evaluation of the file carving methods also provides a unique opportunity to compare the strengths and weaknesses of each file carving approach.

The results of the comparative study showed that the **H1** and **H2** classifications could be successfully identified with no false positives in the corpus. Whilst it is noted that fragment types may exist which were not present in the corpus the high identification rate suggests a significant reduction in the number of fragments an analyst would have to analyse. A hybrid approach may improve classification for **H3-H5** and to improve the overall performance of the method. Since image-only thumbnail cache fragments have similar characteristics to standard image fragments it would be difficult to completely remove false positive matches. The aim of this research is to identify fragments which can be used for the reassembly research in **Chapter 9**; therefore a solution would be to maximise the identification of thumbnail cache file fragments whilst minimising false positives.

This chapter has successfully identified some thumbnail cache fragments; as the methods implemented do not identify all the thumbnail cache file fragments it is possible a piece of relevant evidence may not be found. The results on data set 3 show the potential for the identification of thumbnail cache file fragments from unallocated space and have also highlighted some circumstances where evidence may be recovered which is not available in the live file set. The classification of image-only fragments was the least successful; the next chapter focuses on the problem of image-only fragment identification and the creation of a hybrid approach. It aims to improve upon the classification of thumbnail cache file fragments in this research.

8 The creation of a Hybrid Identification Approach

8.1 Introduction

In **Chapter 4** a set of criteria for establishing evidential value were established; one of the criteria was maximising the recovered data. Therefore in order to establish the evidential value of thumbnail cache information in unallocated space it is necessary to maximise the data recovered.

The four file fragment identification methods implemented in the previous chapter had success rates which had room for improvement; therefore it is proposed to develop an approach based upon a hybrid of the previously implemented methods to improve the identification statistics. In the previous chapter the lowest identification statistics were achieved for image-only fragments identification. It was determined that further research was required to create an implementation which takes into account the features of image-only fragments may improve the identification. The methods implemented in the previous section highlighted the impact understanding the file format can have on the amount of data recovered.

This chapter describes the research performed into improving the identification of thumbnail cache file fragments; it begins by establishing a problem definition [**Section 8.2**], this is followed by a brief overview of the methodology for this research [**Section 8.3**]. **Section 8.4** describes related research into JPEG file recovery; the method implemented for identifying image-only fragments is documented in **Section 8.5**. In **Section 8.6** the new approach for thumbnail cache file fragment identification is described. The results of this research are shown in **Section 8.7**; this is followed by a discussion on the research [**Section 8.8**] and the conclusion of this chapter [**Section 8.9**].

8.2 Problem Definition

In the introduction it was established that the research described in the previous chapter did not produce a method which could identify all the thumbnail cache file fragments without producing any false positives. Therefore there is the potential for further research into improving the identification of thumbnail cache file fragments; the completion of this further research assists with meeting the criteria for maximising the data recovered. In order to compare the results of this research with the results recorded in **Chapter 7** the same classifications will be used, these are listed below for completeness.

H1: *Thumbcache_idx.db* file

H2: Image thumbcache file and contains metadata

H3: Image thumbcache file and only contains JPEG image data

H4: Linux PNG visual thumbnail and contains metadata

H5: PNG visual thumbnail fragment and only contains image data

H6: Not part of a Windows or Linux thumbnail cache

Within the thumbnail caches examined in **Chapters 5 and 6** of this research each visual thumbnail is stored in a standard image format; this means thumbnail cache image only fragments and standard image only fragments have the same structure and characteristics. Whilst the system visual thumbnails may provide useful artefacts such as a sample of the files within a directory, focusing on identifying JPEG fragments provides interesting opportunities for the generalisation of this research. Image fragments in JPEG format could also belong to user created images such as those created by cameras or downloaded from the internet; therefore the ability to identify these fragments would be of significant interest to the forensic computing community. The results for the identification of **H3** fragments were consistently the lowest,

and had a high number of false positives; therefore further research into the format of **H3** fragments is required to assist with maximising the data recovered.

Each method used in the previous research had different strengths; methods 1 and 2 produced the most accurate results on structured information with a known syntax. Method 3 produced accurate results with a file which had a known but variable structure; both methods 3 and 4 improved the accuracy of pattern matching for the image only fragments. If the strengths of each method were utilised it may be possible to construct a single approach to identifying thumbnail cache file fragments. In order to compare any new implementation to the previous methods the information to be carved should remain as described in **Section 7.5**. In order to successfully maximise the data recovered and maximise the relationships formed it is necessary to both identify thumbnail cache file fragments and to reassemble the files where possible. The reassembly of a thumbnail cache file would assist with forming relationships between artefacts and provide contextual information about any artefacts recovered. Therefore it is necessary to consider the potential input requirements of reassembly methods to ensure the output produced from this research is suitable for the next stage of this research project.

For this research it is necessary to consider the feasibility and performance of the reassembly approach. If fragments have not been successfully identified then it will not be possible to completely reassemble the information; this suggests that it is better to identify all relevant fragments and have some false positives than to fail to identify a relevant fragment. However a large number of false positives can reduce the feasibility of the reassembly method. For example a large number of false positives may result in a substantial number of comparisons which may significantly impact the processing time required to complete the reassembly process. It may also affect the number of false positive reassembled files produced. This would mean an analyst had more

information to examine at the end of the process; the large amount of information may mean that a piece of important information is overlooked.

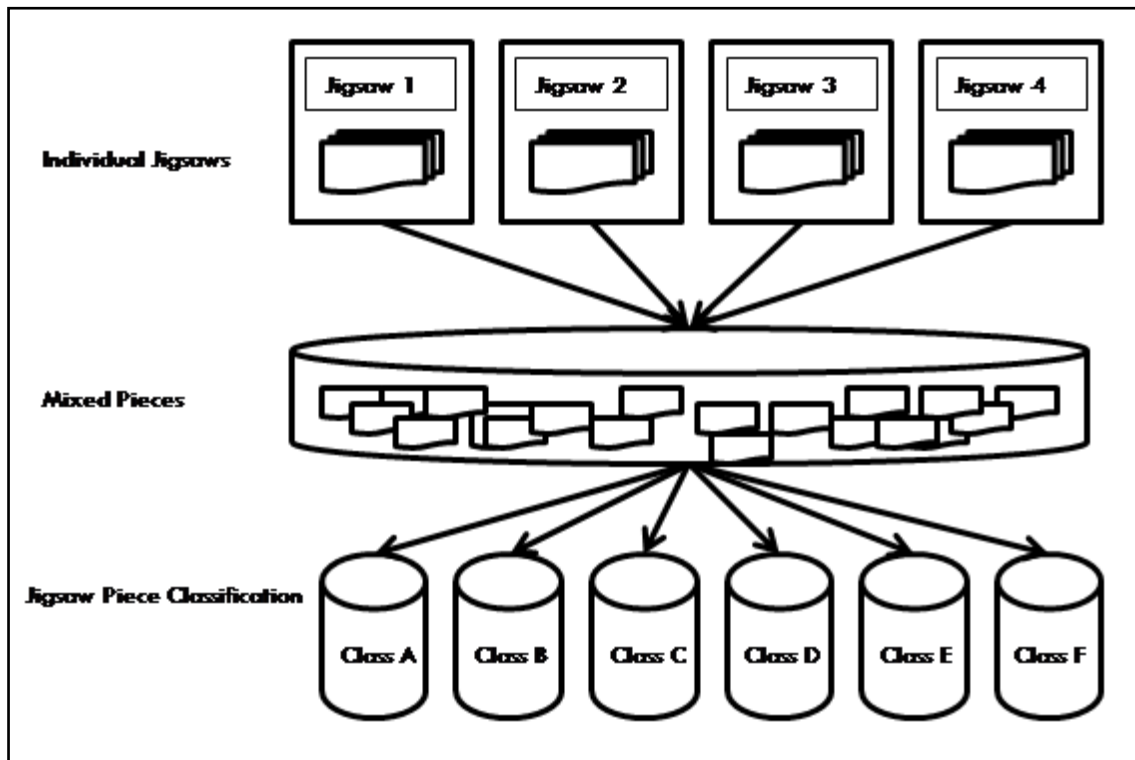


Figure 8-1: Jigsaw piece classification

The reassembly problem can be thought of as a tub full of jigsaw pieces; the pieces are from 1000 jigsaws. The aim is to build a single jigsaw, in order to do so the best use of resources would be to refine the number of pieces likely to belong to the jigsaw as shown in *Figure 8.1*. Refining the number of possible pieces ensures fewer comparisons are carried out during the reassembly process. In order to complete the jigsaw successfully it is better to have too many pieces than too few; whilst attempting to ensure the feasibility and processing required to complete the task is acceptable. Therefore the identification method needs to maximise the thumbnail cache file fragments which are found, whilst attempting to minimise the false positives.

8.3 Methodology

The problem definition highlighted two areas for investigation in this research. Firstly methods to improve the identification of **H3** fragments will be considered; as these are JPEG image data only fragments standard image fragment recovery techniques can be considered. After research relating to JPEG fragment identification has been considered a new method for **H3** identification will be implemented. This will be followed by the construction of a hybrid approach for thumbnail cache file fragment identification; the implementation will take the strengths of the four methods evaluated in the previous chapter and aims to improve the statistics for identification.

In order to conduct this research the three data sets constructed in **Chapter 7** will be used to evaluate the resulting implementation of a thumbnail cache file fragment identification method. Previous work has highlighted the importance of understanding the structural characteristics of files in order to identify potential fragments and this will be considered throughout the construction of an identification method for each classification. The research conducted to implement a new method for **H3** identification will be used in the hybrid approach constructed in **Section 8.6**. The results of the hybrid approach will then be described in **Section 8.7** and evaluated against the identification methods constructed in **Chapter 7 [Section 8.8]**.

8.4 JPEG Identification Research

One of the most common image formats is JPEG; JPEG is a compression method which is commonly used for images such as those on web pages and those taken with digital cameras. Like PNG images, JPEG files are split into sections, however the sections do not contain information about the length of their content. Each section begins with a marker to denote its start and type; since the file type is less structured than a PNG file the markers can occur at any point and therefore at any point in a file fragment. The structure of JPEG

files is described on the Joint Photographic Experts Group website [JPEG, 2013]. Using the *brute force* approach of searching for known signatures and strings present in a file it is possible to identify a JPEG header by searching for the signature string 0xFFD8.

A thumbnail stored in JPEG format is likely to be stored over several clusters; clusters which contain the middle of a JPEG file may not have any easily identifiable features and therefore a tailored identification approach is required. As a visual thumbnail may begin at any point in a cluster it is necessary to perform an exhaustive byte by byte search for the header signature in order to identify the start of the image.

It is possible to successfully decode a potential JPEG fragment that does not belong in the current file sequence; identifying the wrong fragment increases in likelihood when there is a substantial amount of similar images involved. One possible solution to this problem is to analyse the changes in the intensity of the pixels around the join of the clusters; a large change in pixel intensity suggests the clusters should not be reassembled in that pattern [Pal, 2008]. The process of identifying and reassembling JPEG file fragments can be considered process intensive because of the requirement to decode each potential new cluster sequence.

JPEG identification techniques typically combine identification and reassembly of the file into a single phase where each cluster of the file is searched for in order. When a potentially matching cluster is identified JPEG decoding is performed on the current file fragments; if no error occurs during decoding the fragment is added to the file sequence and the next cluster in the sequence is searched for [Garfinkel, 2007].

A bit pattern matching technique proposed by Sencar [2009] aimed to identify JPEG fragments without relying on a chain of clusters from the file. This technique could be used for classification of potential JPEG fragments which would significantly reduce the potential trial matches required during the reassembly process. Guo [2011] described a technique for recovering JPEG files based upon their visual thumbnail; the technique uses visual thumbnails stored in thumbnail caches. This technique could be reversed to assist with identifying thumbnail cache fragments. For example, the original file could be processed using the thumbnail creation methods to provide a visual thumbnail, this visual thumbnail could assist with locating and reassembling pieces of the original thumbnail record.

8.5 Visual Thumbnail JPEG File Fragment Identification

As described in **Section 8.4** previous research has been conducted into JPEG file fragment identification; this research can be applied to thumbnail cache JPEG only file fragments as they have the same characteristics. In this research the aim is to identify single 4096 byte JPEG image only fragments without the use of supplementary system information. As most of the techniques in JPEG identification combine the identification and reassembly phases they cannot be directly applied to this research; this research does not assume further clusters relating to the JPEG fragment exist and therefore reassembly of the original file may not be an option.

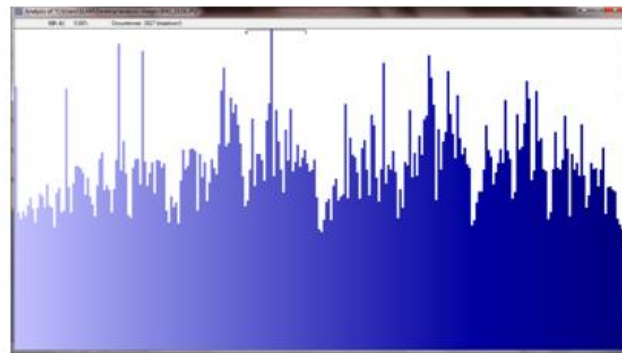
From the previous research it was established that understanding the structure and expected characteristics of a JPEG can assist with identification; therefore this research will focus upon identifying characteristics of JPEG file fragments. From Data set 1, described in **Chapter 7**, 1000 JPEG only thumbnail cache file fragments were selected. A Python program was used to randomly select the fragments from the **H3** category. A further 1000 JPEG only file fragments which

were not from thumbnail cache files were randomly selected for comparison from data set 1.

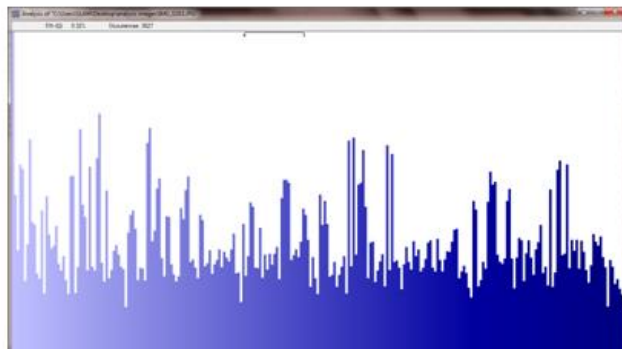
A small software program was implemented to analyse the frequencies of byte patterns for up to 4 bytes to ascertain if any characteristics could be identified for JPEG fragments. It was found that for single byte frequencies 0x00 is often quite high, with 0xFF generally being one of the less frequent bytes; four example images and their associated single byte frequencies are shown in *figure 8.2*. For two byte frequencies there are interesting characteristics for those starting with 0xFF; for example the set of bytes 0xFF FF does not appear in any of the JPEG image-only thumbnail cache fragments. However the sequence 0xFF FF did occur on multiple occasions in the non-thumbnail JPEG fragments. This interesting characteristic is due to the use of 0xFF as the first byte of a start of sequence marker; therefore only sequence markers used in the JPEG specification are valid in the JPEG clusters.

A python program was written to extract each 2 byte sequence starting with 0xFF found in the Windows 7 JPEG visual thumbnails. The program found eight unique 2 byte sequences, which are documented in *table 8.1*; the function of each byte sequence is also described [Cuturicu,1999; Weeks, 1998]. Out of the identified byte pairs, six have static positions relative to the start of the visual thumbnail; therefore only 0xFF 00 and 0xFFD9 can occur after than the first 611 bytes of a JPEG visual thumbnail. As the first 623 bytes of a visual thumbnail are mostly consistent if a marker is identified it is possible to check the bytes around it and validate it occurs at the right relative offset in the visual thumbnail. If the byte sequence 0xFFD9 occurs this should be directly followed by a subrecord header; therefore this byte sequence is unlikely to occur in a **H3** fragment.

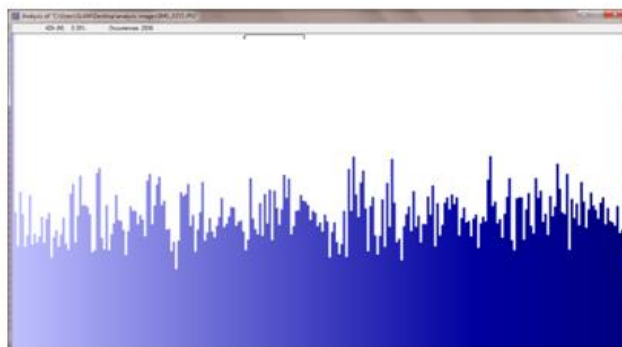
Case 1:



Case 2:



Case 3:



Case 4:

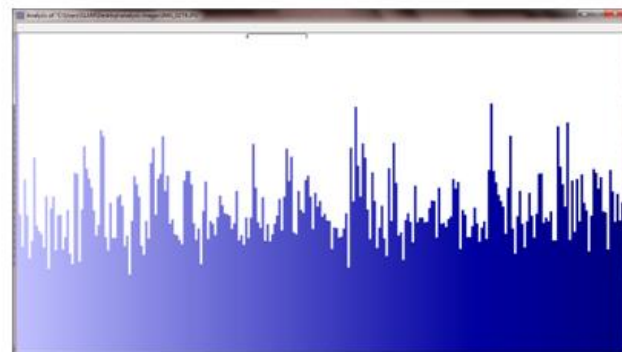


Figure 8-2: Single byte frequencies for 4 visual thumbnails

In order to identify potential **H3** fragments the first check used is to identify if the byte 0xFF occurs; if the byte occurs then it is checked to ensure it is followed by one of the 8 valid bytes. If a 0xFF is found which is not followed by one of the valid byte structures then the fragment is rejected. If a valid byte pair is identified then the area around the byte sequence can also be checked; if the byte sequence is one of the first six in *table 8.1* then the information around the byte can be used to verify it occurs at a valid position. In order to assess if the markers were the only ones used in thumbnail cache JPEG images every **H3** fragment in data set 2 was analysed. The results supported the hypothesis that only the subset of markers found in *table 8.1* are found in JPEG thumbnail cache images.

The byte frequencies identified were also compared with non-JPEG file fragments to identify characteristics which could be used to assist with the classification of JPEG fragments. The byte frequencies for JPEG fragments observed during this research fell throughout the byte range; therefore each potential fragment can be checked to ensure the byte frequencies are spread over the byte range and not heavily favouring a sub-set. Fragments with inappropriate structures tend to favour specific byte range sub-sets such as fragments containing ASCII or Unicode text strings; these fragments can be rejected as thumbnail cache image fragments for JPEG thumbnails do not contain metadata.

On analysis of the single byte frequencies 91.197% of the JPEG visual thumbnails analysed had a maximum single byte frequency of 2.16%; a further 8.802 % had a maximum single byte frequency of 4.2%. Interestingly images which had unusual frequencies were ones which consisted of between 1 and 4 colours. These images did not have a distributed byte frequency; instead several of the bytes had high frequencies, ranging up to 20% of the total whilst up to 14% of the bytes had a frequency of 0%. For this research as the majority of images had single byte frequencies under 4.2% this figure will be used as a

check for each potential fragment. If a byte frequency is over the cut of, or a byte has a frequency of 0% the fragment will be rejected.

Table 8-1: JPEG sequence markers found in Visual thumbnails

Byte Sequence	Meaning	Relative Position(s) in visual thumbnail
FFD8	Start of JPEG image	0
FFE0	JFIF marker	2
FFDB	Define Quantization table marker	20, 89
FFC0	Start of frame marker	158
FFC4	Define Huffman table marker	177, 210, 393, 426
FFDA	Start of Scan marker	609
FF00	0xFF with filling byte 0x00; read as just 0xFF	Anywhere in the scan sequence
FFD9	End of JPEG Image	End of visual thumbnail

In order to construct a method for identifying **H3** fragments the checks detailed in this section were used to pre-process the data. Any fragments which met the checks were used as potential candidates for the *neural network*. The *neural network* architecture is the same as the one used in **Section 7.9**; a copy of the *neural network* implementation was created with a single output node and re-trained using the pre-processed data. The sigmoid transfer function and back

propagation were again used for learning. The training set consisted of a randomly selected third of the fragments which were considered valid candidates after pre-processing; equal percentages of each file type in the pre-processed data set were used. As in **Chapter 7** the learning rate was set at 0.5 and training was set to stop when the error rate was 0.01 or 25000 iterations were reached. The remaining fragments in the pre-processed data set were used for testing the network.

Table 8-2: Results of visual thumbnail JPEG method

	Classification of fragment by method (Percentage)				
Actual Fragment Type (Percentage)			H3 Fragment	H6 JPEG Fragment	Total
	H3 Fragment	No.	755	245	1000
		%	75.5	24.5	100
	H6 JPEG Fragment	No.	391	609	1000
		%	39.1	60.9	100
	Total		1146	854	

Table 8-3: Percentage success and false positive rates for the H3 method

		JPEG identification method
H3	Success	75.5
	False Positive	39.1

The results of the **H3** identification method are in *table 8.2*, and the success rate is shown in *table 8.3*. The results show a success rate of 75.5% for the method; the rate reflects the amount of JPEG fragments which do not belong to the thumbnail cache which contain markers not found in *table 8.1*. The markers identified were the only ones found in the thumbnail cache file fragments analysed.

8.6 A hybrid approach for thumbnail cache identification

The combined approach aims to take the strengths of each of the individual methods and combine them to create a hybrid approach with improved classification statistics. Whilst the bases of the combined approach are the methods previously implemented in **Chapter 7** other information is also added to the individual approaches as appropriate to improve their performance. The complete hybrid method takes in a cluster which comprises of 4096 bytes; the method can also take disk images which once added are broken down into 4096 byte fragments to send to the identification method. Each cluster is individually assessed for classification using the hybrid method. The method is broken down into 6 testing stages. In general each testing stage is followed sequentially, however depending on validation checks performed at each stage a fragment may be classified or miss classification stages. If at any stage a fragment is identified as not belonging to a thumbnail cache it is rejected and categorised as **H6**. The process is summarised in *figure 8.3*; each stage of the identification method is described below.

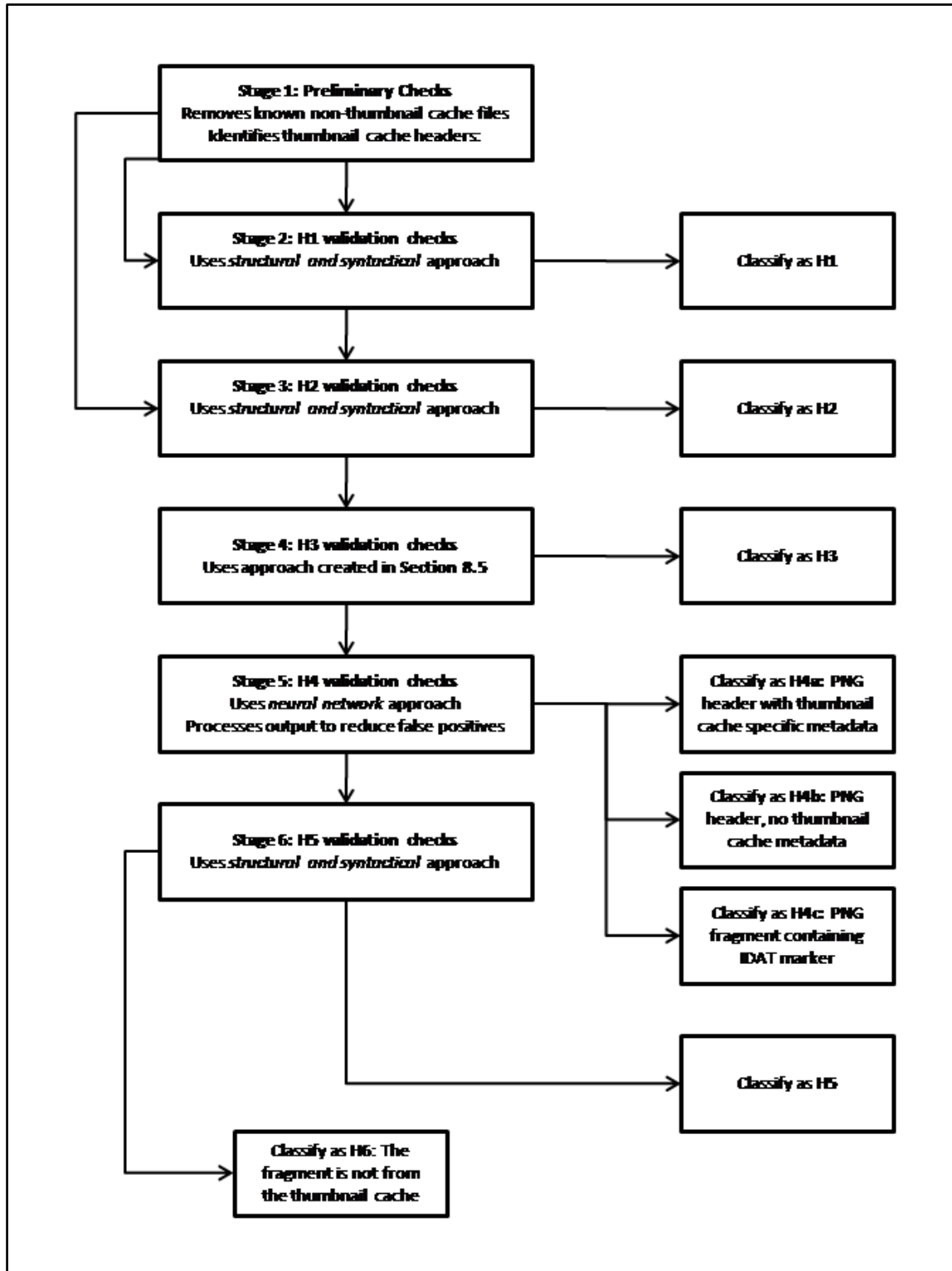


Figure 8-3: A summary of the hybrid approach

8.6.1 Stage 1: Preliminary Checks

Each fragment starts in the 'preliminary checks' section which aims to quickly remove fragments which are not part of the thumbnail cache. Firstly the fragment is assessed to see if it contains a known file signature; if the signature is from an irrelevant file type it is automatically rejected. If a file signature header of "IMMM", "CMMM" or "PNG" was identified the fragment was passed directly to **Stage 2, 3, and 5** validation checks respectively; the values represent header signatures which are found at the start of thumbnail cache files. Known file types were established using the known file signatures listed in the WinHex 16.0 file "File Type Signatures Search.txt". Further potential file system fragment headers were also included such as the start of an NTFS volume "0xEB52904E544653" and the MFT record header "0x46494C4530"; this was based on the information provided about file systems by Sammes [2008] and was used to exclude known **H6** fragments.

The second check identifies whether the fragment contains solely a single hex value for example 0x00 or 0xFF; as thumbnail cache fragments are not likely to be made up solely of a single value such fragments are automatically rejected. Thirdly checks are performed to ascertain whether the fragment consists mostly of ASCII or Unicode values. In order to perform these checks the hex values of the fragment are considered. If 90% or more of the bytes in the fragment fall between 0x41 and 0x7A the fragment contains mainly ASCII characters and is rejected. If 80% or more of the double bytes in the fragment fall between 0x0030 and 0x007A the fragment contains mainly Unicode characters and is rejected.

The final preliminary check is to ascertain if the fragment has a compound structure; file types can be contained within compound structures and therefore a header signature may appear anywhere within a fragment. Only image file types (JPEG, BMP, and PNG) are valid headers to be found within thumbnail

cache file fragments. The headers identified in the first check using the WinHex file are used again for this extensive check of the entire fragment. If a header is found which is not part of a thumbnail cache file the fragment is rejected. As the Linux visual thumbnails are stored in individual PNG files the PNG header signature will only be found at the start of a fragment and would therefore have been identified in the first check of this stage. If the fragment contains a JPEG, PNG or BMP header the fragment is passed to **Stage 3** as these are the file types used to store visual thumbnails in the Windows 7 thumbnail cache.

After the final preliminary check if the fragment has not been rejected or passed directly to another stage it is passed to **Stage 2**.

8.6.2 Stage 2: H1 Validation Checks

If the fragment completes the **Stage 1** checks and is not rejected or identified as containing a thumbnail cache file **H2** or **H4** file header then it is assessed for **H1** classification; given the ability of the *structural and syntactical* approach to identify **H1** fragments with no false positives it was decided to use the checks defined in **Section 7.7.1**. If these checks are all positive the fragment is classified as **H1**; if any check fails the fragment is passed to **H2**.

8.6.3 Stage 3: H2 Validation Checks

A fragment can be passed to **stage 3** by either **stage 1 or 2**. If the fragment is passed directly from **Stage 1** then it begins with the file signature “CMMM”; in this case if the fragment belongs to a thumbnail cache the file signature is either followed by the remainder of the thumbnail cache file header or the subrecord header structure. In **Section 7.7.2** the implementation for identification of **H2** fragments using the *structural and syntactical* approach was described. The method identified all the thumbnail cache file fragments in the three data sets used in the previous chapters without identifying any false positives. Therefore

the checks used in **Section 7.7.2** are used in this method for **H2** validation. If a fragment meets the criteria it is classified as **H2**, otherwise the fragment is passed to **stage 4**.

8.6.4 Stage 4: H3 Validation Checks

In **Chapter 7** for each of the four methods implemented the classification with the lowest identification rate was **H3**; this classification related to visual thumbnail JPEG image only fragments. Earlier in this chapter the research taken to attempt to improve the identification of **H3** fragments was described. The checks described in **Section 8.5** are used for **stage 4**. If the fragment fails **stage 4** it is passed to **stage 5**.

8.6.5 Stage 5: H4 Validation Checks

Originally a *Bayesian network* was used for the identification of **H4** fragments; this method was chosen over the structural and syntactic approach due to the variety of metadata a Linux thumbnail may contain. In **Chapter 7** the *Bayesian network* approach identified all the thumbnail cache file fragments but a significant number of false positives were also recovered. However the large amount of false positives would create significant performance problems for any reassembly approach; using this method would affect the feasibility of using this approach in cases as it would increase the resource requirements. The *neural network* did not successfully identify all the fragments for this category but had a 99.96% success rate for data set 2; the method also had a low false positive rate of less than 0.01%. Therefore on consideration of the feasibility of the reassembly of the information the *neural network* method was selected for this classification.

In **Chapter 5** it was established that the thumbnail cache used in Ubuntu and Kubuntu stores each visual thumbnail in a separate PNG file. These subrecords

follow the standard PNG specification but contain varying amounts of metadata. Once the *neural network* has classified a fragment as **H4** further checks are performed with the aim of reducing the false positives. Within the **H4** classification there are two distinct types of PNG thumbnail cache fragment which contain metadata. The first type is the first fragment of the individual PNG subrecord files; these fragments contain the PNG header and any metadata chunks that are present in the file. The second type of fragment occurs throughout the subrecord files and contains the IDAT chunk identifier; **Chapter 5** described how this identified appears in every other fragment of the file.

Both types of fragment may occur in standard PNG files as well as thumbnail cache files; therefore the removal of false positives is not possible. Instead this stage aims to minimise the number of false positives. If a fragment contains the start of a PNG file and contains thumbnail cache specific metadata it is classified as **H4a**. The IDHR chunk which is compulsory at the start of PNG files contains the width and height of the image; if these dimensions are greater than the large thumbnail cache image size the fragment is rejected. If the fragment contains the start of a PNG and metadata but does not contain any thumbnail cache specific metadata it is rejected. If the fragment contains the start of a PNG file which meets the image size requirements and contains no metadata it is classified as **H4b**. Fragments classified as **H4a** contain thumbnail cache specific data and are therefore considered thumbnail cache file fragments; any fragment classified as **H4b** is a potential candidate thumbnail cache fragment. Fragments which contain the keyword “IDAT” may also be categorised as **H4**; if the fragment also contains the PNG header it would have already been classified as **H4**. In order to separate the fragments for the reassembly method fragments just containing IDAT are classified as **H4c**.

If the *neural network* does not categorise the fragment as **H4** the fragment is rejected and passed to **Stage 6**.

8.6.6 Stage 6: H5 Validation Checks

The highest identification statistics for **H5** classification in **Chapter 7** were achieved by the *neural network*. Therefore for stage 6 of this method the *neural network* implementation used in **Section 7.9** was used; the fragment is provided as input for the *neural network*. If the fragment is classified as **H5** by the *neural network* it is accepted, otherwise the fragment is rejected from **stage 6**.

If the fragment fails the validation checks for Stage 6 the fragment is classified as **H6**; the fragment has failed to satisfy the validation checks for any of the thumbnail cache classifications and is therefore unlikely to be a thumbnail cache file fragment.

8.6.7 Logs

Each classification method logs the reasoning for rejecting or accepting a fragment; this is stored in a CSV file. The CSV file contains: the name and location of the fragment, its classification, the reason for classification, and any supplementary information. The additional information stores details such as if the fragment started with a thumbnail cache file header, or has been sub-categorised; the extra information is useful for file fragment reassembly. The logging was designed to assist an analyst with corroborating the results and to provide information on each fragment to assist with reassembly.

8.6.8 Accepted Fragments

Fragments which are awarded a classification of **H1 – H5** are identified as thumbnail cache file fragments. Each classification has a separate sub-directory; any fragment which is classified as belonging to the thumbnail cache is copied into the relevant sub-directory. It is the copy of the fragment which can

then be viewed by an analyst and used by the reassembly methods implemented in the next Chapter. When a fragment is copied it is given a file header which contains metadata about the fragment. The header contains:

Signature ["TCFRAG"]

Individual fragment identifier [8 bytes]

Path of original file [44 bytes]

Relative position in file [8 bytes]

Classification [4 bytes]

The file signature identifies the fragment as a thumbnail cache file fragment. The individual fragment identifier is a unique number based upon the order the fragments are processed; the first fragment of a batch to be processed is given the number 1. The path of the original file and relative offset in the file allows an analyst to maintain a relationship between each fragment and the original source file. The classification holds in ASCII the classification of the fragment in the first 2 bytes; the third byte is used to store the sub-category, and the fourth byte is set to 0xFF if the fragment begins with a thumbnail cache file header.

8.7 Results

This section contains the results of testing the hybrid thumbnail cache file fragment identification method against the data sets used in **Chapter 7**.

Table 8-4: Method 5 Results for the Training Set (Half of Data Set 1)

		Classification of fragment by method (Percentage)						
Actual Fragment Type (Percentage)		H1	H2	H3	H4	H5	H6	Total
	H1	No.	1609	0	0	0	0	1609
	H1	%	100.00	0.00	0.00	0.00	0.00	100
	H2	No.	0	384020	0	0	0	384020
	H2	%	0.00	100.00	0.00	0.00	0.00	100
	H3	No.	0	0	2990084	0	47781	3387039
	H3	%	0.00	0.00	88.28	0.00	1.41	100
	H4	No.	0	0	0	31693	0	31693
	H4	%	0.00	0.00	0.00	100.00	0.00	100
	H5	No.	0	0	140	0	26322	31215
	H5	%	0.00	0.00	0.44	0.00	84.32	100
	H6	No.	0	0	8454928	1038	6563467	112812795
	H6	%	0.00	0.00	7.49	0.00	5.81	100

Table 8-5: Method 5 Results for the 1st Testing Set (Half of Data Set 1)

		Classification of fragment by method (Percentage)							
Actual Fragment Type (Percentage)			H1	H2	H3	H4	H5	H6	Total
	H1	No.	1609	0	0	0	0	0	1609
		%	100.00	0.00	0.00	0.00	0.00	0.00	100
	H2	No.	0	384019	0	0	0	0	384019
		%	0.00	100.00	0.00	0.00	0.00	0.00	100
	H3	No.	0	0	2975951	0	44325	366763	3387039
		%	0.00	0.00	87.86	0.00	1.30	10.82	100
	H4	No.	0	0	0	31691	0	0	31692
		%	0.00	0.00	0.00	99.99	0.00	0.00	100
	H5	No.	0	0	147	0	22929	8138	31214
		%	0.00	0.00	0.47	0.00	73.45	26.07	100
	H6	No.	0	0	8482214	1131	6333494	97995955	112812794
		%	0.00	0.00	7.51	0.00	5.61	86.86	100

Table 8-6: Method 5 Results for the 2nd Testing Set (Data Set 2)

	Classification of fragment by method (Percentage)								
Actual Fragment Type (Percentage)			H1	H2	H3	H4	H5	H6	Total
	H 1	No.	3277	0	0	0	0	0	3277
		%	100.00	0.00	0.00	0.00	0.00	0.00	100
	H 2	No.	0	672922	0	0	0	0	672922
		%	0.00	100.00	0.00	0.00	0.00	0.00	100
	H 3	No.	0	0	5163582	0	102984	704984	5971550
		%	0.00	0.00	86.46	0.00	1.72	11.80	100
	H 4	No.	0	0	0	11673	0	0	11677
		%	0.00	0.00	0.00	99.96	0.00	0.00	100
	H 5	No.	0	0	644	0	8755	1269	10668
		%	0.00	0.00	6.03	0.00	82.06	11.89	100
	H 6	No.	0	0	10933051	1363	17589760	328698843	357223017
		%	0.00	0.00	3.06	0.00	4.92	92.01	100

Table 8-7: Method 5 Results for the unknown data set (Data Set 3)

Classification of fragment by method (Percentage)							
	H1	H2	H3	H4	H5	H6	Total
No.	71	874	2542	932	60974	308479529	308544922
%	0.000	0.000	0.000	0.000	0.019	99.97	

8.8 Discussion

The end aim of **Chapters 7-9** is to identify and reassemble thumbnail cache file fragments located in unallocated space in order to assess their evidential value. Given this aim it is more important at this stage in the research to have too many fragments available for the reassembly process in order to maximise the chance of a successful reassembly. This research successfully improved the number of JPEG image thumbnail cache file fragments identified; despite the large number of false positives it still represents a reduction compared to the results of **Chapter 7**. Despite the reduction the large number of fragments may still significantly impact the performance of any reassembly method created.

Further identification of JPEG specific fragment characteristics enabled the hybrid identification method created in **Chapter 7** to be modified with the aim of improving JPEG classification. When compared to the results of the previous chapter it is clear that a noticeable improvement in the identification of JPEG file fragments has occurred. This highlights the need to consider each different classification of file type independently; it also highlights the need to consider the requirements before determining appropriate classifications. For example in this research JPEG thumbnail cache file fragments are classified separately to JPEG file fragments from other sources, this resulted in a high level of false positive classifications. If the identification classification had been JPEG image-only fragments from any source the false positive rate would have been substantially lower. The same classification problem applies to PNG image-only fragments where a substantial amount of the false positive fragments identified were from other PNG files.

A decision was made in this research to use the observed subset of JPEG markers found in visual thumbnails instead of the full set of valid markers. Whilst this decision has improved the identification of **H3** fragments for this research and reduced the number of JPEG false positives it may not have the

same affect on other data sets. The alternative option would have been to accept all markers which are defined as valid by the JPEG specification. It is possible that whilst in the data sets collected by this research all JPEG visual thumbnails conform to the marker list and positions documented in table 8.1, other visual thumbnails may not; therefore this decision may exclude potential evidence. However given the size of the data set it is also possible that the visual thumbnails use a sub-set of markers as observed during this research. Further research into visual thumbnail markers would be required to ascertain whether the identification method should use the values observed during this research or accept any marker which is valid in the specification.

The hybrid approach significantly improved identification and produced logs which show the reasoning for accepting or rejecting each fragment. Neither the methods in the comparative study nor the hybrid approach achieved high classification rates for image-only fragments. Since image-only thumbnail cache fragments have similar characteristics to standard image fragments it would be difficult to completely remove false positive matches. Since the aim of this research is to identify fragments which can be used for the reassembly research in **Chapter 9** then a requirement is to maximise the identification of thumbnail cache file fragments whilst minimising false positives. *Table 8.8* shows the success and false positive rate of the hybrid approach.

This research highlights the improved identification of file fragment information by understanding the structure of each fragment classification and by combining methods. Unique characteristics identified for each classification may not be unique when implemented and tested against a large data set; with the variety of possible file fragment structures available it is likely that another file fragment type would also have the same characteristic. By combining multiple characteristics and methods for identification it is possible to reduce the number of false positive identifications whilst maximising the identification of fragments in the classification. The combination of approaches and breaking the

identification method into stages led to a structured identification approach which an analyst could repeat by hand to verify the classification of an evidentially significant fragment.

Table 8-8: Percentage Success and False Positive rates for the hybrid approach and the commercial recovery tool Blade for Data Set 2

		Hybrid Approach	Blade, Forensic Recovery Software
H1	Success	100.00	3.22
	False Positive	0.00	0.61
H2	Success	100.00	68.74
	False Positive	0.00	16.5
H3	Success	86.46	0.15
	False Positive	3.05	14.99
H4	Success	100	78.08
	False Positive	0.00	28.3
H5	Success	82.06	0.01
	False Positive	4.86	0.04
H6	Success	100.00	99.83
	False Positive	0.00	0.00

The hybrid method produces a log of the fragments it classifies, the log provides information on the original location of the fragment, the classification it was awarded and the reason for the classification. By providing an analyst with this log it is possible for the analyst to show a relationship between the original data and the artefact produced; it is also possible to ascertain the characteristics of the fragment which led to its classification. This log also can assist with providing information for the potential reassembly of file fragments; research into the reassembly of the fragments identified during this research is described in the **Chapter 9**.

The hybrid method was also tested against Blade, a leading forensic data recovery program (*Table 8.8*). The hybrid method achieved higher success rates and lower false positive rates in each category. This is due to the hybrid method performing a greater number of validation checks on the data.

8.9 Chapter Conclusion

This chapter improves the identification statistics for thumbnail cache file fragments by producing a single hybrid approach with the maximum data recovery for each classification. In order to improve the identification of JPEG only thumbnail cache fragments an examination of the characteristics in JPEG files determined a set of unique criteria which could be used for identification. The research used observed information about a subset of JPEG markers to reduce the false positives for the **H3** category; whilst the corpus conformed to the observed markers it is possible that Windows 7 visual thumbnails exist on other machines which use other markers. If other markers were used the fragments would not be identified. A hybrid approach was created which utilised the strengths of the methods implemented in **Chapter 7**; the approach also enabled other information learnt about the thumbnail cache file format during the previous research to be added.

This research resulted in a method which due to its structured approach could be easily verified by an analyst; the method was broken down into stages to assist with maintaining a clear relationship between the original artefact and the classification. The hybrid approach was tested on the same test set as the previous methods; the results showed the method had the highest identification rate for each classification. Image only fragments produced a significant amount of false positives; this may affect the approach taken to reassemble thumbnail cache files. The next chapter uses the results of both this and the previous chapter to construct reassembly methods for the identified file fragments.

9 Thumbnail cache fragment reassembly

9.1 Introduction

In **chapters 7 and 8**, the focus of the research was the identification of thumbnail cache file fragments; the fragments were classified without using supplementary information about the system. The research conducted in the previous chapters resulted in the identification of a significant amount of each thumbnail cache file fragment type under investigation. The classifications used in the research represented the different types of file fragment that were found in the thumbnail caches used for this research. The results of the identification process were stored in detailed logs to assist with verifying the results and maintaining a relationship with the original data; each classification of file fragment was stored in a separate sub-directory to assist with locating each type of fragment. This chapter uses the results of the previous research to investigate methods for reassembling thumbnail cache file fragments; the aim of this research is to produce a method which can reassemble thumbnail cache file fragments to reconstruct the original files.

The reassembly of thumbnail cache file fragments provides an analyst with the opportunity to identify a greater number of relationships with artefacts identified in thumbnail cache file fragments. A single thumbnail cache file fragment on its own may provide metadata, or part of a visual thumbnail; however when multiple fragments are reassembled it may be possible to extract complete subrecords. As described in the previous chapters some of the thumbnail cache file fragments are indistinguishable in classification from image fragments; this is because they are essentially the same structure and syntax. Therefore in order to maximise the identification of potential thumbnail cache file fragments there may be significant false positives for some classifications. One of the criteria for evaluating evidential value of thumbnail cache file fragments requires the data recovered to be maximised [**Chapter 4**]; therefore a valid

reassembly is more likely when there are too many fragments rather than too few. It should also be noted that the larger the number of fragments in a data set the more resources that will be required to complete the reassembly.

This chapter is structured as follows: firstly previous research into file fragment reassembly is considered in **Section 9.2**. The methodology for this research is presented in **Section 9.3**; the method for reassembling the thumbcache_idx files is described in **Section 9.4**. The method for reassembling thumbcache_32, thumbcache_96, and thumbcache_1024 files is described in **Section 9.5**. In **Section 9.6** the thumbcache_256 reassembly method is described; this is followed by the reassembly method for Linux thumbnail cache files in **Section 9.7**. The results of the implemented reassembly methods are provided in **Section 9.8**; **Section 9.9** provides a description of the information from thumbnail cache files reassembled from **Data Set 3 of Chapter 7**. A discussion of this research is provided in **Section 9.10**; the chapter conclusion is in **Section 9.11**.

9.2 Image Reassembly

In **Chapters 7 and 8** a significant proportion of the previous research reviewed used methods for file identification that combined the identification and reassembly process; the methods combining the steps tended to focus on single file types with unique characteristics and without a compound structure. The combined identification and reassembly approach tended to focus on the reassembled files having a valid structure or valid result after processing the data. In this research the identification and reassembly research have been conducted separately; this permitted the identification of single file fragments where the rest of the fragments of the file were no longer available.

Once the fragments have been identified and classified, reassembly methods can be used to attempt to reconstruct the files; the reconstruction enables a larger amount of information to be extracted. One task which requires fragment reassembly is completing a jigsaw [Pomeranz, 2011]; the methodologies employed for the completion of a jigsaw can be applied to other fragment reassembly problems. For example, fragment reassembly has been a large area of research in subjects such as archaeology where it is often necessary to reconstruct artefacts [Willis, 2008].

Fragment reassembly methods in related work have focused on solving path problems such as using the Hamiltonian, and K-Vertex disjoint path algorithms [Pal, 2006]; the K-Vertex disjoint path problem uses each cluster in unallocated space as a vertex. Pre-processing by first classifying the fragments and then only using related classifications in the graph would reduce the processing time. The use of context based statistical models for reassembly was explored by Shanmugasundaram [2003]; the method relies on in depth knowledge of the structure and syntax of the file type in order to build appropriate models to assist with the reassembly process.

For this research it has been shown that reassembly approaches use information about the expected structure of the object; the structural information provides guidance for selecting the next component at each point of the reassembly and validation of the reassembled object. This section has also established the adaptability of jigsaw reassembly methodology to other reassembly problems.

9.3 Methodology

As described in **Chapter 8**, the reassembly of a file is similar to the reassembly of a jigsaw; this research will apply jigsaw reassembly methodology to the

creation of a file fragment reassembly method. The general reassembly approach is documented in **Section 9.3.5**. Throughout this thesis the importance of understanding the structure and valid syntax of a file type have been highlighted. The information about the file type can be used to validate the reassembly of the individual files and assist with customising the reassembly process for each file type.

In order to complete this research a general approach to file fragment reassembly will be used; this approach will be customised by each individual file type to enable the selection and validation of file fragments at each point in the reassembly of a file. This section describes the general method for file fragment reassembly and discusses the considerations for customisation. A description of the methods for training and testing, and the data sets used for evaluating the reassembly methods are also provided.

9.3.1 Potential clashes and missing fragments

For each file type reassembly there are two important individual considerations: dealing with potential clashes for a reassembly point, and dealing with missing fragments. If no suitable fragment can be found for a point in the reassembly this research will place an empty fragment in its place; in order to ensure an analyst may easily identify the substitute fragment it will contain the ASCII phrase “SUBSTITUTE” repeated through the fragment. As shown in *figure 9.1*, after the reassembly process has gone through each point of the file it tries to resolve any points where a clash has occurred. A clash occurs where multiple fragments appear to fulfil the requirements for reassembly at a point in the file; once the remainder of the file has been reassembled it may be possible to identify which of the clashing fragments belongs to the file. The method for determining which clashing fragment to select is individual to each file type; the method requires an understanding of the characteristics of each file type to ascertain which fragment is the most likely to belong to the file.

9.3.2 Logging

In order to maintain a relationship with the original data it is necessary to produce comprehensive logs of any actions taken by the reassembly method. For this research 2 logs will be created: an overview log of the reassembly approach, and an individual logs for each reassembled file. The overview log will provide information on the number of reassembled files, and the fragments each file contains. The individual logs provide comprehensive information on each reassembled file; it states the original location of each fragment and the reasoning for selecting each fragment. The log also identifies any potential clashes for a point in the reassembled file and any missing fragments.

9.3.3 Data sets

In the reviewed previous research reassembly methods used a pre-selected data set; this research will use three data sets for the reassembly of thumbnail cache file fragments. The fragments for use with the reassembly method are taken from the file corpus described in **Chapter 7**. Each of the reassembly data sets is described below:

Reassembly Set A:

This set consists of all the complete thumbnail cache files from Data Sets 1 and 2 described in **Chapter 7**. This set only contains thumbnail cache file fragments and each file is complete.

Reassembly Set B:

This set consists of all the file fragments identified as **H1 – H5** by the hybrid approach created in **Chapter 8** from **Data Set 2**. The thumbnail

caches in the file may not be complete, as not all the known thumbnail cache file fragments were identified.

Reassembly Set C:

This set consists of all the file fragments identified as **H1 – H5** by the hybrid approach created in **Chapter 8** from **Data set 3**. The data set used for this research was a collection of unallocated space; given the volume of unallocated space it was not feasible to ascertain the type of each fragment and any possible relationships prior to conducting the research.

9.3.4 Training and testing

In order to develop the reassembly approach a selection of fragments are required to identify characteristics which can be used for the reassembly of files. Therefore half of each data set was available for examination of their individual characteristics to assist with the development of reassembly methods; the characteristics of individual fragments and fragments put together were examined. Both valid and invalid reassembled fragment sequences were examined to determine information which could be used to identify valid reassembly sequences. After the reassembly methods were complete, they were tested on remaining data, for which the characteristics had not been analysed.

9.3.5 A general reassembly approach

The classification of file fragments listed in **Chapter 7** separated the fragments into different classifications which would assist with their identification; these are listed below for completeness.

H1: *Thumbcache_idx.db* file

H2: Image thumbcache file and contains metadata

H3: Image thumbcache file and only contains JPEG image data

H4: Linux PNG visual thumbnail and contains metadata

H5: PNG visual thumbnail fragment and only contains image data

H6: Not part of a Windows or Linux thumbnail cache

The classifications break down the components of the thumbnail cache files to assist with their identification and classify them to assist with reassembly. In order to construct appropriate reassembly methods it is also necessary to consider the characteristics of each part of a file; after the characteristics have been identified it will be possible to search for the fragments in their identification classifications based upon this information. In general, for reassembly there are four categories of file fragment; these are described below:

Category 1: This file fragment contains the entire file; it begins with the file type header signature

Category 2: This file fragment contains the file type header signature and is the first fragment in a larger file

Category 3: This file fragment is in the middle of a larger file

Category 4: This file fragment is the last fragment of a file; therefore the data relevant to the file type may not fill the entire fragment

When reconstructing a jigsaw, generally the corners and outside edges are identified and reassembled first, as these have an easily identifiable characteristic; generally these pieces are easy to identify as they contain at least one straight edge (**Categories 1 and 2**). For a file type it is possible to identify the header based upon the identified structure and syntax. If the fragment falls into **Category 1**, then reassembly is complete; otherwise it is then possible to reassemble the internals of the jigsaw using the edges as a method of identifying and validating the remaining reassembly. The internal file fragments (**Category 3**) can then be identified in sequence and validated against the selected header; after all the internal fragments have been identified the final fragment can be added to the file (**Category 4**).

Having considered the reassembly methods used for jigsaws a model for the reassembly of file fragments was created. The general flow chart for the reassembly model is described in *figure 9.1*; this figure shows the process for reassembling multiple files, given a set of file fragments identified using the methodology documented in the previous chapter. The methodology can be used to construct a program to automate the process. Whilst the generalised method shows the approach for reassembly, individual file characteristics assist with selecting and validating fragments in each of the categories. Therefore each file type requires an individual approach to the selection and validation of fragments; these individual methods are discussed in later sections [**Sections 9.4-7**].

9.4 Thumbcache_idx

In **Chapter 6** the Windows 7 thumbnail cache structure was described; the index for the Windows 7 thumbnail cache is stored in a file named *thumbcache_idx*; the index file fragments were identification type **H1**. Using the hybrid fragment identification approach described in **Chapter 8**, the **H1** classification was identified 100% with no false positives in any data set.

Therefore this research assumes that false positives in the **H1** sub-directory are likely to be minimal; the research assumes that all fragments present in this sub-directory belong to the *thumbcache_idx* file type.

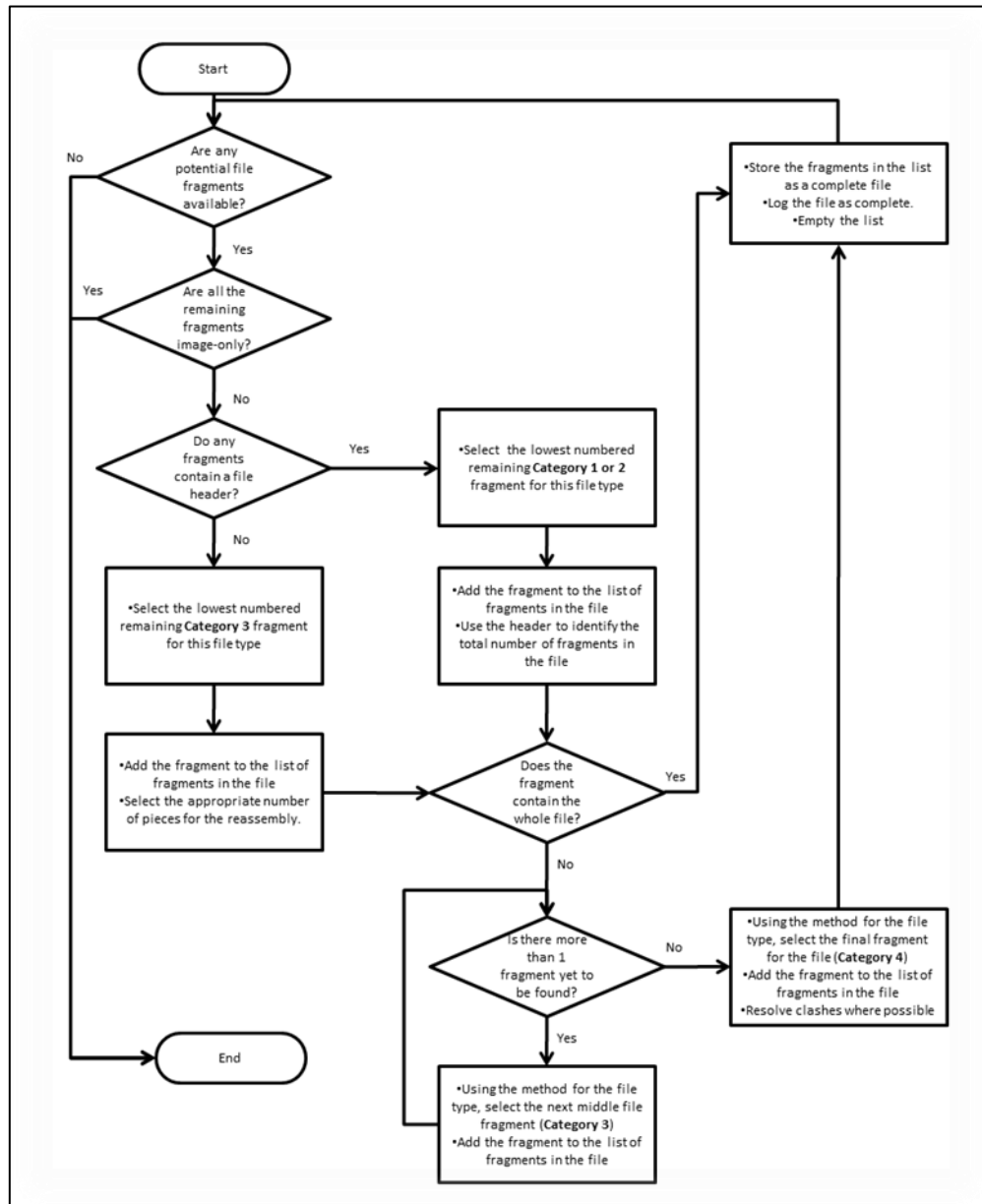


Figure 9-1: Flow Chart for File Fragment Reassembly

9.4.1 Thumbcache_idx file fragment categories

The repeating and fixed size record structure of the *thumbcache_idx.db* file can assist with reassembling the various fragments of the file. The index file stores records in a hash table whose size is always a multiple of 101; each file fragment of 4096 bytes can hold up to 127 complete records as shown in *figure 9.2*. The figure shows there are four different categories of this file fragment type. Below each category is described:

IDX Category_1: Fragment with 101 records and file header

In this case the single fragment contains the entire 101 record index and therefore does not require further reassembly. The first check to identify these fragments is to check the file header identifier “IMMM”; this is followed by checking the header structure is a valid *thumbcache_idx* structure as identified in **Appendix A**. Within the header the number of records should be set to 101, if the number is larger, then the fragment belongs to **Category_2**. As a 4096 byte fragment can contain up to 127 complete records a further check is made to ensure that the fragment is empty after the 101 expected records; therefore the last 840 bytes of the fragment is checked to ensure they are 0x00. If a fragment meets the criteria for this category it is logged as a reassembled file. As the entire 101 record file has been recovered the file is complete and the method moves on to the next fragment for identification.

IDX Category_2: Fragment with 127 complete records and file header

This fragment is the beginning of a larger index file which is stored over multiple fragments. The checks performed for this category are the same as in **category_1**; the difference between the two categories is the total number of records stored in the complete file. The file header contains the size of the index hash table and it is therefore possible to work out the total number of file fragments storing the index file; each fragment can hold up to 127 complete

records and 2 partial records. The number of allocated records in the hash table is also given in the file header this can assist in reassembly; in a situation where two fragments could fit into the same position of an index file, calculating the total number of records in the file could rule out a potential matching record.

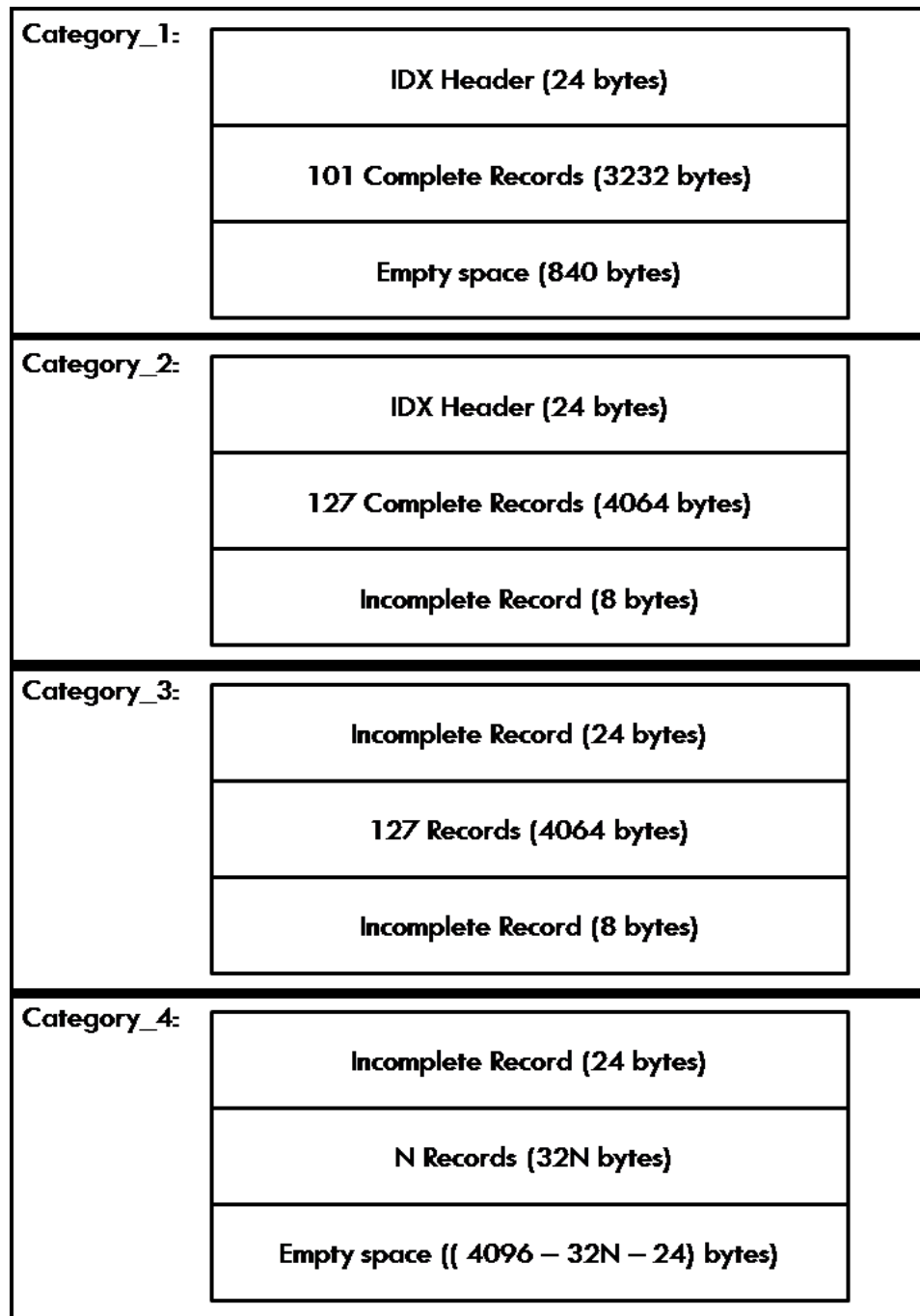


Figure 9-2: Thumbcache_IDX Classifications

IDX Category_3: Fragment with 127 complete records and no file header

In this case the fragment begins with 24 bytes of a record and finishes with the first 8 bytes of a record; in between these two partial records are 127 complete records which can be used to identify the potential size of the hash table that the fragment relates to. The complete records can also be used to identify the position of the fragment within the hash table by working out the relative position of the fragment for each valid hash table size. In order to establish a valid hash table size multiple records can be used to corroborate the validity of using the possible size in the formula for calculating relative positions; the formula for validating the thumbcache ID described in **Chapter 6** can be used to find a list of table sizes which would be valid for the fragment. This can be achieved by finding a common table size which produces valid sequential record numbers for the fragment.

IDX Category_4: Fragment with less than 127 complete records and no file header

In this case the records which are in use will not reach the end of the fragment and whilst it is likely there will be at least one in use record present the fragment may be completely empty. Therefore any records present will be used in the same way as in **Category_3** to determine the potential hash table size. Once this fragment has been added to the file reassembly chain, it is possible to calculate the number of in use records and check this matches what was expected.

9.4.2 Thumbcache_idx file fragment reassembly

In order to reassemble the index file fragments each fragment in the **H1** sub-directory is evaluated; the fragments are separated into those which start with the index file signature and those which do not. In order to minimise the

requirement for generating table sizes the possible table sizes for fragments which do not contain a header are calculated at the beginning of the reassembly process and stored in a CSV file. This file stores the valid table sizes for each fragment, and relative position in the file. For this research 100 possible table sizes are evaluated against each fragment.

The CSV file can be accessed during file reassembly to identify fragments which are valid for the current table size; the fragments can then be added according to their relative offset in the file. If multiple fragments are valid for a point in the file then they are stored as a clash, until the remainder of the file has been reassembled. After all the points of the file have been considered for reassembly the clash points are evaluated; using the number of records expected in the file it may be possible to accept or reject potential fragments. The system also looks at the number the fragment was assigned and gives priority to fragment numbers closest to the preceding fragment in the current file.

In order to reassemble the index files the characteristics of the four fragment types are used to reconstruct each index file; the method begins by identifying all fragments which contain header information and attempting to reassemble each file starting with the largest file size. Once the reassembly of all files with headers has been completed any remaining fragments are reassembled. Reassembly takes into account the possible sizes and relative positions. If multiple fragments are valid for a point in the file the physical location of each fragment on the disk is also considered; priority for reassembly is given to fragments located close to each other on the disk when a clash occurs. An example of an index reassembly is given in *figure 9.3*.

In order to evaluate the performance of this algorithm the asymptotic computational complexity of this method is evaluated using $O(n)$ notation; this establishes the worst case performance of the algorithm for a given input size,

n. To calculate the asymptotic computational complexity of this reassembly method an analysis of the implementation was performed; each step in the implementation was evaluated to assess whether it had a constant running time, or if the running time was dependant on the number of fragments being evaluated. For this method, $f(n)$ the number of computational steps was calculated to be:

$$f(n) = 122n^1 + 58c*n^0$$

where:

n represents the number of fragments being evaluated by the method

c represents a constant processing time

When $f(n)$ is converted into a standardised polynomial with the constants removed the function becomes:

$$f(n) = 61n^1 + 29$$

As the number of fragments being analysed by the method tends towards infinity, the constant value of 29 becomes insignificant compared to the value n^1 ; to find the computational complexity it is necessary to identify the upper bound of the algorithm. Therefore in this case as n increases the value of n^1 has the greatest effect on the running time of the algorithm; to calculate the computational complexity the constant value of 61 is removed leaving n. Therefore the upper limit running time of the reassembly algorithm for thumbcache_idx fragments given a set of n fragments is $O(n)$.

The method makes use of the properties of the hash table to minimise clashes and evaluations. The running time has a linear growth, which means that the growth is in proportion to the number of fragments the method has to evaluate. The identification method used for the thumbcache_idx fragments achieved a

100% success rate, with 0% false positives; therefore the input is optimised and the running time will be relative to the number of index fragments found.

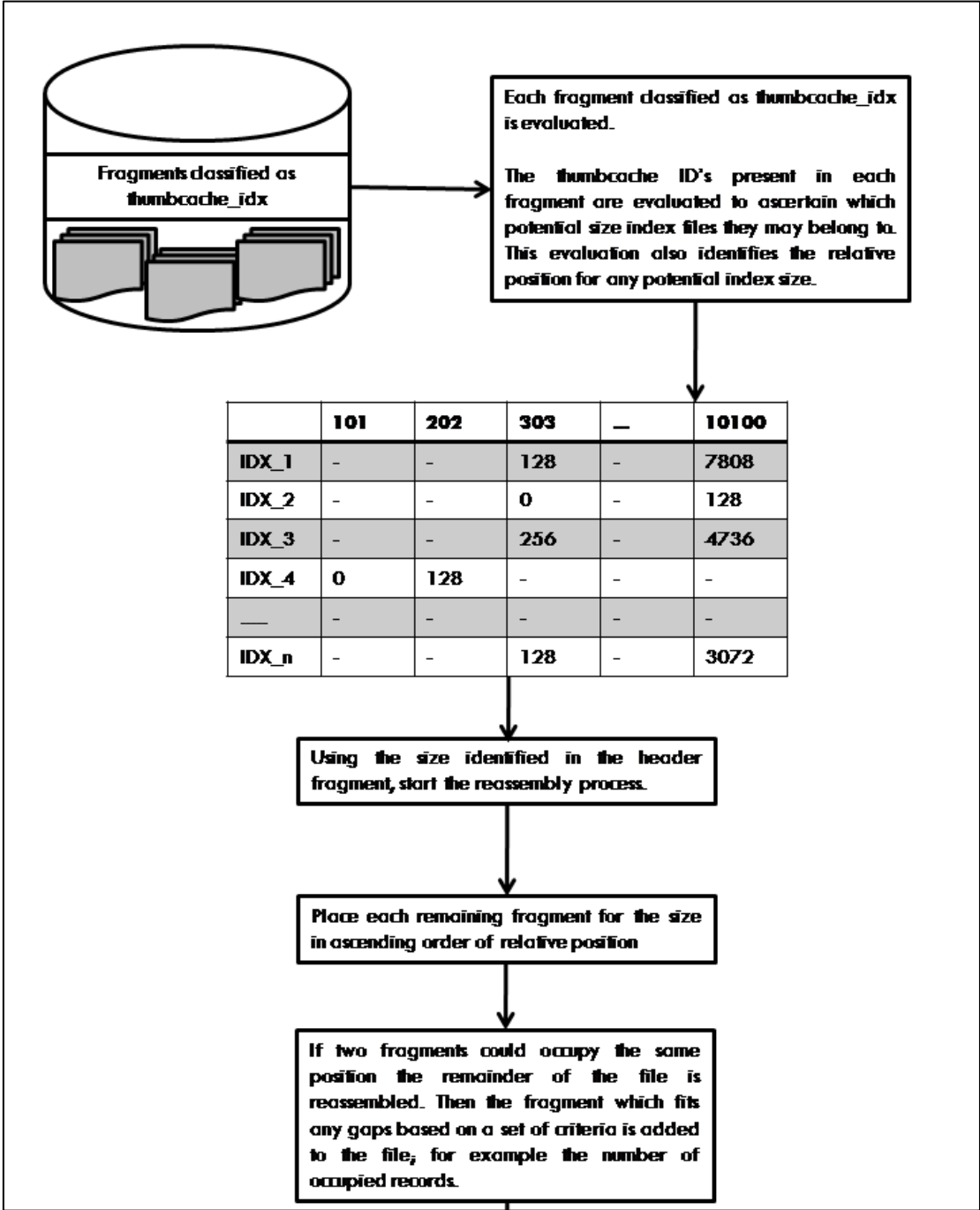


Figure 9-3: Thumbcache_IDX reassembly

9.5 Thumbcache_32, 96, 1024

In **Chapter 6** the Windows 7 thumbnail cache structure was described; the subrecords for the Windows 7 thumbnail cache are stored in files named *thumbcache_32*, *thumbcache_96*, *thumbcache_256*, *thumbcache_1024*. The 32, 96, and 1024 files store only standard subrecords; in **Chapter 7** the fragments which comprise these files were identified as classifications **H2** and **H3**. Using the hybrid fragment identification approach described in **Chapters 8**, the **H2** was identified 100% with no false positives in any data set. The **H3** set had a substantial amount of false positives. Therefore this research assumes that false positives in the **H2** sub-directory are likely to be minimal.

9.5.1 Thumbcache_32, 96, 1024 file fragment categories

The files which hold standard subrecords contain data in variable sized chunks; each file starts with a file header and then a series of variable sized subrecord chunks. Each subrecord chunk is made of two components: a header, and a visual thumbnail stored in a standard image format. Due to the variable sized nature of the chunks the structures can begin at any point in a fragment. As described in **Section 9.3** there are four types of file fragment which need to be considered for reassembly, these are described below:

Store Category_1: Fragment contains whole file

If a thumbcache file which only stores standard subrecords consisted of a single fragment it is likely to contain the file header with no subrecords; in this case the file would be resident in the MFT record. In the unlikely event that a single fragment whole file thumbcache holds a visual thumbnail, the header contains the size of the file and therefore the file could be identified as complete. If the fragment contains the file signature, valid structure and a file size less than or equal to a fragment size it is considered a single fragment thumbcache file; it is

then stored and marked as a complete file in the logs. Fragments in this category would be stored in the **H2** directory.

Store Category_2: Fragment is the start of the file

In this case the file would not be resident in the MFT and would be the start of a larger file; therefore it should begin with the file signature “CMMM” and the valid header structure identified in **Chapter 6**. The file header contains the total size of the file; when divided by the fragment size the number of fragments in the file can be calculated. After the file header the remainder of the file stores subrecord chunks; each chunk has a header which contains the total size of the chunk. Fragments in this category would be stored in the **H2** directory.

Store Category_3: Fragment is in the middle of the file

As described in the previous category, these files store information in chunks; each chunk has a header and a visual thumbnail. Information in the middle of the file can therefore either contain some information from a subrecord header, or consist entirely of visual thumbnail data.

If the fragment contains metadata it will have been classified as **H2**, these fragments at least part of a structurally valid subrecord header, or a visual thumbnail file signature. These fragments represent the start and the end of chunks; they can provide information on the size of the chunks which can be used to ascertain the number of fragments in the subrecord.

If the fragment contains image-only data it will have been classified as **H3**; they are the second part of a subrecord chunk. The visual thumbnail may span over several fragments, therefore in a file there may be multiple image-only fragments next to each other.

Store Category_4: Fragment is at the end of the file

The thumbnail cache files pre-allocate space, which they then fill with subrecords; once the file is full a new larger file is created, which can then be populated with the existing data giving the thumbnail cache room to continue growing. The last subrecord marker in the file contains an empty record; the empty record continues to the end of the file. The empty subrecord may contain non-thumbnail cache data in its data section; therefore there may be fragments which are part of the thumbnail cache file but do not contain thumbnail cache file data.

As it is not possible to know the contents, or structure of the last subrecord's data section, this research focuses on the reassembly of the files until the last subrecord is identified; the research does not aim to reassemble the data section of the last subrecord. The last subrecord header is identifiable: it begins with the header "CMMM", four bytes indicating the size of the subrecord, 32 bytes of 0x00, followed by the thumbnail cache ID value. The 32 empty bytes show this is the last subrecord and therefore all complete subrecords are above this point in the file. By identifying the fragment, where the last "CMMM" occurs at the expected relative offset given the size of the preceding chunk, the reassembly has reached the end of the file.

9.5.2 Thumbcache_32, 96, 1024 file fragment reassembly

Both category 1 and 2 represent the first fragment in a file; both categories represent a fragment which starts with the file signature "CMMM". As the file signature is also used as a subrecord signature it is possible that if a subrecord started on a fragment boundary it may be confused for the start of a file. To ensure a valid **category 1 or 2** fragment is identified the remainder of the file header is validated against the expected structure described in **Chapter 6**.

Once a start of file fragment has been identified the size of the file is extracted from the header; if the size is less than 4096 bytes the file is considered reassembled and logged. If the file is larger than 4096 then the number of fragments containing the file is calculated; this provides the number of fragments in the chain.

The thumbnail cache files which contain visual thumbnail records have a set structure [Appendix A]; however given the variable size of elements of the structures the information could occur at any point in the fragment. This means that two connecting file fragments could join at any point. Since each file consists of a collection of subrecords the information that bridges two fragments will either belong to subrecord metadata or the image data.

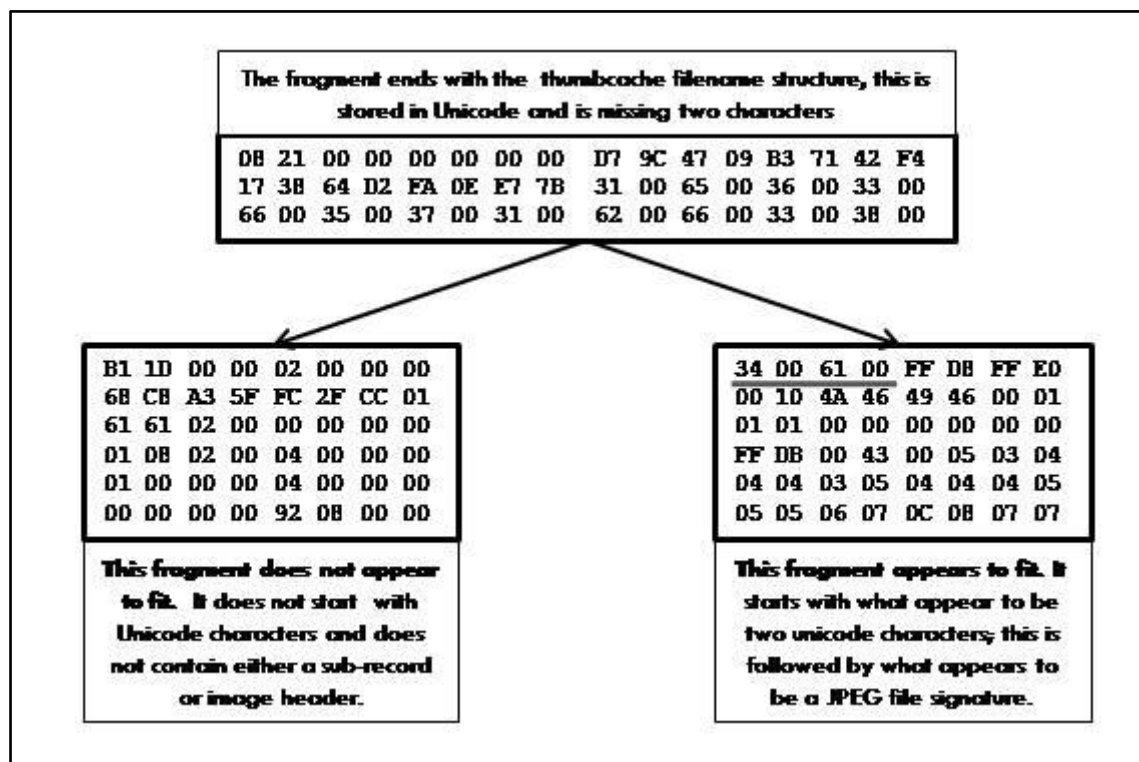


Figure 9-4: Reassembling fragments with a metadata join

If the join is in the middle of a metadata section such as in *figure 9.4*, the structural template can be applied to identify a fragment with a valid structure and syntax to follow the fragment. The header section of image records contain fixed metadata structure; this allows the type of structure to be identified from the first fragment and the template to be applied until the end of the fragment. Each remaining fragment can then be assessed to see if it is valid based on the remainder of the structural template being applied. In the example shown in *figure 9.4* the template finishes part way through a variable and this is matched to a fragment which fits the remainder of the structure.

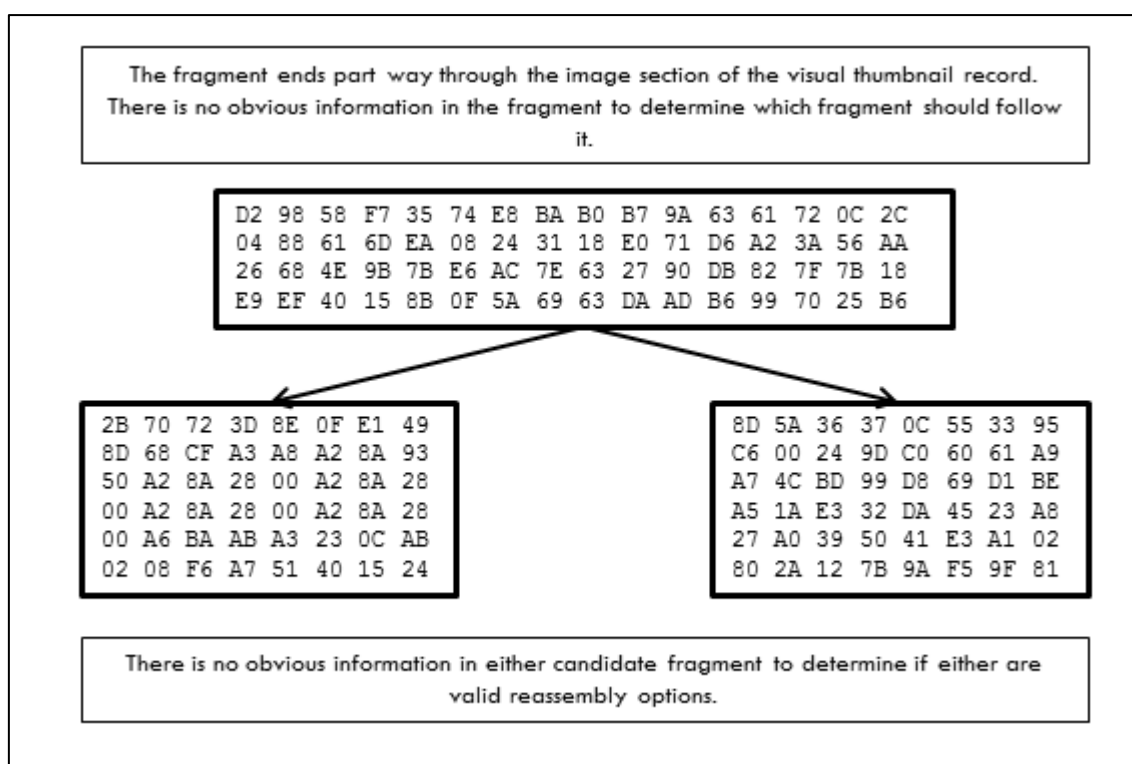


Figure 9-5: Reassembling an image fragment

If the fragments join in the middle of image data like the example shown in *figure 9.5* then image fragment reassembly methods would need to be employed to identify suitable candidates. Firstly if the fragment which contained the size of the image data was available it would be possible to identify the total

number of fragments which contain only image data. The last fragment of the image can be identified by working out the position of the last image data byte relative to the start of the fragment; this is shown in *figure 9.6*. Secondly assuming the visual thumbnail was written and any subsequent changes to the visual thumbnail resulted in a completely new visual thumbnail being generated which replaces the subrecord it can be assumed the chances of fragmentation are small; therefore the first step in image fragment reassembly is to identify if any of the fragments available would have been located next to the reassembled data on the disk. If a candidate fragment is next to the last fragment in the reassembled sequence it is likely to be the next fragment; therefore by first attempting to reassemble potential chains on the disk it is possible to minimise reassembly resources.

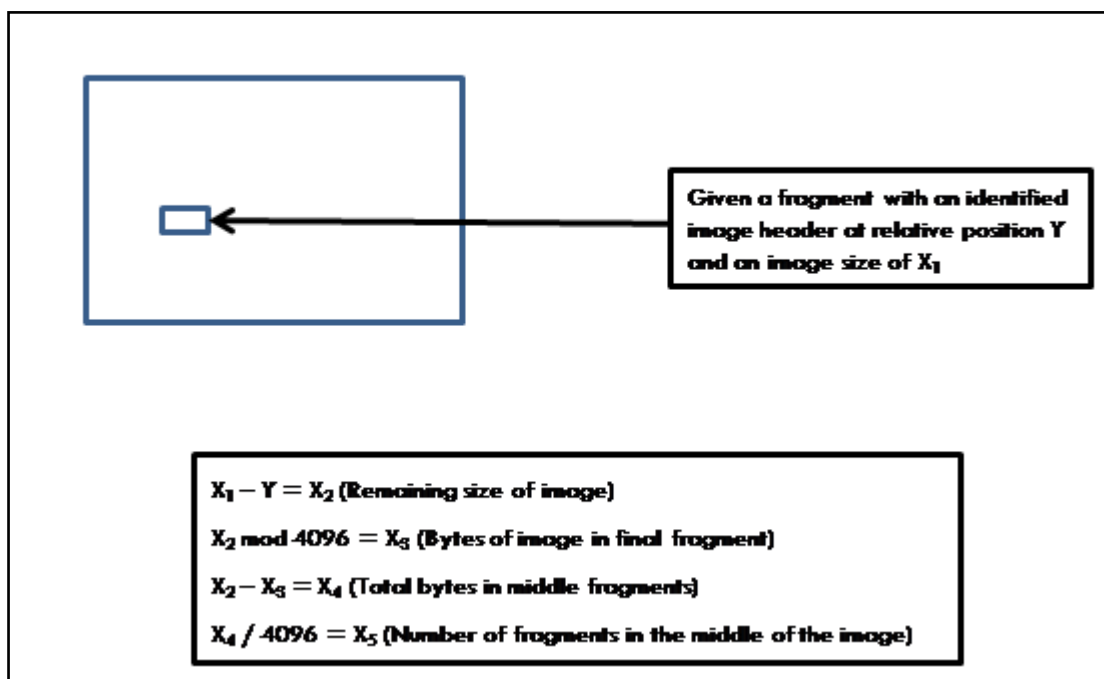


Figure 9-6: Calculations for image reassembly

If trying the fragments located close to the existing sequence did not provide a valid file sequence it would be necessary to try other techniques. An option for reassembling the image fragments for a known size of image data is to try every

possible combination of fragments and attempt to interpret the data as an image in the appropriate format. An example of this is shown in *figure 9.7*; however since image fragments may not belong to the thumbnail cache there is likely to be a significant number of fragments making this approach resource intensive. The numbers of possible permutations which require evaluation for potential visual thumbnail data sizes can be calculated using the formula:

$$\frac{n!}{(n-x)!} = c$$

Where

n is the total number of fragments being considered during reassembly

x is the expected number of fragments in the reassembled file

c is the number of possible permutations of the file fragments

For this research after classification of the file fragments there is still a substantial difference between the amount of fragments classified as a file type and the average size of an individual thumbnail cache file. The high number of false positive fragments which belong to other image files results in a significant number of permutations. JPEG reassembly approaches could be adopted such as checks to identify changes in pixel colour between two connecting fragments; if two fragments are connected the change in pixel information at the join of the fragments is likely to be small. It is feasible that multiple visual thumbnails may exist which are the same and may relate to multiple files; in unallocated space there may be limited information to form further relationships.

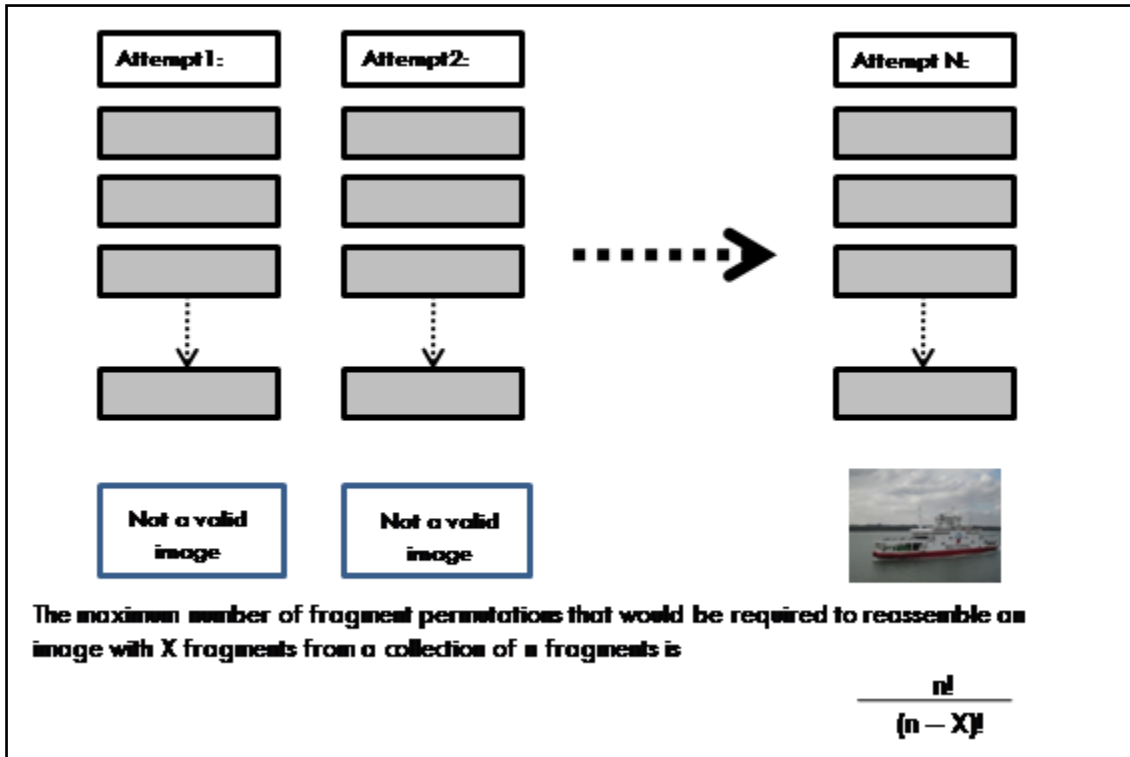


Figure 9-7: Reassembling an image by trying every permutation

The production of all syntactically valid image fragment sequences provides an analyst with the opportunity to decide how to proceed. Therefore unless a valid image fragment sequence directly follows a subrecord header fragment the method will produce and store the visual thumbnails for any valid JPEG sequence which can be formed from the remaining fragments. In order to validate the images each set of fragments was passed to a python program which ran an Image module and attempted to verify and open the images. If the entire image appeared to be a valid sequence the image was exported and put in the same directory as the rest of the reassembled file. If multiple fragment sequences were valid for a subrecord the image only sections of the subrecord were left blank and the valid image sequences were stored in a sub-directory named with the reassembled files name and relative offset of the subrecord. The log for reassembling images shows the original name of each image only fragment and its position in the completed sequence.

On examination of the training set 72% of the subrecords were contiguous; therefore it appears likely that once a subrecord header is identified the remainder of the record would contiguously follow it. In the remaining 28% of subrecords were fragmented a maximum of twice; the fragments were spread across the storage media. Without further knowledge of the storage structure it would not be possible to form a relationship between the fragments. In these circumstances producing every valid combination provided a maximum of 42 results for each visual thumbnail. Out of the images produced a maximum of 11 resulted in full visual thumbnails which could be viewed using a standard image viewer. Therefore an analyst could manually go through the resulting images along with the logs and decide what evidence if any to produce.

To calculate the asymptotic computational complexity of this research an analysis of the implementation was performed using the same approach as **Section 9.4.2**. For this method $f(n)$ the number of computational steps was calculated to be:

$$f(n) = 6n! + 34n^2 + 116n^1 + 75c*n^0$$

where:

n represents the number of fragments being evaluated by the method

c represents a constant processing time

When $f(n)$ is converted into a standardised polynomial with the constants removed the function becomes:

$$f(n) = 6n! + 34n^2 + 116n^1 + 75$$

As the number of fragments being analysed by the method tends towards infinity, the constant value of 75 becomes insignificant compared to the value $116n^1$; however both $116n^1$ and $34n^2$ become insignificant when compared with $6n!$. Therefore in this case as n increases the value of $6n!$ has the greatest effect on the running time of the algorithm; to calculate the computational complexity the constant value of 6 is removed leaving $n!$. Therefore the upper limit running time of this reassembly algorithm given a set of n fragments is $O(n!)$.

This algorithm will take a significant time to process large data sets, therefore pre-processing the data to minimise the fragments being evaluated will assist with reducing the potential running time. The running time increases quickly with large data sets, this is due to the substantial number of comparisons that are required to provide all valid image reassemblies. In Chapter 8 the identification method had a relatively high number of false positive image-only file fragments; therefore significant computation is required to evaluate the reassembly of the large amount of image-only fragments, the running time of this method could be reduced by improving the identification of **H3** fragments.

9.6 Thumbcache_256

As described in **Chapter 6**, the *thumbcache_256.db* file can contain subrecords which contain Unicode strings instead of visual thumbnail data; the string is contained within the metadata section of a subrecord. The reassembly of the *thumbcache_256.db* file is very similar to the reassembly of the other Windows 7 image files; the *thumbcache_256.db* utilises the same reassembly methods as those in **Section 9.5** as well as an extra method to take into account the records which do not contain visual thumbnails

9.6.1 Thumbcache_256 file fragment categories

As described in **Chapter 6**, this file type also stores subrecords for the thumbnail cache; this file can store both standard and non-standard subrecords. Therefore it is possible that if a subrecord header is found near the end of the fragment, the proceeding fragment may not contain visual thumbnail data as expected; instead after the subrecord header there may be another subrecord header. The fragments for this file type are also stored in **H2** and **H3**.

This file type uses the same categories and methods as those described for the thumbcache files in the previous section. However **Category_3** may contain fragments which have non-standard records. These records typically contain a larger Unicode string at the end of the header; the subrecord does not contain an image so the subrecord header is followed by a new subrecord header.

9.6.2 Thumbcache_256 file fragment reassembly

These files are reassembled using the same characteristics as **Section 9.5**; a check was performed to ensure a subrecord header was followed by a valid image file signature. For this file type the subrecord header may also be followed by a subrecord header. Each metadata section contains a field for the total size of the subrecord which assists in identifying the potential end of the Unicode string. *Figure 9.8* shows a fragment join in the middle of a Unicode string; the fragment to be joined should start with Unicode data whose size completes the subrecord. This should be followed by either a subrecord or if it is the last subrecord in the file it will be followed by file slack information.

Fragments that contain solely the Unicode data and file slack can be difficult to both identify and reassemble as there is very little structural data available; these fragments may not be identified by the identification method which was implemented in **Chapter 7** and therefore there is a case for performing further

file fragment identification on storage media after the reassembly phase. This would be specifically for missing fragments which because of their characteristics may not have been identified in the first phase.

9.7 Linux Thumbnail Records

The open desktop specification used for the thumbnail cache in Ubuntu and Kubuntu; this structure was described in **Chapter 5**. The thumbnail subrecords are stored in standard PNG files. The PNG structure uses chunks which contain a size and a CRC value for validation. The Linux thumbnail cache fragments were identified as **H4** and **H5**. The **H4** category represented a Linux thumbnail cache fragment which contained metadata; **H5** represented an image-only PNG thumbnail cache file fragment.

30 00 41 00 41 00 30 00	30 00 32 00 46 00 39 00
35 00 34 00 45 00 7D 00	<u>43 4D 4D 4D</u> 64 AD 00 00
21 79 7C 71 0B 1A CB 3F	<u>20 00 00 00</u> 00 00 00 00
14 AD 00 00 00 00 00 00	34 7E D4 B5 EE C0 0A A3
FF B4 45 C5 2A AD B6 9A	<u>33 00</u> 66 00 63 00 62 00

<u>43 4D 4D 4D</u>	- Start of Record
<u>20 00 00 00</u>	- Size of unicode filename
<u>33 00</u>	- Start of unicode section

8 bytes of unicode at the end of the fragment

The next fragment should start with 24 bytes of unicode followed by an image or new record signature

31 00 61 00 30 00 38 00	37 00 31 00 37 00 63 00
37 00 39 00 32 00 31 00	89 50 4E 47 00 0A 1A 0A
00 00 00 00 49 4B 44 52	00 00 01 00 00 00 01 00
0B 06 00 00 00 5C 72 AB	66 00 00 00 01 73 52 47
42 00 AE CE 1C E9 00 00	00 04 67 41 4D 41 00 0055g

Figure 9-8: Fragment reassembly using file structure

9.7.1 Linux thumbnail file fragment categories

Linux thumbnails are stored over several fragments; in **Chapter 7** generally half the fragments of each individual file contain metadata. In **Section 9.3** it was determined that this research uses four categories of file fragment for reassembly. The categories are described below:

Linux Category_1:

This category represents a fragment which is a self-contained file. Given the structure of the PNG format the chance of a thumbnail cache image in the format which is less than 4096 bytes is small. If a fragment fits into this category the file size would be less than or equal to the fragment size; the fragment would also contain complete chunks which had valid CRC values.

Linux Category_2:

This category represents the first fragment in a larger PNG Linux thumbnail cache file; the fragment starts with a PNG header and may contain thumbnail cache specific metadata chunks. The PNG header contains the expected size of the image, and each chunk also contains an expected size. It is likely the fragment will end part way through a chunk; in this case fragments can be found which match with the expected size of the fragment.

Linux Category_3:

This category represents fragments which are in the middle of a file; these fragments may either contain metadata or be image-only. On analysis of the training file it was found that the only metadata present in the middle fragments of PNG thumbnail images was the IDAT chunk identifier. This identifier would be preceded by the CRC value for the previous chunk and the expected size of the IDAT chunk.

Linux Category_4:

The last fragment in a Linux PNG thumbnail contains the IEND chunk. The end of the IEND chunk is the last byte of the file; the file size should equal the file size documented in the header of the file.

9.7.2 Linux thumbnail file fragment reassembly

As identified in **Chapter 5**, subrecords stored within the Gnome and KDE thumbnail caches are stored in standard PNG format. The PNG format uses data chunks which contain a data size, type and crc value; these provide useful reassembly information by indicating both the expected structure of a fragment and validation information. The hexadecimal output below shows a breakdown of a thumbnail cache specific chunk found during the research conducted in **Chapter 5**.

```
00000000  00 00 00 1874 45 58 7454 68 75 6D 62 3A 3A 49  ....tEXtThumb::I
00000010  6D 61 67 65 3A 3A 48 65  69 67 68 74 00 33 35 38  mage::Height.358
00000020  0F B9 03 29                .1.)
```

Each chunk begins with a data size value; this indicates the complete size of the chunk and therefore can be used to assist in identifying the number of bytes in the chunk and specifically identifying the final byte of the chunk. This is followed by an identifier, for the example above the identifier is 'tEXt', indicating the chunk holds a string of data. The identifier provides syntactical information to allow the chunk to be interpreted in an appropriate context. Finally each chunk ends with 4 bytes which contain a 32 bit CRC value for the identifier and data sections. The CRC value can also assist with validation, and is particularly useful during reassembly if a chunk spans two or more fragments as it provides a method for validation.

In the training set there was only a single image-only fragment between every fragment containing metadata. This assists with reassembly; the first fragment contains the expected size of the chunk. The size determines the position of the end of the chunk which can assist with finding the fragment containing the end of the chunk; the next chunk should start directly after the specified chunk size. The middle fragment of the chunk is from the **H5** fragments. Each fragment in the **H5** category can be put into the middle of the chunk; a CRC value can then be calculated. If the CRC value matches the one stored at the end of the chunk, the fragments in the chunk are accepted and added to the reassembly sequence. As the last fragment containing the chunk would also contain the start of the next chunk the process can be repeated until the end of the file is reached. The last fragment in the file would also contain the IEND chunk. If the validation of each chunk was successful, the final chunk IEND was successfully identified, and the size matches the expected size in the header the reassembly is considered successful.

9.8 Results

The table below shows the final results for the reassembly methods described in this chapter. As shown below the *thumbcache_idx.db* file had 100% valid reassembly, whereas the thumbnail caches containing image-only fragments had a substantial amount of clashes or incorrectly identified fragments. This is due to difficulties in selecting the correct image-only fragment.

Table 9-1: Results from reassembly of data set A

		Correctly Reassembled	Clashes / Incorrectly Identified Fragments			Total
			IDX	Metadata	Image-Only	
Thumbcache_idx	No.	6495	0	0	0	6495
	%	100.00	0.00	0.00	0.00	100.00
Thumbcache_32	No.	217067	0	0	21254	238321
	%	91.08	0.00	0.00	8.92	100.00
Thumbcache_96	No.	2477288	0	6463	495255	2979006
	%	83.16	0.00	0.22	16.62	100.00
Thumbcache_1024	No.	7944428	0	7971	964621	8917020
	%	89.09	0.00	0.09	10.82	100.00
Thumbcache_256	No.	2046002	0	4831	181409	2232242
	%	91.66	0.00	0.22	8.13	100.00
Linux PNG	No.	72817	0	0	280	73097
	%	99.62	0.00	0.00	0.38	100.00

Table 9-2: Results from reassembly of data set B

		Correctly Reassembled	Clashes / Incorrectly Identified Fragments			Total
			IDX	Metadata	Image-Only	
Thumbcache_idx	No.	3277	0	0	0	3277
	%	100.00	0.00	0.00	0.00	100.00
Thumbcache_32	No.	10001	0	0	1673	11674
	%	85.67	0.00	0.00	14.33	100.00
Thumbcache_96	No.	121896	0	325	27778	149999
	%	81.26	0.00	0.22	18.52	100.00
Thumbcache_1024	No.	353024	0	361	50816	404201
	%	87.34	0.00	0.09	12.57	100.00
Thumbcache_256	No.	94188	0	232	12628	107048
	%	87.99	0.00	0.22	11.80	100.00
Linux PNG	No.	20350	0	0	78	20428
	%	99.21	0.00	0.00	0.38	99.59

Table 9-3: Results from reassembly of data set C

	Fragments Reassembled
Thumbcache_idx	71
Thumbcache_32	178
Thumbcache_96	293
Thumbcache_1024	176
Thumbcache_256	531
Linux PNG	238755

9.9 Artefacts from fragments identified in unallocated space

This chapter has implemented reassembly methods for the thumbnail cache file fragments identified in **Chapters 7 and 8**. This section describes the artefacts identified from the reassembled fragments.

All the *thumbcache_idx.db* fragments were identified and successfully reassembled. The *thumbcache_idx.db* provides the thumbcache ID string, identifies the number of records stored in the thumbnail cache and the number of subrecords for each record. This information is useful for linking individual subrecords together and potentially for forming a relationship with the source file.

Thumbcache fragments which contained metadata were also identified and reassembled however not all visual thumbnails were successfully reassembled. The metadata sections provide information about records containing visual thumbnails; for a record only one of up to four possible visual thumbnails needs to be successfully reassembled for an analyst to potentially have evidence of an

activity. All the non-standard subrecords were recovered providing a variety of information about icons, network places and drive allocations.

The thumbnail caches recovered show that it is possible to recover fragments from previous thumbnail caches on the system. The caches may be from previously installed operating systems, deleted user accounts or from thumbnail caches deleted by the system or user to enhance system performance or privacy. Whilst the range of thumbnail caches is wide it is difficult to determine where the thumbnail cache was originally from. However using the live file set it is possible to determine earlier versions of the current thumbnail cache.

For the Linux thumbnail cache PNG visual thumbnails all identified fragments were successfully reassembled however they required a significant amount of processing time. The visual thumbnails provide the same range of information as those detailed in **Chapter 5**. The visual thumbnails which contain the URI assist in forming a relationship with individual source files; the visual thumbnails can also be used to form relationships with the system. As with the Windows thumbnail cache it is not possible to determine whether the thumbnails come from the current or a previous installation of an operating system. It is also difficult to determine the user account. Information such as who had permission to access a URI stored in a visual thumbnail can assist with forming further relationships between the information in unallocated space and the live file set.

9.10 Discussion

Each of the reassembly methods implemented during this research used knowledge of the structure and syntax of the file types; the general method provided a structure for the reassembly which was supplemented by providing information on the unique characteristics of each file type. Whilst other reassembly methods could have been selected the use of file characteristics

makes the method simple to explain and repeat; this would assist in corroboration and explanation for analysts using the software. An important feature of each implemented method are the logs which provide information on what reassembly has occurred and the reasoning the software used to reach this decision.

The reassembly of image-only fragments proved the most difficult; the large number of fragments retrieved for this classification by the identification methods meant it was resource intensive to iterate through the complete set of fragments during reassembly. On analysis of the reassembled files, the clashes occurred on almost every occasion where the subrecord was not contiguously stored on the disk. This may be due to the structural similarities between visual thumbnails which make it more likely a false positive reassembly of the subrecord would appear valid. It is also possible that the number of visual thumbnails which were the same had an impact on the number of false positives; in a thumbnail cache it is likely that some visual thumbnails will occur multiple times in the cache.

This research chose to present all valid image reassembly sequences created during this research; this was instead of using advanced image techniques. If the subrecord containing the visual thumbnail was fragmented on the disk and identified in unallocated space it would be difficult to determine which fragments belonged to the image. For visual thumbnails the difficulty lies in the number of visual thumbnails stored which would be the same. Every time the thumbnail cache updates its file size, artefacts can be found in unallocated space. It is difficult to relate individual artefacts together, especially when the image is common. Therefore the decision was taken to display every valid combination of sequences for a subrecord. This prevents the method from displaying a single sequence for a subrecord which may not be accurate; the method puts the emphasis on the analyst using the tool to review the results and verify any artefacts they feel are relevant. The method only displays valid sequences,

which substantially reduces the information the analyst has to review compared to viewing every possible combination.

The inbuilt validation used in the Linux thumbnail subrecords provided assistance with validating the reassembly. The use of CRC values provided a simple validation of proposed chunk sequences. For each image-only fragment required every **H5** value was tested; however by using the inbuilt file validation very few false positives were found. The false positives related to subrecords which were the same; therefore whilst the reassembly was valid the file fragments put together were not as the system would have originally viewed them. This produces significant problems for presenting the evidence in Court and reinforces the decision to display every valid combination in cases where a clash occurs; allowing the analyst to perform further investigation into the results before presenting his results.

The production of a general model based upon reassembling a jigsaw provided four categories of file type; the detailed knowledge of the format from the research conducted in the previous chapters could be applied to the categories. By focusing on reassembling the individual categories into a complete file it was possible to establish the characteristics for validation of the file fragment reassembly sequences. The results of this research show it is possible to successfully reassemble sections involving metadata accurately. By using characteristics from outside the fragment such as those from: the file system, file fragmentation rates, and general file structures can also assist in reassembly for image-only fragments. The additional characteristics provide further information to assist with selecting appropriate fragments.

9.11 Conclusion

This chapter has focused on the creation of reassembly methods for thumbnail cache file fragments identified in **Chapters 7 and 8**. A general reassembly approach was created based upon reassembling a jigsaw; the method was personalised for each file type. Each reassembly method focuses on specific characteristics of that file type to provide an accurate and repeatable approach. One of the most difficult reassembly problems involved identifying the correct image-only fragment; in order to correctly reassemble these fragments further information about the system was used such as fragmentation characteristics for the file type. This research has shown it is possible to both identify and reassemble thumbnail cache file fragments located in unallocated space; the next chapter discusses the evidential value of the reassembled fragments.

10 Establishing the evidential value of thumbnail cache file fragments identified in unallocated space.

10.1 Introduction

The aim of this research is to establish the evidential value of thumbnail cache file fragments identified in unallocated space. In order to achieve this it was necessary to establish a set of criteria which could be used to evaluate the evidential value of thumbnail cache evidence; the criteria are described in **Chapter 4**. Structural and behavioural studies were then conducted on popular operating system thumbnail caches in order to provide a baseline for evaluating the evidential value of the fragments identified and reassembled in the later stages of this research. This chapter uses the criteria for evaluating evidential value along with the baseline data to evaluate the evidential value of the fragments recovered from unallocated space throughout this research. The chapter begins by evaluating the evidential value of thumbnail cache file fragments against the previously defined criteria. This is followed by a discussion of the comparative evidential value of live and recovered thumbnail cache artefacts. Finally this chapter ends with a conclusion.

10.2 Replication, Corroboration and Clear Documentation

“Results are considered *replicable* when they are documented sufficiently that another analyst could conduct the experiment again and achieve the same results.” [Chapter 4.3.1]

“A result can be *corroborated* when the documentation is clear enough to allow the process to be repeated and therefore the results can be verified.”
[Chapter 4.3.1]

“Clear documentation is defined as something which another analyst could read, understand and use to repeat the described processes.” [Chapter 4.3.1]

The identification methods implemented were documented in **Chapter 7**, and the image fragment identification was further discussed in **Chapter 8**; an analyst could use the documentation to replicate the experiments. If necessary, an analyst could confirm the research results by replicating the experiments; however it may be difficult for a typical analyst to implement both the statistical and *neural network* identification methods described in **Chapter 7**. In **Section 2.2.2** the use of dual tool verification was discussed; if a second tool or an analyst working by hand could independently achieve the same result then this could be used for corroboration. Each implemented method provides a log which contains: the classification of each fragment, the original position of the fragment on the disk, and the reason for the classification [Section 7.9]. The log provides sufficient information about individual fragments to allow an analyst to corroborate the classification by hand or with a second tool; therefore the logs can be used to understand the process and corroborate the results.

The reassembly methods implemented in **Chapter 9** also provide logs which contain: the original location of the fragment, its position in the reassembled file, and the reason for its current reassembled position. This information can be used by an analyst to identify the fragments of a reassembled file by hand on the original disk and follow the information provided in the log to verify the reassembled file [Section 9.3]. The reassembly methods described in **Chapter 9** are based on structural and syntactical properties of the individual file types; an analyst could use the information provided to replicate the method and corroborate the results.

The thumbnail cache artefact extractor documented in **Section 9.10** uses the structures provided in **Appendix A** that were established through the experimentation performed in **Chapters 5 and 6**. The results of the extraction document the original offsets of the artefacts and show the relationships between artefacts. The results could be replicated and corroborated by using the same experimental procedure to ascertain the structure and behaviour of the thumbnail caches using the clear documentation provided in **Chapters 5 and 6**.

The documentation provided in both **Chapters 7-9** which detail the implemented methodologies and the log files utilised by each method [**Sections 7.9, 9.3**] provide sufficient information to enable an analyst to conduct their own experiments.

10.3 Justifiable Method, Understandable Consequences and Scientific Method

“An action is defined as *Justifiable* if it can be shown how the results were obtained and the effects on the original data are understood.” [**Chapter 4.3.1**]

“A process is defined as having *understandable consequences* when at each stage of the process the results and their relationship to the original data can be explained.” [**Chapter 4.3.1**]

“The process can be considered to follow the *scientific method* when a hypothesis is formed based on the problem to be solved and tested through a process of data observation and experimentation.” [**Chapter 4.3.1**]

As discussed in the previous section the log files produced at each stage of the process provide sufficient information to show how the results were obtained and their relationship to the original data. For the identification methods each log documents the reason a fragment was put in a particular classification [Sections 7.9, 8.2]; by documenting the reason for each classification it is possible to determine how the results were obtained. The logs also show the original location of a fragment which permits an analyst to find a fragment in the original data and analyse it in the context of the complete set of data.

For the reassembly methods a similar log is used [Section 9.3] to provide documentation for the reassembly process. The log provides a breakdown of the reassembled files and the reason for the position of each reassembled fragment; therefore it an analyst is shown how the results were obtained and how they relate to the original input data. The extraction of artefacts from the reassembled thumbnail caches is achieved by using the structures identified in Chapters 5 and 6 which are documented in Appendix A. The extraction software displays each extracted artefact with a breakdown of the structure that was applied to interpret the data, details of the original location, and links to other related artefacts [Section 9.10].

In order to extract thumbnail cache artefacts from unallocated space the method created for this research was broken down into stages. Firstly the method classifies fragments into types [Chapters 7 and 8]; this allows the removal of fragments which are unlikely to belong to thumbnail cache files. Once the fragments are classified the thumbnail cache files are reassembled from the identified fragments [Chapter 9]; at this stage all identified fragments which could not be reassembled are stored for potential hand analysis at a later stage [Section 9.10]. Thumbnail cache artefacts are then extracted from the reassembled files using the structures and syntaxes identified in Chapters 5

and 6. Each stage of the method has been shown to be justifiable earlier in this subsection by providing logs which document the effect on the original data therefore the process for the retrieval of thumbnail cache artefacts from unallocated space has understandable consequences.

The process has a scientific method as it was created based on a hypothesis [Section 1.4], based on the justification provided in Section 1.2. A problem definition and objectives were described in Section 3.1; the methodology to test the hypothesis using both observation of the data and experimentation was provided in Section 3.3. Each individual section also contains a detailed description of the specific methods used.

10.4 Typical Operation, Behaviour, Relationships and Maximising Corroboration

“In order to determine an artefact was created during the *typical operation* of a machine it is necessary to determine the system behaviour which led to an artefacts current state.” [Chapter 4.3.2]

“In order to establish the *behaviour* an artefact represents it is necessary to form relationships with other artefacts to identify potential system activity.”
[Chapter 4.3.2]

“A *relationship* between artefacts can be defined as an association between artefacts; the relationship may provide corroboration or supplementary information that enhances what is known about the system activity.”
[Chapter 4.3.2]

“Maximising corroboration can be defined as verifying and providing additional information the artefact with other alternative sources of information.” [Chapter 4.3.2]

Experiments were performed to identify the structure and behaviour of the operating system thumbnail caches [**Chapters 5 and 6**] to ascertain whether the artefacts occurred during the typical operation of the machine. The potential user behaviour which led to an artefact’s current state was documented in **Sections 5.6 and 6.5**; it was necessary to form relationships with other system artefacts [**Sections 5.7, 5.9, and 6.6**]. These relationships provided complementary and supplementary information relating to the thumbnail cache artefacts and assisted in narrowing down potential user behaviour.

In order to identify the structure and behaviour of the operating system thumbnail virtual drives were used to simulate potential user activity; these drives were then analysed using forensic techniques [**Sections 5.2, 6.2**]. The analysis method ensured that any changes between the baseline virtual machine and the experiment data were analysed; this allowed relationships to be formed between the thumbnail cache and system artefacts. **Chapters 5 and 6** maximised the corroboration of the data by evaluating the potential relationships with any altered system artefacts and the thumbnail cache; any identified relationships were then documented.

The thumbnail cache artefacts retrieved from unallocated space can potentially form relationships with the system artefacts identified in **Sections 5.7, 5.9, and 6.6** which are present in the live file set. However neither the thumbnail cache artefacts identified in unallocated space or the live thumbnail cache artefacts can currently maximise corroboration or form relationships with other system artefacts identified in unallocated space. The methodology used for the extraction of thumbnail cache artefacts in unallocated space could potentially be

adapted to extract artefact from other file types and therefore extract potentially maximise corroborating information.

10.5 Maximum Amount of Data

“The *maximum amount of data* retrieved is defined as the identification and reassembly of all thumbnail cache file fragments used during the research.”

[Chapter 4.3.2]

Fragment recovery is attempted to maximise the data available to an analyst; the information recovered may provide additional information relating to the investigation. In order to establish if the maximum amount of data was retrieved using this method each file fragment in the corpus contained a header which provided information about the actual classification of the file fragment and its original location on a disk [Section 7.3]. The use of a controlled corpus allowed the results at each stage of the process to be monitored and compared to the original data.

For the identification methods the results in **Section 7.10** show the identification of each fragment classification; the results identify the actual number of fragments in a classification and provide a breakdown of where the fragments were classified using the method. As can be seen from the results of method 5 100% of the file fragments in classifications **H1** and **H2** were correctly identified with no false positive identifications in these categories. Of the remaining classifications the lowest achieved accuracy was 66.84% for **H3**; this category had a substantial amount of false positives.

Table 8.2 shows the improved accuracy of image-only JPEG fragment recognition with an improved result of 75.5% of this category being correctly classified. The hybrid approach took the most successful method for each classification from **Chapter 7** and continued to develop the approach into a single identification method [**Chapter 8**]. Finally for the reassembly methods, **Section 9.8** shows the results of reassembly with the lowest result for data set 2 for the thumbcache_96 file with 81.26% of available fragments correctly reassembled; the result was due to the high number of clashes on image-only fragments. The thumbcache_idx file was reassembled with 100% accuracy.

10.6 Automation

“Automation is defined as a scientific process which has been implemented to minimise the interaction of an analyst.” [**Chapter 4.3.2**]

This process has been shown to automate the identification and reassembly of thumbnail cache file fragments by implementing a selection of methods documented in **Chapters 7, 8, and 9**. The extraction of thumbnail cache artefacts is also automated and documented in **Section 9.10**. As the previous section showed both the identification and reassembly stages of this methodology have failed to maximise the data available [**Sections 7.10, 8.7, and 9.8**]; therefore the resulting reassembled files may not be completed and may require further work by an analyst to identify any missing fragments should they exist. In the case of a potential clash of fragments for a position in a file during reassembly an analyst would have to perform further verification by hand. The extensive log files would assist an analyst by documenting the reasoning for each potential fragment and providing the original locations of the fragments; the original locations allow an analyst to place the pieces in context.

10.7 Discussion

Both live system and identified file fragments of thumbnail cache can be analysed using replicable methods which can be corroborated by an analyst either using a second tool or by hand. The clarity of the documentation affects the ability of an analyst to replicate the methodology; in this research all methods were implemented to produce detailed logs to provide reasoning for any action and to maintain a relationship between each artefact or fragment and the original data. The use of logs allows an analyst to confirm important results independently; although it would be straightforward for an analyst to replicate the results of the structural and behavioural studies conducted on live thumbnail caches, the techniques documented in **Chapters 7-9** may require additional cross disciplinary knowledge.

The use of an extensive logging system ensures that any action taken by any implemented process during this research can be justified and the relationship to the original data can be shown. By splitting the process of extracting thumbnail cache artefacts from unallocated space into stages (identification, reassembly, and extraction), it assists in ensuring the process has understandable consequences; the relationship to the original data can be explained clearer if the process is broken down.

By ascertaining the potential behaviour which led to an artefacts current state it is necessary to establish relationships between the thumbnail cache and other artefacts on the system. The relationships provide corroboration and supplementary information about the thumbnail cache artefacts which can assist in establishing potential user behaviour and if the system was operating typically. Both the live thumbnail cache and the reassembled fragments can potentially form relationships with the related system artefacts on the live system. However the methods implemented for identifying and reassembling file fragments were specifically designed for thumbnail cache files and therefore

fragments of potentially related files were not recovered. Therefore the corroboration of external artefacts was not maximised; for thumbnail cache files which have been identified and relate to deleted user accounts or previous installations this may prevent any relationships being formed with other artefacts. In this case the thumbnail cache file would be isolated and may therefore only provide a limited subset of information.

A live thumbnail cache is complete and therefore the maximum amount of data for the thumbnail cache is available; other live artefacts can potentially form relationships with the thumbnail cache to supplement and corroborate the information identified in the thumbnail cache. File fragments recovered from unallocated space may not potentially be fully recoverable and therefore the maximum amount of data may not be available. The process used during this research was able to identify 3 classifications of thumbnail cache fragment with 100% accuracy and no false positives using the test corpus. However it is likely these figures would be slightly lower in a less controlled environment as it was not feasible to test every piece of software and file type available to a user.

Both the extraction of artefacts and the identification and reassembly of file fragments process have automated implementations which aim to minimise the potential interaction for an analyst. For live thumbnail caches the process can provide a complete extraction of the files and a detailed report. It is possible that a reassembled file is not complete or there are potential clashes for some sections of the file; in these circumstances an analyst would have to continue the analysis by hand. The pre classification of files may assist an analyst in narrowing down the potential search for missing fragments. The log file can also assist an analyst by allowing them to put the reassembled fragments in context; the context may narrow down potential fragments for the missing sections.

10.8 Conclusion

This chapter evaluated the evidential value of thumbnail cache file fragments identified and recovered during this research. The methods created during this research showed it is possible to recover a significant proportion of thumbnail cache file fragments. It has been shown that complete thumbnail caches in a live operating system have a higher evidential value as they successfully meet a larger selection of the criteria. Forming relationships between the thumbnail cache and other artefacts on the system can provide corroboration and supplementary information about the systems activity. Thumbnail cache file fragments identified and reassembled from unallocated space may not be complete and therefore do not maximise the amount of data available. Therefore it may not be possible to maximise corroboration by building extensive relationships with other artefacts. However forming relationships with other file types which may be identified using the same methodology or creating relationships with artefacts on the live file system may assist in increasing their evidential value.

Finally, artefacts in unallocated space have significant evidential strengths in being able to provide information about previous installations of the operating system or user accounts which have been deleted. The file fragments may also provide significant information in situations where a user has removed artefacts from the system which relate to their activity. This chapter has shown that both live and deleted thumbnail caches have unique evidential strengths and that when used together in conjunction can provide a significant insight into the systems behaviour. The following chapter evaluates the research conducted in this thesis and discusses the approaches taken.

11 Discussion

11.1 Introduction

The aim of this research was to evaluate the evidential value of thumbnail cache file fragments identified in unallocated space; the research to achieve this has been documented in this thesis. This Chapter provides a critical evaluation of the methodologies employed during this research; it begins by reviewing the general methodology for this research [**Section 11.2**]. This is followed by an evaluation of the methods employed for identifying the structure and behaviour of complete thumbnail caches [**Section 11.3**]; **Section 11.4** evaluates the methods used for thumbnail cache file fragment identification. **Section 11.5** evaluates the methods used for thumbnail cache file fragment reassembly; **Section 11.6** evaluates the research conducted into solving the image fragment identification problem. In **Section 11.7** the generalisation of this research is evaluated; this is followed by the chapter conclusion in **Section 11.8**.

This research required a set of criteria for evidential value to be formed to enable the evaluation of the research; the criteria were developed by analysing existing forensic computing methodologies and the various types of constraints placed on evidence. Various categories of constraint were examined to identify their potential impact on evidence in a digital investigation; the results of the constraint research were combined to create the criteria. This approach resulted in a wider range of issues being considered than the common approach among practitioners of relying on the ACPO guidelines. The criteria may have been improved by further discussion with practitioners to identify organisation specific constraints; however it was felt that this research would benefit from only considering general constraints to enable the research to be adapted to a wider variety of applications.

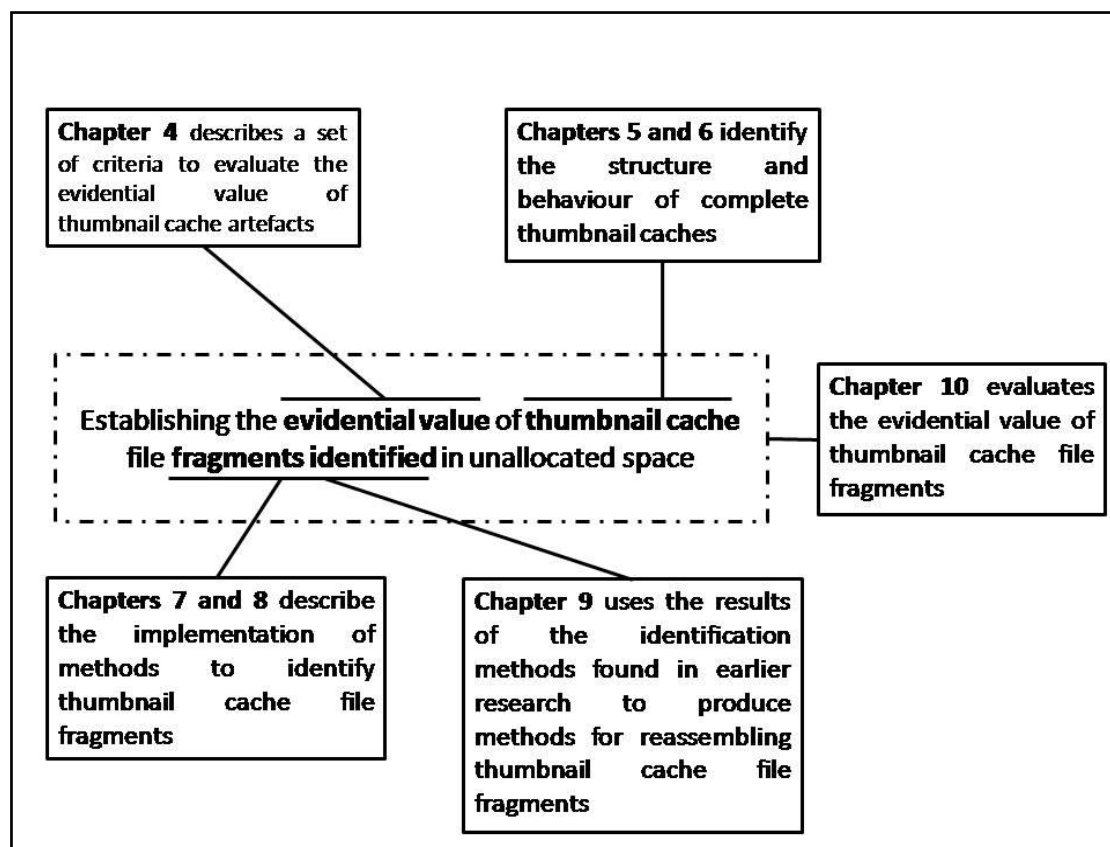


Figure 11-1: A breakdown of aim of this research into smaller research problems

After the development of criteria for assessing the evidential value research was conducted into the evidential value of complete and identified fragments of thumbnail cache files; a critical evaluation of this is provided in the sections below. *Figure 11.1* highlights the key words in the aim and shows how they directly relate to smaller research problems which are documented in separate chapters.

11.3 Structure and Behaviour of Complete Thumbnail Caches

In order to evaluate the evidential value of thumbnail cache file fragments identified from unallocated space it was necessary to firstly research the potential evidential value of a complete thumbnail cache located in a live file system. A complete cache in a live file set is likely to provide the maximum

evidential artefacts as it has all the parts of the cache and can form relationships with other parts of the system which are also complete. The structural and behavioural study focused on thumbnail caches in Ubuntu, Kubuntu, and Windows 7. An earlier preliminary study of thumbnail cache artefacts was conducted during this research to assist in identifying potential differences between the Windows Vista and Windows 7 thumbnail cache implementations. The Windows 7 operating system also utilises the earlier *thumbs.db* styles of thumbnail cache; each thumbnail cache in this range was tested using the same experiments in order to ensure accurate and unbiased results.

The experiments were produced to mimic typical user actions on the system; since thumbnail caches generally cache user generated files the user action involved the creation, modification and deletion of various types of user created files on various kinds of storage media. These experiments aimed to encompass the likely range of values which would occur in the thumbnail caches; however it is not possible to cover all user behaviour which may directly or indirectly affect the contents of the thumbnail cache as it is not known. It could be suggested that a wider study of potential user behaviour should have been conducted before performing the experiments; however since it is not viable to perform tests on every piece of potentially relevant software with each operating system it was necessary to limit the scope of this research to the most likely to be of interest in an investigation. Further, by analysing the real thumbnail caches in Data Set 2 alongside the experimental data, there was an opportunity to identify and explore artefacts which had not been seen in the experiments thereby furthering the structural and behavioural studies.

One consideration of this research was the stage at which to investigate the internal structure of the thumbnail cache files. Identifying the structures too early may influence the experimentation in an attempt to directly influence the variables identified in the structure. By performing a set of experiments before

analysing the file type it is possible to concentrate on typical user behaviour instead of the bias towards the known information; this could be followed up with a set of experiments attempting to alter the known variables. A comparison of the results from the initial user experiments and those achieved from the second set of experimentation provide a wider insight into the file and potential sources of further investigation.

This research has highlighted the importance of contextual analysis of digital artefacts by showing the difference small variations in file structures and differing methods of storing data relating to the same user interaction can have on the artefacts found on a system. This research has provided documentation on the structure and behaviour of thumbnail caches for digital analysts which was not previously documented in a peer reviewed publication. Whilst the research could have gone further into a study of the *thumbs.db* cache as this research has shown it is still implemented in Windows 7 the complex OLE2 structure was felt to be too time consuming to attempt file fragment identification on this type on top of the thumbnail caches already under investigation.

11.4 Thumbnail Cache File Fragment Identification

A significant contribution of this research was the comparative study of file fragment identification techniques which led to the creation of a hybrid technique which improved the identification of thumbnail cache file fragments. It was necessary to adapt existing techniques to identify single file fragments. If a single method had been created for the identification the criteria for maximising the data may not have been met; by using the strengths of individual methods it was possible to maximise the identification of fragments in each classification.

Whilst several common file fragment identification techniques were compared time and resource constraints did not permit the evaluation of every available

technique; therefore an open question is do any other existing techniques have similar identification statistics when adapted to single file fragment identification. It is also possible that the methodologies selected and implemented were biased by the creator's knowledge and or experience; however each method was documented to assist an analyst in corroborating the results.

The creation of a hybrid method for the identification of file fragments allowed the strength of each individual method to be combined and further modification to be made based upon the initial experimental results. The methodology for creating a hybrid method could be applied to any file type by analysing both the structure and behaviour of that type; the information obtained from this analysis could then be used to implement the common identification methods and the results could be used to create a file type specific hybrid. This research highlighted the strengths of structural and syntactical identification methods; some basic file types may only require this type of identification to retrieve the maximum number of fragments.

Following the completion of this research, a new challenge was launched by DFRWS in 2012, which focused on the identification of single file fragments. This challenge showed the change in research interest of the forensic computing community towards identifying small pieces of information [DFRWS, 2013]. Results from the 2012 challenge have not been published; the current 2013 challenge is also focused on the identification of single file fragments.

11.5 Thumbnail Cache File Fragment Reassembly

In order to ascertain the evidential value of the thumbnail cache file fragments identified in unallocated space it is necessary to reassemble the fragments into their appropriate files. Reassembling the files can provide further context than a

singular file fragment; by combining multiple fragments together into their various files it is possible to build up further context on events that occurred. This research focused on using the identified structures and syntax for each thumbnail cache for reassembly; since there may be multiple versions of similar thumbnail caches in unallocated space it was necessary to refine the reassembly process. In order to show the reassembled file is an accurate representation of the order the fragments occurred in when they were last present in a file in the live data set it is necessary to show a relationship between the fragments.

A contribution of this research are the methods for thumbnail cache file fragment reassembly which use the unique traits learnt about each type of thumbnail cache file to maximise the potential for valid fragment reassembly. This research has shown that by understanding the structure and behaviour of a file type it is possible to adapt the reassembly method to take into consideration this information and therefore increase the accuracy in the reconstruction.

A strength of this research was the documentation created are the two logs created during the reassembly process (**Section 9.3.2**); providing sufficient information to permit an analyst to corroborate the software's output must be balanced with the need to produce logs which are understandable and do not require an unreasonable amount of time or resources to use. The logs show the provenance of the results produced; an analyst can verify the results of the approach using a second technique if required.

This research could potentially have utilised methods from other disciplines such as techniques utilised in Computer Science for reconstructing jigsaw puzzles for the reassembly phase. For this research it was deemed more appropriate in a Forensic Computing context to focus on reassembly methods based upon known information about the file and the file system; an analyst

could adapt the use of structural and syntactical information to reassemble other file types. Using known information to reassemble the files assists in showing relationships between the fragments and makes it easier for an analyst to explain their results and have them corroborated; as any fragment reassembly performed may need to be justified in a Court of law any method used must be understandable and have a scientific basis.

11.6 Image Format Problems

This research found problems with both the identification and reassembly of image-only file fragments; this problem resulted from the fact that image-only fragments from the thumbnail cache contain the same type of data as image-only fragments from other file types. This meant that for the identification phase it was necessary to have a significant amount of false positive identification in the image-only fragment category; it was important to ensure the maximum number of thumbnail cache image-only file fragments were identified to ensure a maximum number of fragments were available for the reassembly research. The methods produced in this research pre-process the storage media to provide a relatively small amount of potential fragments; if an analyst identified any potentially important artefacts in the data produced by these methods it would need to be verified using a second method. Dual tool verification is a common approach in Forensic Computing and provides a method for verifying the results that were obtained.

The significant number of fragments identified as belonging to the thumbnail cache image-only category impacted on the reassembly methods; the large number of fragments meant a large number of evaluations were required which slowed down the processing time considerably. This research could have been improved by further research into refining the identification of image-only thumbnail cache file fragments; any reduction in the number of fragments being classified in the image-only category would have a significant impact on the

processing time of the reassembly method. For this research the identified information maximised the data retrieved and provided information which assisted in proving the research hypothesis.

In **Chapter 8** the research considered the characteristics of JPEG file fragments, both from the thumbnail cache and other sources. The research considered JPEG start of sequence markers; *table 8.1* lists the 8 sequence markers found in the visual thumbnails tested. As all 8 sequence markers occurred in every JPEG thumbnail cache image tested a decision was made to exclude fragments which contained any other sequence starting with 0xFF. This meant that other sequences, which were valid markers in the JPEG specification, would have resulted in a fragment being rejected. Given the number of thumbnails tested it is possible that the 8 markers are the only ones which occur in visual thumbnails; however it is also possible that thumbnail caches outside the test data may contain other sequences.

The use of observed characteristics instead of the specification made it possible to significantly reduce the number of false positive JPEG identifications; this led to a smaller number of fragments for the reassembly method. The reduction in false positives improved the performance and feasibility of the reassembly approach. If JPEG visual thumbnails exist which contain other sequence markers they will not be identified; therefore this may significantly impact the artefacts available for an analyst to present in his report. Further work is required to explore the characteristics of JPEG and JPEG thumbnail cache file fragments to assist with distinguishing between them for file fragment identification and reassembly.

11.7 Generalising the approach to file carving

The use of the approach on different styles of thumbnail cache highlights the ability to generalise this method to other file types. One outcome of this research is the improvement which can be made to both the identification and reassembly of a file type by understanding the file type. The methodology used in this research focused on using file type specific information to improve the ability to identify and understand the file fragments; this increased understanding of the file type also assists when performing a contextual analysis of a complete file in a live data set.

Whilst using a file type specific recovery method may be resource intensive it has the potential to identify and reconstruct important file types; it has been shown that little research has been conducted into identifying single file fragments. A file fragment may potentially add vital information to a contextual analysis and provide an artefact which may be of interest to the investigation. This research also has a prototype logging system which aims to provide enough detail to enable an analyst to corroborate the software's results whilst making it readable and not unnecessarily comprehensive. The logging system has the potential to be adapted to the needs of each individual; allowing the removal or addition of fields in the logging system would make a more customisable environment.

This research used hashes and CRC values which were part of the thumbnail cache file structures to assist with both the identification and reassembly of the fragments. Both the hash and CRC values permitted multiple equations to be formed which enabled the identification of valid file fragments; this method proved very effective in validating the results and reducing the number of false positives. Therefore a consideration when adapting this approach to other file types would be the identification of values which can potentially form equations to validate and identify other fragments in the classification.

In **Chapter 9.5** a decision was taken to present the analyst with all valid image reconstructions. The file system information relating to fragments in unallocated space may not be available; this can make forming relationships between fragments difficult. There may be multiple versions of the thumbnail cache and subrecords relating to individual files which may lead to a file fragment reassembly which appears valid but was not an actual file on the device. This led to the decision to present the analyst with all valid image reconstructions; allowing the analyst to review the logs showing the reasoning which led to each reconstructed images. The analyst can then verify the information and make their own decision about what artefacts to present in their report.

11.8 Conclusion

In conclusion, this research has taken a large research question and logically broken it down into smaller research goals with the aim of supporting or disproving the central hypothesis. This has been achieved by creating a set of criteria in order to evaluate the evidential value of thumbnail cache artefacts; the criteria were developed by evaluating the potential constraints on digital evidence and applying this information to the thumbnail cache. The development of the criteria before performing further investigation ensured that the criteria were independent of the results of this research. This research has described both the structure and behaviour of several popular implementations of the thumbnail cache; this research could have been improved by further investigation into the *thumbs.db* format.

This research compared a variety of commonly implemented file fragment identification techniques; identification methods independently attempted to identify file fragments based solely on their content. Reassembly methods were based on the structural and behavioural research performed as this was deemed the most suitable method for Forensic Computing. Further research

into both identification and reassembly of image-only thumbnail cache file fragments may further refine the methodologies produced and decrease processing time. Finally since there is no standard implementation of a thumbnail cache the methodologies created in this research were kept as general as possible; this means the research can easily be generalised to other file types to improve identification and reassembly of their file fragments.

12 Conclusions and future work

12.1 Research Conclusions

This chapter provides a summary of the thesis by providing a description of the contributions of this research, a description of possible future work and finally a summary of the research performed and the conclusions resulting from this.

12.2 Contributions of Research

Throughout this thesis several contributions have been shown, this section briefly describes each of those contributions. Firstly **Chapter 4** has developed a set of criteria for establishing the evidential value of artefacts from operating system thumbnail caches. In **Chapters 5 and 6** the structure and behaviour of 3 common thumbnail caches are described; until Morris [2011a] published a paper based on the results of this research only selected part of the structure and behaviour were documented in the public domain. Alongside understanding the structure and behaviour of the thumbnail cache this research also showed how relationships can be formed between the thumbnail cache and other system artefacts; each relationship was evaluated to show its strength and where possible the ways in which the relationship could be strengthened.

In **Chapter 7**, this research compared a selection of the most commonly used file carving approaches and evaluated their performance against identifying thumbnail cache file fragments. This enabled a critical evaluation of the strengths and weaknesses of each file carving method and resulted in a new approach based on a hybrid of the most common methods being developed for the identification of thumbnail cache file fragments.

In **Chapter 9** methods for reassembling file fragments from each thumbnail cache file type evaluated during this research were created; this in conjunction with the fragment identification methods created in **Chapters 7 and 8** allows the reassembly of thumbnail cache files in unallocated space which may potentially provide useful artefacts to an analyst. The reconstruction methods benefitted from an in depth understanding of the file formats allowing each method to be tailored to the unique structure of each file type; this resulted in an accurate and explainable reconstruction method.

Finally the aim of this research was to establish the evidential value of thumbnail cache file fragments identified in unallocated space; this was achieved in **Chapter 10** which compares the evidential value of live and deleted thumbnail caches and contributes to the understanding of their evidential value.

12.3 Future Work

In this research it was noted that whilst **Chapter 8** improved the ability of the identification method implemented at the end of **Chapter 7** to identify image-only thumbnail cache file fragments work could still be done to further improve this. As discussed in **Section 11.6** further work is also required into the identification of characteristics which can distinguish between JPEG and JPEG thumbnail cache file fragments. Image-only fragments are interesting challenges as a wide variety of other file formats also create image-only fragments which share the same general characteristics.

This research has involved a variety of different thumbnail cache file structures; each method used for identification and reassembly was in a format which was general enough to be applied to each structure effectively. Therefore the methodology could be generalised to any other file format; this provides

potential for improving the identification and reassembly of single file fragments for important file types.

As well as looking at other file types it is important to note that there are various styles of thumbnail cache such as those used with photographic applications; each of these thumbnail caches may possess a unique structure and behaviour and therefore is an opportunity for further research.

Finally, future work could be conducted on the changes that occur to the thumbnail cache overtime; in order to reconstruct the order in which artefacts were added to the cache, research could be performed into the creation of event timelines for thumbnail cache artefacts. If several thumbnail cache files were identified in unallocated space the ability to determine an event timeline of the modification of the artefacts would assist in placing the information in context.

12.4 Summary

Thumbnail caches were originally implemented to minimise unnecessary use of system resources by storing visual thumbnails and their associated metadata; whilst the basic definition of a thumbnail cache is known there is currently no standard structure or behaviour for its implementation. This lack of standardisation creates an interesting test ground for research as methods created to access data from the thumbnail cache are already generalised to an extent in order to be applicable to the different implementations.

The thumbnail cache is part of the operating system and users may not necessarily be aware of its function or existence; however with the growth of the internet and user understanding it is becoming increasingly simple to access

information about hiding your actions. This growth of understanding makes it more likely that a user wishing to hide information relating to their activities may seek to remove sources of potential artefacts such as the thumbnail cache. If the thumbnail cache is deleted it can still be found in unallocated space; potentially fragments of the thumbnail cache may exist in unallocated space for a significant time after deletion. Whilst the evidential value of fragments of data recovered from unallocated space may be less than a complete file the fragments have the potential to provide information that is not available in the live file set and which may provide vital information to prove or disprove a hypothesis.

This research has shown that the ability to understand a file format can improve the ability to construct methods for the identification and reassembly of file fragments. This was highlighted by the creation of proof of concept software which implemented a hybrid identification approach which was refined using structural and syntactical knowledge about the file identified during this research. Further support for the importance of understanding a file type came from the use of this information in producing specific reassembly methods; each method relied upon unique file characteristics to reassemble the files as accurately as possible.

This research has shown the structure and behaviour of common operating system thumbnail caches and highlighted the importance of contextual analysis of an artefact in order to fully understand the user and system behaviour it represents. The methodologies produced for the identification and reassembly of thumbnail cache file types showed that there are some forms of information which may only be retrieved from unallocated space. Whilst a complete thumbnail cache in a live file set gives an analyst the opportunity to develop relationships and construct a detailed contextual analysis of the systems behaviour deleted thumbnail cache artefacts may also assist in an analysis. For example thumbnail caches in unallocated space may belong to previous

installations of the operating system or deleted user profiles; this information may provide unique artefacts which alter the meaning of a contextual analysis of the system.

REFERENCES

- Access Data, 2012. FTK. Available at: <http://www.accessdata.com/products/digital-forensics/ftk>. [Accessed June 14, 2012].
- ACPO, 2010. Good practice guide for computer based evidence. Available at: http://www.7safe.com/electronic_evidence/ACPO_guidelines_computer_evidence.pdf [Accessed October 20, 2012].
- Anderson, C.A., Benjamin, A.J. & Bartholow, B.D., 1998. Does the gun pull the trigger? Automatic priming effects of weapon pictures and weapon names. *Psychological Science*, 9(4), pp.308–314.
- Bainbridge, D., 2007. Criminal law tackles computer fraud and misuse. *Computer Law & Security Review*, 23(3), pp.276–281.
- Ball, C., 2007. Cross-examination of the Computer Forensics Expert. Available at: <http://www.craigball.com/expertcross.pdf> [Accessed December 11, 2011].
- Bem, D., Huebner, E., 2007. Computer forensic analysis in a virtual environment. *International journal of digital evidence*, 6(2).
- Berinato, S., 2007. How Online Criminals Make Themselves Tough to Find, Near Impossible to Nab. *CIO.com*. Available at: http://www.proofspace.com/UserFiles/File/CIO_com2007_05_31.pdf. [Accessed July 24, 2012].
- Bogen, A., Dampier, D., 2005. Preparing for Large-Scale Investigations with Case Domain Modeling. *Digital Forensics Research Workshop, New Orleans, LA*, pp.1–10.
- Boyd, C., Forster, P., 2004. Time and date issues in forensic computing - A case study. *Digital Investigation*, 1(1), pp.18–23.
- Buchholz, F., Spafford, E., 2004. On the role of file system metadata in digital forensics. *Digital Investigation*, 1(4), pp.298–309.
- Burghardt, A. et al., 2008. Using the HFSD journal for deleted file recovery. *Digital Forensics Research Workshop, Baltimore, MD*, pp.76–82.
- Byers, S., 2004. Information leakage caused by hidden data in published documents. *Security & Privacy, IEEE*, 2(2), pp.23–27.
- Calhoun, W.C. & Coles, D., 2008. Predicting the types of file fragments. *Digital Investigation*, 5(SUPPL.), pp.S14–S20.

- Carnagey, N.L., Anderson, C.A. & Bushman, B.J., 2007. The effect of video game violence on physiological desensitization to real-life violence. *Journal of Experimental Social Psychology*, 43(3), pp.489–496.
- Carney, M., Rogers, M, 2004. The Trojan Made Me Do It: A First Step in Statistical Based Computer Forensics Event Reconstruction. *International Journal of Digital Evidence*, 2(4).
- Carrier, B., 2002. *Open Source Digital Forensics Tools*,@stake Research Report.
- Carrier, B., Spafford, EH, 2003a. Getting physical with the digital investigation process. *International Journal of Digital Evidence*, 2(2), pp.1–20.
- Carrier, B., 2003b. Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of digital evidence*, 1(4), pp.1–12.
- Carrier, B., 2005. *File system forensic analysis*, Addison Wesley. Available at: <http://www.amazon.co.uk/System-Forensic-Analysis-Brian-Carrier/dp/0321268172> [Accessed March 29, 2013].
- Carrier, B., Spafford, EH, 2006. Categories of digital investigation analysis techniques based on the computer history model. *digital investigation*, pp.121–130.
- Carrier, B., 2011. Autopsy. Available at: <http://www.sleuthkit.org/autopsy/>. [Accessed February 12, 2012].
- Casey, E., 2004. Tool review WinHex. *Digital Investigation*,1(2), pp.114–128.
- Casey, E, 2006. The value of behavioral analysis in digital investigations. *Digital Investigation*, 3(2), pp.57–58.
- Casey, E, 2007. What does “forensically sound” really mean? *Digital Investigation*, 4(2), pp.49–50.
- Casey, E, 2008. The impact of full disk encryption on digital forensics. *ACM SIGOPS Operating Systems Review*, 42(3), pp.93–98.
- Castiglione, A., De Santis, A. & Soriente, C., 2007. Taking advantages of a disadvantage: Digital forensics and steganography using document metadata. *Component-Based Software Engineering of Trustworthy Embedded Systems*, 80(5), pp.750–764.
- Chivers, H., Hargreaves, C., 2011. Forensic data recovery from the Windows Search Database. *Digital Investigation*, pp.114–126.

- Chow, K. et al., 2007. The Rules of Time on NTFS File System. *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering*, pp.71–85.
- Ciardhuáin, S., 2004. An extended model of cybercrime investigations. *International Journal of Digital Evidence*, 3(1), pp.1–22.
- Cohen, M.I., 2007. Advanced carving techniques. *Digital Investigation*, 4(3-4), pp.119–128.
- Cooper, P., 2005. Speciation in the computing sciences: digital forensics as an emerging academic discipline. In *Proceedings of the 2nd annual conference on Information security curriculum development*. New York, NY, USA: ACM, pp. 19–23.
- Cuturicu, C., 1999. THE JPEG COMPRESSION and THE JPG FILE FORMAT. Available at: <http://www.opennet.ru/docs/formats/jpeg.txt> [Accessed July 8, 2010].
- DFRWS, 2013. Digital Forensics Research Workshop. Available at: <http://www.dfrws.org/>. [Accessed February 13, 2013]
- Dickerman, D., 2006. Advanced Data Carving. In *Digital Forensics Research Workshop - 2006 Data Carving Challenge*. Lafayette, In.
- Douceur, J., Bolosky, W., 1999. A large-scale study of file-system contents. *ACM SIGMETRICS Performance Evaluation ...*, 27(1), pp.59–70.
- Douglas, J., 2009. *Forensic Implications of Windows Desktop Search*. MSc Thesis. Cranfield University.
- Dreyfus, S., 1998. COMPUTER HACKERS: JUVENILE DELINQUENTS OR INTERNATIONAL SABOTEURS? In *Internet Crime Conference Melbourne*. pp. 16–17.
- Dufournaud, Y., Schmid, C., Horaud, R., 2000. Matching images with different resolutions. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*. pp. 612–618.
- Duren, M., Hosmer, C., 2002. Can Digital Evidence Endure the Test of Time? *Digital Forensic Research Workshop*. Syracuse, NY.
- Dziadosz, S., Chandrasekar, R., 2002. Do thumbnail previews help users make better relevance decisions about web search results? Research and development in information retrieval Conference, pp.365–366.
- Elerath, J., 2007. Hard Disk Drives: The Good, the Bad, and the Ugly. *Queue*, 5(6), pp.28–37.

- Facebook, 2013. Facebook. Available at: www.facebook.com [Accessed March 29, 2013].
- Fellows, G., 2005. The joys of complexity and the deleted file. *Digital Investigation*, 2(2), pp.89–93.
- Finke, J., 2004. Thumbnail Managing Standard. Available at: <http://specifications.freedesktop.org/thumbnail-spec/thumbnail-spec-latest.html> [Accessed July 24, 2012]
- Fleischmann, S., 2008. X-Ways Forensics / WinHex. Available at: <http://www.x-ways.net/winhex/manual.pdf> [Accessed March 29, 2013].
- Fogel, D.B., 2001. *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann.
- Forensics Wiki, 2013. Forensics Wiki. Available at: http://www.forensicswiki.org/wiki/Main_Page [Accessed March 29, 2013].
- Forte, D., Power, R., 2006. Electronic discovery: digital forensics and beyond. *Computer Fraud & Security*, (April).
- French, P., 2001. Scrub, Scrub, Scrub. *Data Forensics*.
- Galvão, R., 2006. Computer Forensics with Sleuth Kit and The Autopsy Forensic Browser. *The International Journal of Forensic Computer Science*. 1(1) pp.41–44.
- Garfinkel, S.L., 2007. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 4(SUPPL.), pp.2–12.
- Garber, L., 2001. Computer Forensics: High-Tech Law Enforcement. *IEEE Computer*.
- Gladyshev, P., Patel, A., 2004. Finite state machine approach to digital event reconstruction. *Digital Investigation*, 1(2), pp.130–149.
- Gladyshev, P., Patel, A., 2005. Formalising event time bounding in digital investigation. *International Journal of Digital Evidence*, 4(2).
- Google, 2013. Google. Available at: www.Google.co.uk. [Accessed March 03, 2012]
- Greenfield, R.S., 2002. *Cyber Forensics: A Field Manual for Collecting, Examining, and Preserving Evidence of Computer Crimes*, Google Ebook.
- Griffiths, J., 1979. Is law important. *NewYork Univeristy Law Review*., 54, p.339.

- Guidance Software, 2011. Encase. Available at:
<http://www.guidancesoftware.com/encase-forensic.htm> [Accessed December 28, 2011].
- Guo, H., Xu, M., 2011. A Method for Recovering JPEG Files Based on Thumbnail. In *2011 International Conference on Control, Automation and Systems Engineering (CASE)*. IEEE, pp. 1–4.
- Hall, G., Davis, W., 2005. Toward Defining the Intersection of Forensics and Information Technology. *International Journal of Digital Evidence*, 4(1), pp.1–20.
- Hama, G., Pollitt, M., 1996. Data reduction – refining the sieve. In *Second International Conference on Digital Evidence*.
- Hannan, M., 2004. To Revisit: What is Forensic Computing? *Proceedings of the Second Australian Computer, Network & Information Forensics Conference*. Available at:
<http://igneous.scis.ecu.edu.au/proceedings/2004/adf/Hannan.pdf> [Accessed March 29, 2012].
- Hargreaves, C., Chivers, H., 2007. Potential Impacts of Windows Vista on Digital Investigations. *Proceedings of 2nd Advances in Computer Security and Forensics*, Liverpool, UK.
- Hargreaves, C., Chivers, H., Titheridge, D., 2008. Windows Vista and digital investigations. *Digital Investigation*, 5(1-2), pp.34–48.
- Hargreaves, C., 2009. *Assessing the reliability of digital evidence from live investigations involving encryption*. PhD Thesis. Cranfield University.
- Harley, D., Citp, B., Cissp, F., 2010. SODDImy & the Trojan Defence. In *4th International Conference on Cybercrime Forensics Education and Training*. Canterbury, UK.
- Harris, R., 2006. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digital Investigation*, 3, Supplem(0), pp.44–49..
- Harris, R., 2007. Using Artificial Neural Networks for Forensic File Type Identification. *CERIAS Tech Report*. Available at:
https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2007-19.pdf [Accessed July 19, 2010].
- Hayes, D., Qureshi, S., 2009. Implications of Microsoft Vista operating system for computer forensics investigations. IEEE Systems, Applications and Technology Conference, 2009. LISAT '09. Long Island, NY.

- Helix, 2011. Helix. Available at: <http://www.e-fense.com/products.php> [Accessed August 6, 2011].
- Hurlbut, D., 2005. Thumbs.db Files Forensic Issues. *Access Data Training Whitepaper*.
- Joukov, N., Papaxenopoulos, H. & Zadok, E., 2006. Secure deletion myths, issues, and solutions. In *Proceedings of the second ACM workshop on Storage security and survivability*. pp. 61–66.
- JPEG, 2013. Joint Photographic Experts Group. Available at: <http://www.jpeg.org/> [Accessed January 5, 2013].
- Kenneally, E., 2004. Digital logs-Proof matters. *Digital Investigation*, 1(2), pp.94–101.
- Kerr, F., 2006. Media analyses based on Microsoft NTFS file ownership. *Forensic science international*, 162(1-3), pp.44–8.
- Khan, M., Chatwin, C. & Young, R., 2007. A framework for post-event timeline reconstruction using neural networks. *Digital Investigation*, 4(3-4), pp.146–157.
- Kornblum, J., 2008. Using JPEG quantization tables to identify imagery processed by software. *Digital Investigation*, 5(SUPPL.), pp.S21–S25.
- Kwon, H. et al., 2008. A Tool for the Detection of Hidden Data in Microsoft Compound Document File Format. *IEEE International Conference on Information Science and Security*.
- Lyu, S., Farid, H., 2005. How realistic is photorealistic? *IEEE Transactions on Signal Processing*, 53(2 II), pp.845–850.
- McKay, C. et al., 2008. Image acquisition forensics: Forensic analysis to identify imaging source. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*. pp. 1657–1660.
- McKemmish, R., 1999. What is forensic computing? *Trends and Issues in Crime and Criminal Justice*, (118).
- Memon, N., Pal, A., 2006. Automated reassembly of file fragmented images using greedy algorithms. *IEEE Transactions on Image Processing*, 15(2), pp.385–393.
- Mercuri, R., 2010. Criminal Defense Challenges in Computer Forensics. In *Digital Forensics and Cyber Crime*. Springer, pp. 132–138.
- Meyers, M. & Rogers, Marc, 2004. Computer forensics: the need for standardization and certification. *International Journal of Digital Evidence*, 3(2), pp.1–11.

- Microsoft, 2007. Fundamental computer investigation guide for windows. Available at: <http://www.microsoft.com/en-us/download/details.aspx?id=23378>. [Accessed August 17, 2012].
- Microsoft, 2012a. MSDN. Available at: <http://msdn.microsoft.com/en-gb/> [Accessed February 23, 2012].
- Microsoft, 2012b. Procmon. Available at: <http://technet.microsoft.com/en-gb/sysinternals/bb896645.aspx>. [Accessed December 01, 2012]
- Microsoft Press, 2002. Microsoft computer dictionary. Available at: <http://www.amazon.co.uk/Microsoft-Computer-Dictionary-Fifth-Edition/dp/0735614954> [Accessed March 29, 2013].
- Morris, S.; Chivers, H.; 2010. A comparative study of the structure and behaviour of the operating system thumbnail caches used in Kubuntu and Ubuntu (9 . 10 and 10 . 04). In *Cybercrime Forensics Education and Training*. Canterbury, UK.
- Morris, S.; Chivers, H.; 2011a. An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache. In *Cyberforensics*. Glasgow, UK.
- Morris, S.; Chivers, H.; 2011b. Forming a Relationship between Artefacts identified in thumbnail caches and the remaining data on a storage device. In *Cybercrime Forensics Education and Training*. Canterbury, UK.
- Morris, S.; Chivers, H.; 2013. Thumbnail Cache File Fragment Identification using a Bayesian Network. In *Cyberforensics*. Cardiff, UK.
- National Center for Forensic Science, 2003. Digital Evidence in the Courtroom: A Guide for Preparing Digital Evidence for Courtroom Presentation. Available at: http://www.classstudio.com/scaltagi/papers/grad_papers/forensics/Palmer/digital_evidence_in_courtroom.pdf [Accessed March 11, 2011].
- Nethercott; R v Nethercott [2001] EWCA Crim 2535
- NISO, 2001. Understanding Metadata. Available at: <http://www.niso.org/publications/press/UnderstandingMetadata.pdf>. [Accessed January 3, 2013].
- NIST, 2004. Forensic Examination of Digital Evidence: A Guide for Law Enforcement. Available at: <http://www.ncjrs.gov/App/abstractdb/AbstractDBDetails.aspx?id=199408> [Accessed February 12, 2013].
- NIST, 2013. NIST computer forensics tool testing project. Available at: <http://www.cftt.nist.gov/> [Accessed January 3, 2013].

- Noblett, M., Pollitt, M., Presley, L., 2000. Recovering and examining computer forensic evidence. *Forensic Science Communications*, 2(4), pp.1–13.
- NOXA, 2008. NOXA: Vista Thumbnail Cache. Available at: <http://www.noxa.org/blog/2008/02/02/vista-thumbnail-cache/>. [Accessed October 31, 2011]
- OED, 2011. OED - Oxford English Dictionary. Available at: <http://www.oed.com/> [Accessed May 25, 2011].
- Open Office, 2012. Open Office. Available at: <http://www.openoffice.org/> [Accessed January 18, 2012].
- Ozdemir, S., 2002. Non-Malicious Destruction of Data. *SANS Institute InfoSec Reading Room*.
- Pal, A., Sencar, H., Memon, N, 2008. Detecting file fragmentation point using sequential hypothesis testing. *Digital Investigation*.5 Supp, pp. S2-S13.
- Pal, A., Memon, N, 2009. The evolution of file carving. *IEEE Signal Processing Magazine*, 26(2), pp.59–71.
- Palmer, G., 2002. Forensic analysis in the digital world. *International Journal of Digital Evidence*, 1(1), pp.1–6.
- Pan, L., Batten, L., 2007. An effective and efficient testing methodology for correctness testing for file recovery tools. *IEEE Intelligent Information Hiding and Multimedia Signal Processing*.
- Park, B., Park, J., Lee, S., 2009. Data concealment and detection in Microsoft Office 2007 files. *Digital Investigation*, 5(3-4), pp.104–114.
- Parsonage, H., 2012. Under my thumbs. Available at: <http://computerforensics.parsonage.co.uk/downloads/UnderMyThumbs.pdf>. [Accessed June 14, 2012]
- Penhallurick, M., 2005. Methodologies for the use of VMware to boot cloned/mounted subject hard disk images. *Digital Investigation*, 2(3), pp.209–222.
- Police and Criminal Evidence Act 1984 (PACE) (1984 c. 60)
- Pomeranz, D., Shemesh, M. & Ben-Shahar, O., 2011. A fully automated greedy square jigsaw puzzle solver. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. pp. 9–16.
- R v Porter (Ross Warwick) [2006] EWCA Crim 560

- Ramjohn, M., Landa, J., 2009. *Unlocking Evidence*, Hodder Education.
- Randers-Pehrson, G., 1999. PNG (Portable Network Graphics), Specification Version 1.2. Available at: <http://www.libpng.org/pub/png/spec/1.2/png-1.2.pdf>. [Accessed April 28, 2011].
- Reed, C., 2005. The Admissibility and Authentication of Computer Evidence - A Confusion of Issues. In *5th BILETA Conference British and Irish Legal Technology Association*. pp. 1–9.
- Reith, M., Carr, C., Gunsch, G., 2002. An examination of digital forensic models. *International Journal of Digital Evidence*, 1(3), pp.1–12.
- Richard, G., Roussev, V., 2005. Scalpel: A frugal, high performance file carver. *Digital Forensics Research Workshop*. New Orleans, LA.
- Roukine, M., 2012. Vinetto. Available at: <http://vinetto.sourceforge.net/docs.html> [Accessed May 17, 2012].
- Ruhnka, J., Bagby, J.W., 2008. Forensic Implications of Metadata in Electronic Files. *CPA Journal*, 78(6), p.68.
- Russell, S., Norvig, P., 2003. *Artificial Intelligence: A Modern Approach (International Edition)*, Pearson.
- Sammes, T., Jenkinson, B., 2007. *Forensic Computing: A Practitioner's Guide*. Springer
- Sanderson, P., 2006. Mass image classification. *Digital Investigation*, 3(4), pp.190–195.
- Sansurooah, K., 2006. Taxonomy of computer forensics methodologies and procedures for digital evidence seizure. In *Australian Digital Forensics Conference*. p. 32.
- Sencar, H. Memon, N, 2009. Identification and recovery of JPEG files with missing fragments. *Digital Investigation*, 6(Supp.), pp.S88–S98.
- Shanmugasundaram, K., 2003. Automatic reassembly of document fragments via context based statistical models. *IEEE Proceedings Computer Security Applications Conference*.
- Smith, C., 2008. The Facts About Computers - Seven Common Misconceptions. *Ezine Articles*. Available at: <http://ezinearticles.com/?The-Facts-About-Computers---Seven-Common-Misconceptions&id=1275149>. [Accessed September 09, 2011]
- Stephenson, P., 2003. A comprehensive approach to digital incident investigation. *Information Security Technical Report*, 8(2), pp.42–54.

- Stone-kaplan, K., Roter, M., 2003. Date, Time, and Time Zone Examination. *Guidance Software Whitepaper*.
- Taylor, C., 2012. *Law Express: Evidence (revision Guide)*, Pearson.
- Thomas, D., Forcht, K., 2004. Legal methods of using computer forensics techniques for computer crime analysis and investigation. *Issues in Information System*, 5(2), pp.692–698.
- Truecrypt, 2013. Truecrypt. Available at: <http://www.truecrypt.org/> [Accessed January 9, 2013].
- Twitter, 2013. Twitter. Available at: www.twitter.com [Accessed January 2, 2013].
- Ubuntu: Lucid, 2010. Ubuntu. Available at: www.ubuntu.com [Accessed June 30, 2010].
- Veenman, C., 2007. Statistical disk cluster classification for file carving. *Information Assurance and Security Symposium*. pp.393–398.
- Vmware, 2011. VMWare Workstation. Available at: www.vmware.com. [Accessed May 27, 2011]
- Vosburgh; United States v. Vosburgh, 602 F.3d 512, 526 (3d Cir. 2010)
- Waghorn, T., 2003. Testing the Date Maintenance of the File Allocation Table File System. *1st Australian Computer, Network & Information Forensics Conference*. Perth, Australia.
- Weeks, J., 1998. Videotechnology.com. Available at: <http://www.videotechnology.com/jpeg/j1.html> [Accessed August 26, 2011].
- Weil, M., 2002. Dynamic time & date stamp analysis. *International Journal of Digital Evidence*, 1(2).
- White, P.C., 2010. *Crime Scene to Court*, Royal Society of Chemistry.
- Willis, A.R., Cooper, D.B., 2008. Computational reconstruction of ancient artifacts. *Signal Processing Magazine, IEEE*, 25(4), pp.65–83.
- Woodward, A., 2006. BitLocker-the end of digital forensics? *Australian Digital Forensics Conference*.
- Youtube, 2013. YouTube. Available at: <http://www.youtube.com/> [Accessed March 29, 2013].

Zetterstrom, H., 2002. Deleting Sensitive Information Why hitting delete isn't enough.
SANS Institute InfoSec Reading Room.

APPENDICES

Appendix A Thumbnail Cache File Structures

A.1 Windows 7 Thumbnail Cache Structure

A.1.1 IDX File Header Structure

```
typedef struct {  
    CHAR FileSignature[4]; // 4 bytes  
    DWORD Version; // 4 bytes  
    DWORD CacheNo; // 4 bytes  
    DWORD HeaderSize; // 4 bytes  
    DWORD NumRecordsUsed; // 4 bytes  
    DWORD HashTableSize; // 4 bytes  
} ThumbcacheIDXHeader7;
```

The version number is 20 for Windows Vista and 21 for Windows 7
The CacheNo is 0 for 32, 1 for 96, 2 for 256, 3 for 1024 and 4 for sr

A.1.2 IDX Record Structure

```
typedef struct {  
    UQUAD HashTableKey<format=hex>; // 8 bytes  
    UINT flags; // 4 bytes  
    UINT Position32<format=hex>; // 4 bytes  
    UINT Position96<format=hex>; // 4 bytes  
    UINT Position256<format=hex>; // 4 bytes  
    UINT Position1024<format=hex>; // 4 bytes  
    UINT Positions<format=hex>; // 4 bytes  
} ThumbcacheIDXRecord7;
```

The record structure repeats throughout the idx file; the number of records present is always a multiple of 101. If a position field contains FF FFFFFF, then there is no associated image of that size for the record. If the position is 00 00 00 00 then the whole record should be empty.

A.1.3 Image File Header

```
typedef struct {  
    CHAR FileSignature[4]; // 4 bytes - CMMM  
    DWORD Version; // 4 bytes  
    DWORD CacheNo; // 4 bytes  
    DWORD HeaderSize; // 4 bytes  
    DWORD PositionLastSubRecord; // 4 bytes  
    DWORD EntryCount; // 4 bytes  
} ThumbcacheImageHeader7;
```

The version number is 20 for Windows Vista and 21 for Windows 7
The CacheNo is 0 for 32, 1 for 96, 2 for 256, 3 for 1024 and 4 for sr

A.1.4 Image File Subrecords

```
typedef struct {  
    CHAR SubRecordSignature[4]; // 4 bytes - CMMM  
    DWORD sizeHeaderAndData; // 4 bytes  
    UQUAD HashTableKey<format=hex>; // 8 bytes  
    DWORD SizeFileID; // 4 bytes  
    DWORD Padsizes; // 4 bytes  
    DWORD ImageSize; // 4 bytes  
    DWORD Empty; // 4 bytes  
    UQUAD DataChecksum; // 8 bytes  
    UQUAD HeaderChecksum; // 8 bytes  
    CHAR FileID[SizeFileID]; // size of SizeFileID value  
    CHAR Padding[Padsizes]; // size of Padsizes value  
} ThumbcacheImageSubRecord7;  
//Followed by image of size [ImageSize]
```


A.2 Linux Thumbnail Cache Structure

Each subrecord consists of the following chunks:

A header signature

IDHR (Header chunk)

SBIT

tEXt

IDAT

IEND

The chunks follow the PNG Specification [Randers-Pehrson, 1999]. A chunk consists of:

```
typedef struct {  
    DWORD DataSize; // 4 bytes  
    CHAR ChunkType[4]; // 4 bytes  
    CHAR Data[DataSize]; // size of DataSize value  
    CHAR CRCValue[4]; // 4 bytes  
} LinuxThumbnailChunk;
```


Appendix B Bayesian probability tables

B.1 SR Position either 0xFFFFFFFF or 0x00000000

	Probability
True	0.014847
False	0.985153

B.2 Valid Thumbnail Cache ID (TCID) values

	Probability
True	0.0002
False	0.9998

B.3 Fragment contains CMMM

	Probability
True	0.00101
False	0.99899

B.4 Fragment contains a Start of Image (SOI) marker

	Probability
True	0.01553
False	0.98447

B.5 Fragment starts with “IMMM”

	Probability
True	0.000000001
False	0.999999999

B.6 Fragment contains an IDAT marker

	Probability
True	0.004407
False	0.995593

B.7 Fragment contains a TEXT marker

	Probability
True	0.001646
False	0.998354

B.8 Fragment contains the ASCII string “thumb”

	Probability
True	0.00317
False	0.96828

B.9 Fragment stored in valid PNG chunks

	Probability
True	0.06726
False	0.93274

B.10 Each byte frequency is less than 2 %

	Probability
True	0.0549
False	0.94509

B.11 Valid JPEG start of sequence markers

	Probability
True	0.062757
False	0.937243

B.12 High text frequency (Ascii)

	Probability
True	0.34485
False	0.65515

B.13 High text frequency (Unicode)

	Probability
True	0.09664
False	0.90336

B.14 Known header signature NOT thumbnail cache

	Probability
True	0.985742
False	0.014258

B.15 32 byte record structure

SR position either 0xFFFFFFFF or 0x00000000	Valid TCID values	Probability
True	True	1
True	False	0
False	True	0
False	False	0

B.16 Subrecord header structure

Valid TCID values	Fragment contains CMMM	Probability
True	True	0.965548
True	False	0.025903
False	True	0.008549
False	False	0

B.17 H1

32 byte record structure	Fragment starts with "IMMM"	Probability
True	True	0.0916
True	False	0.9084
False	True	0
False	False	0

B.18 H2

Subrecord header structure	Fragment starts with CMMM	Start of Image marker	Probability
True	True	True	0.07407
True	True	False	0.018518
True	False	True	0.876242
True	False	False	0.0307
False	True	True	0
False	True	False	0
False	False	True	0.00047
False	False	False	0

B.19 H3

Each byte frequency is less than 2 %	Valid JPEG start of sequence markers	Probability
True	True	0.821516
True	False	0.05057
False	True	0.12202
False	False	0.005894

B.20 H4

IDAT marker	TEXT marker	Ascii string “thumb”	Valid PNG chunks	Probability
True	True	True	True	0.16807
True	True	True	False	0
True	True	False	True	0
True	True	False	False	0
True	False	True	True	0
True	False	True	False	0
True	False	False	True	0.82415
True	False	False	False	0
False	True	True	True	0.00778
False	True	True	False	0
False	True	False	True	0
False	True	False	False	0
False	False	True	True	0
False	False	True	False	0
False	False	False	True	0
False	False	False	False	0

B.21 H5

Stored in valid PNG chunks	Each byte frequency is less than 2 %	Probability
True	True	0.77184
True	False	0.22816
False	True	0
False	False	0

B.22 H6

High text frequency (ASCII)	High text frequency (Unicode)	Known header signature NOT thumbnail cache	Probability
True	True	True	0.0000001
True	True	False	0.0000001
True	False	True	0.049962
True	False	False	0.10672
False	True	True	0.020005
False	True	False	0.0997
False	False	True	0.0524279
False	False	False	0.6711849

Appendix C Peer Reviewed Publications

C.1

Morris, S.; Chivers, H.; 2010. A comparative study of the structure and behaviour of the operating system thumbnail caches used in Kubuntu and Ubuntu (9 . 10 and 10 . 04). In *Cybercrime Forensics Education and Training*. Canterbury, UK.

A comparative study of the structure and behaviour of the operating system thumbnail caches used in Kubuntu and Ubuntu (9.10 and 10.04)

Sarah Morris¹, Howard Chivers²

Centre for Forensic Computing,
Cranfield University, Shrivenham, SN6 8LA, UK

¹S.L.Morris@cranfield.ac.uk

²H.Chivers@cranfield.ac.uk

Abstract

Browsing directories in thumbnail mode can assist the user in locating relevant documents quickly by providing a graphical representation of each file. Whilst thumbnail images can assist the user, they can be resource intensive to generate; therefore operating systems generally cache these images, along with associated metadata, to prevent unnecessary rendering. Thumbnail caches can provide a variety of information, including file paths and images of documents; however it is necessary to understand the user activity which resulted in the artefacts being created to understand their forensic significance.

This research used baseline versions of Ubuntu and Kubuntu (both versions 9.10 and 10.04) in virtual machines to determine the effects of a variety of user actions on the information stored in the thumbnail caches. A series of experiments were conducted to identify the structures used to store artefacts both in the thumbnail cache and any related file throughout the system, as well as determining the meaning of each artefact. Each experiment was performed on both operating systems and mimicked a variety of typical user behaviours, such as moving a file or accessing a USB stick.

Whilst both thumbnail caches implement the same structure for storing data, the user behaviour which leads to artefacts being stored in the thumbnail caches differs considerably between the two operating systems. Other information about user activity can be deduced from the thumbnail cache itself; for example, Kubuntu uses an RGB format for items cached without the directory being viewed and uses an RGBA format for standard record creation. This paper also identifies the user

activity which led to artefacts being recovered and discusses the strengths and weaknesses of the thumbnail caches.

This research shows that similar artefacts from two closely related operating systems may nevertheless suggest different types of user activity, and hence have a different forensic significance.

Introduction

Thumbnail images are used in a variety of applications and operating systems to enable the identification of various source files; for example when browsing directories. Whilst the thumbnail images enhance the usability of applications, they can be resource intensive to generate, particularly if a directory contains multiple files which require the generation of thumbnails. A thumbnail cache allows images to be stored to prevent unnecessary rendering, therefore requiring less resources. Such caches are of particular significance to forensic examiners as they can contain metadata, a thumbnail image of the original file and information about the user's activity.

There is no standard implementation of a thumbnail cache, therefore software applications and operating systems each use the basic concept in their own way. This can be a source of confusion for forensic examiners as thumbnail caches which look structurally similar may not represent the same activity, and may therefore have a different evidential significance.

This paper shows the structural similarity of the thumbnail caches implemented in Ubuntu and Kubuntu and the differences in the user activity which lead to their generation. It also highlights the requirement to educate forensic examiners to identify the context of any evidence identified. The experiments were performed using standard ISO images of Ubuntu and Kubuntu downloaded from their WebPages.

This paper discusses the operating system thumbnail cache used by Ubuntu and Kubuntu (9.10 and 10.04) and is structured as follows. Firstly Section 2 provides relevant background information on operating system thumbnail caches and their forensic significance. In Section 3, the structures of the thumbnail caches are described. Section 4 identifies the user actions which cause information to be present within the thumbnail cache. Section 5 compares the thumbnail caches used in Ubuntu and Kubuntu. Finally Section 6 concludes the paper and discusses the possibilities for future research.

2.0 Related Work

This section provides relevant background information and explains why thumbnail caches are important in digital investigations.

Paivio [1] has shown that the use of images when searching can improve a user's ability to identify information. This concept has been integrated into computing by the addition of graphical user interfaces and thumbnail images. Ubuntu uses Nautilus, and Kubuntu uses Dolphin, for thumbnail image creation and storage [2]. Thumbnail images of a variety of file types are stored in a centralised directory, which can be accessed by any application. Open Office.org [3] created a script which allows thumbnails for open office documents to be created and stored, increasing the variety of images found in the thumbnail cache. Whilst this script is currently not installed by default on Ubuntu or Kubuntu, it is available to download on the Open Office website.

Whilst thumbnail caches are implemented by a variety of software and operating systems, there is no standard structure. This can lead to a variety of forensic artefacts being available, each created by a different user activity, making it difficult for a forensic examiner to establish the value of any artefacts they recover. A Free Desktop specification for thumbnail caches was created in 2004 [4], it aims for each system to implement a single cache which can be utilised by both the operating system and applications. File managers such as Konqueror and Nautilus base their thumbnail caches on this specification. The specification does not specify the user activity which leads to the thumbnail cache being manipulated. Therefore whilst both file managers may attempt to implement the specification the user activity involved and therefore the forensic significance of the artefacts may be significantly different in each case.

Whilst images form a significant part of the cache, the metadata may also be useful to investigators. In the case of *R v Ross Warwick Porter* [5], the defendant was accused of possessing indecent images of children. A forensic examination of his computer showed a piece of software with its own thumbnail cache had been used to store graphical images by the defendant; however the images had been deleted, leaving only metadata in the cache [6]. At appeal the defendant's conviction was overturned on the grounds he was not in possession of the indecent images as he did not have the means or knowledge to retrieve them from slack space. Despite the overturning of the conviction this case still highlights the role metadata from the cache can have in ascertaining whether the files are known to the defendant, had the files not been deleted but moved to another storage media it would have shown a relationship between the devices and may have assisted a conviction.

Modification, access and creation (MAC) times of a file can give the investigator an insight into how the files arrived in their current location. Chow proposed seven rules based on these times, which assist in building event timelines [7]. The rules elaborate on the relationships between MAC times and the behaviour of the file. Chow reviews the value of the thumbnail cache, in relation to these rules, for establishing information relating to the viewing of such files (Scenario 9 in [7]). This suggests that thumbnail caches can be used to assist in building a timeline of activity on a machine.

This section has identified the use of thumbnail images in Ubuntu and Kubuntu, and identified their potential significance in digital investigations. Thumbnail images are used to improve a user's identification of material, in order to minimise the use of system resources the images are usually cached. The thumbnail caches also store metadata, which can be useful in establishing event timelines. Entries in a thumbnail cache can be useful in establishing user activity and forming a relationship with files.

3.0 Methodology

To establish the structure and behaviour of the thumbnail caches, four baseline virtual machines were created. Each virtual machine had a six gigabyte hard drive; the operating systems Ubuntu and Kubuntu were installed using ISO images downloaded from Ubuntu's website. To mimic a standard user's setup, the default options were selected during installation and no additional software was loaded onto the machine. The baseline machines were then cloned, for each experiment; this enabled the results to be compared with a fresh installation.

In some cases, it was necessary to compare the results of several experiments, such as when looking at the modification of data within the thumbnail cache. All comparisons were conducted in WinHex, with the Virtual disk file being 'interpreted as a disk' in order to ascertain the structure of the operating system.

To investigate the structure of the thumbnail caches, various files were loaded onto the computer using a mixture of methods: USB sticks, CDs, downloading from the Internet and creation of a file using pre-installed software. The records created in the cache were then analysed and compared with both the Free Desktop Thumbnail Cache and PNG specifications. The data collected from these experiments also assisted in establishing how information is added to the thumbnail cache. To investigate the modification of data within the thumbnail cache, source files were altered, renamed and given new Uniform Resource Identifiers (URI). To find out when information is deleted, source files were deleted from the virtual disk. Also USB devices were not plugged into the system for a period of thirty days to establish if this would cause the thumbnail cache to remove unused data.

4.0 The structure of thumbnail cache information

In Ubuntu and Kubuntu, the default setting within the file browser is to display thumbnail images of files. However not all file types display thumbnail images by default, for example office documents currently do not. For files such as image formats, the thumbnails shown in the browser are of a large enough size to aid file identification. Whilst generally thumbnail images are thought to be a graphical interpretation of an image, or the first page of a document, thumbnails of text files use the first few characters from the document. The thumbnail images of text documents are not stored in the thumbnail cache and are generated every time the

directory is viewed. Figure 1 shows a file browser and the thumbnail images generated when using Ubuntu's default settings.

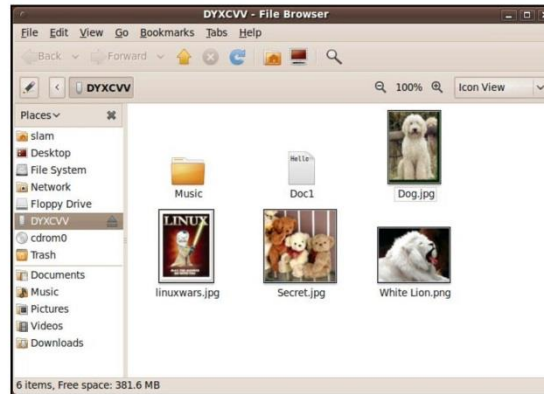


Figure 1: The default view in Ubuntu, when opening a directory in a file browser

Whilst the specific methods of thumbnail image generation are beyond the scope of this paper, it is worth noting that most thumbnail images are created through the default thumbnail process. In Ubuntu this is known as the Gnome Thumbnail Factory; other software can use its own thumbnail creation process and use the .thumbnails cache.

Structurally, both Ubuntu and Kubuntu have implemented elements from the Free Desktop specification for their thumbnail caches [4]. The following sections provide an overview of the structures present in both the Ubuntu and Kubuntu thumbnail caches. Firstly the overall structure of the thumbnail cache is described; this is followed by an examination of the thumbnail cache record structure.

4.1 .thumbnails

The main thumbnail cache structure is a hidden directory in the user's home folder, named '.thumbnails'. This directory behaves like any other hidden directory in the operating system. Figure 2 shows the directory structure for the thumbnail cache. Thumbnail records are not stored in the .thumbnails directory itself, but in one of three sub-directories: fail, large and normal. There is generally only one thumbnail cache present on the system which can be accessed by any software as required

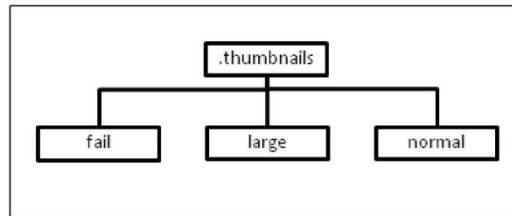


Figure 2: The structure of thumbnail caches in Ubuntu and Kubuntu (9.10 and 10.04)

4.2 thumbnail cache records

A record within the thumbnail cache can be defined as a complete set of data relating to an original file. In the case of Ubuntu and Kubuntu, a thumbnail cache record can be defined as the set of files located in `.thumbnails` or any of its subdirectories which have the same filename and extension as each other. A sub-record can be defined as a file stored in the thumbnail cache, which contains metadata about an original file; sub-records may contain a thumbnail image. In Figure 3, shows the thumbnail sub-records for the file `z2nxaujt.jpg`, together these sub-records form a complete record of thumbnail cache information about the source file.

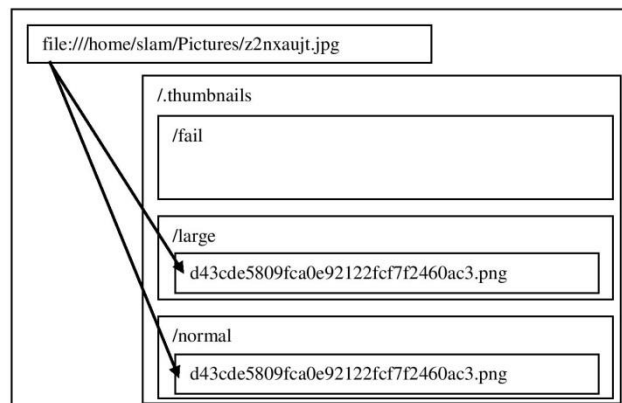


Figure 3: A diagram showing the thumbnail cache sub-records which make up the complete record for the file `z2nxaujt.jpg`

In operating systems such as Windows 7, a centralised index file, thumbcache_idx.db is located in the thumbnail cache to provide a single record for each original file. The records within the index contain pointers to the location of the various image sizes and associated metadata for each original file. In Ubuntu and Kubuntu, there is no centralised index for each record, and therefore the only way of identifying a complete record is to search for sub-records which have the same filename and path as each other, and where the filename is an exact MD5 hash of the URI of the source file.

Each sub-record has a PNG file extension; the filename is an MD5 hash of the URI of the original source file. A thumbnail cache sub-record is stored in the PNG format, which is open standard. The structure can be broken down into a series of 'chunks', each of which has a defined structure. Each record generally consists of a header and up to five different types of data chunks; IDHR, SBIT, TEXT, IDAT, IEND. Each chunk consists of four bytes denoting the size of the data stored, four bytes denoting the type of the chunk, a data section, and a four byte CRC value.

The header of a sub-record is in the standard PNG format, showing the file consists of a single image. This is followed by the SBIT chunk, which defines the significant bits of the colours; generally this is set to 8bit, non-interlaced, with full alpha transparency.

The TEXT chunks contain the metadata on the original file and the thumbnail image; there are usually several TEXT chunks in each file. Each TEXT chunk holds a single piece of metadata in the form of a key and a value, the key specifies the meaning of the value. The keys may be inbuilt into the PNG format, specific for the thumbnail cache or for the file type. Thumbnail cache specific keys start with 'Thumb', all keys are written in Ascii, and are easy for an forensic examiner to interpret; for example 'Thumb:MTime', is the key for the modification date of the source file. The modification time and URI of the source file are generally present in all sub-records. Other information, such as the dimensions of the original file and the number of pages it contained may also be present.

The thumbnail image itself is stored within one or more IDAT chunks, which are compliant with the PNG specification. Finally each sub-record file ends with an IEND chunk, which denotes the end of a file.

4.3 The normal and large directories

The normal and large directories contain sub-records which the thumbnail creation methods have generated successfully. The normal directory is most frequently populated and contains sub-records whose thumbnail image is no larger than 128x128 pixels. In the large directory, the thumbnail image may be up to 256x256 pixels. Whilst the sizes for the directories have been defined, individual thumbnail creation methods may not follow the specification and therefore anomalies may occur.

4.4 The fail directory

The fail directory contains incomplete sub-records, where the generation was unsuccessful. The sub-record may have failed for a variety of reasons, for example, the source file contains an error. The fail directory provides a way of storing information on files which were not successfully added to the cache to prevent the system attempting to create the thumbnail image again, saving resources. These records may contain some useful metadata; however it is necessary to establish the reliability of the artefacts within the fail directory.

4.5 Conclusion

This section has described the structure of the thumbnail caches used in the Ubuntu and Kubuntu operating systems, which are contained within the hidden directory `.thumbnails`. This directory contains three sub-directories, which allow the storage of multiple sizes of thumbnail images and information on files which have unsuccessfully attempted to generate data for the thumbnail cache. Each original source file may have multiple sub-records within the cache, each stored in the PNG format. Each sub-record contains metadata about both the sub-record and the original source file and it may also contain an image. Whilst it is outside the scope of this paper, there are a number of interesting artefacts throughout the Ubuntu and Kubuntu file systems, which are created as a result of actions on the thumbnail cache which may provide information to a forensic examiner; for example the directory listings.

This section has identified the structures used in the Ubuntu and Kubuntu thumbnail caches as being the same, the next section shows the user activity which leads to artefacts being present in the cache, therefore allowing a context to be placed on the evidence.

5.0 The behaviour of the thumbnail cache

By default, when Ubuntu and Kubuntu are installed, the thumbnail cache directory does not exist. Instead, the directory is created when the first call to create a thumbnail image is made. This section shows the primary methods used to alter the information within the thumbnail cache, and discusses the resulting artefacts.

5.1 Adding a sub-record

When a thumbnail image is requested, the system first checks if the image is already being stored in the thumbnail cache. In order for this check to occur, the system attempts to form a relationship between the original source file and a sub-record within the thumbnail cache. To establish if a relationship occurs, an MD5 hash of the source files URI is calculated, which is used as a filename. An extension of 'PNG' is added to the filename which is then compared with sub-records in the thumbnail cache to look for an exact match. If a record is found with a matching file name and extension, further checks are performed to ascertain whether the sub-record requires modification. These checks are discussed further in 5.2 Modifying a sub-record.

If no matching sub-record is identified, then a new one is created, using the structure identified in Section 3, and the thumbnail creation method for that file type. When the sub-record is initially being written, a six character random extension is appended to the end of the filename. This extra extension is removed upon the successful creation of the sub-record and ensures that only complete records are accessed through the thumbnail cache. If a sub-record is not successfully completed, the extra extension is removed, and the file is moved to the fail directory. When performing an analysis it is possible a forensic examiner would find several entries in the directory with the file extension; occasionally these can contain partial sub-records.

5.2 Modifying a sub-record

When a request is made to find a thumbnail image for a source file and a match is found, a further check must be performed. This is to ensure the thumbnail image is an accurate representation of the current state of the source file. The last modification time of the source file is checked with the thumbnail record, to establish if they are the same. Checking the dates are the same ensures the image is updated if the file has been modified since the sub-record was last updated.

For example, if a file with the same filename and extension as an existing file is added to a directory, the existing file may be overwritten. Since the new file has the same URI, it would generate the same MD5 hash (see 4.2), therefore the thumbnail cache entries for that URI would be outdated. Since the new file may have a modification time that is greater than the previous file, checking the time is an exact match helps to ensure the correct record is displayed.

By default, no checks, other than those in the PNG standard, are performed on the IDAT chunks of a sub-record to ensure they are an accurate representation of the file. Should a file require modification, the entire record is re-written, with the additional six character extension appended to the end, which is removed after successful creation.

5.3 Deleting a record

If a record is no longer required, a call can be made to the system to delete a record. As each sub-record is a unique file, their deletion is handled the same as any other file. Traces of sub-records can still be found, which may enable a relationship with a source file to be formed.

5.4 Conclusion

This section has shown how the thumbnail cache structures outlined in Section 3 can occur in the system. To find a sub-record for a file, the MD5 hash of the source file's URI, with the extension PNG, is searched for. If an exact match cannot be found, a new sub-record is created. If a match is found, the last modification time is checked to establish if the time is an exact match with that of the source file. An

exact match of time is used as there are circumstances where the last modification date of the file may be greater than the one stored in the sub-record. Since any addition or modification to the cache involves writing an entire sub-record, the system appends a six character extension to the filename to ensure it can only be accessed when it is successfully created.

6.0 A comparative review of Ubuntu and Kubuntu thumbnail caches

In the previous two sections, the structure of the thumbnail caches were identified, together with structural changes to the information within the cache when information is added, modified or deleted. This section shows the user activity, which can cause the structures previously identified to be present in the cache and how they differ between the two operating systems.

6.1 Adding and modifying a sub-record

For thumbnail creation, both operating systems operate a queue, into which potential source files can be added. Once a source file reaches the front of the thumbnail creation queue, the system attempts to locate a matching sub-record in the appropriate directory of the thumbnail cache. If a matching file cannot be found, or the URIs match but the modification times do not, the system creates a new sub-record using the appropriate generation method for that file type. This section identifies when source files are added to the thumbnail generation queue.

In Ubuntu, sub-records are generally added to the normal directory, with each potential source file viewed in the file browser added to the thumbnail queue. Sub-records may not be created or updated if a directory contains a substantial number of files, which are potential thumbnail cache source files, and the directory is no longer being viewed in the file browser. This is because the files are taken out of the thumbnail cache queue when the directory is no longer being viewed. By default, potential thumbnail cache source files for Ubuntu are standard image formats and PDFs.

GIMP, a pre-installed graphics program can initiate the creation and update of sub-records to the normal directory when the user selects 'open image, and potential thumbnail cache source files are viewed as directories are being viewed. F-Spot also causes sub-records to be updated and created when importing files; however these sub-records are only added to the large directory.

In Kubuntu, both the large and normal directories are frequently used; therefore a greater amount of information is available to a forensic examiner, than that provided by Ubuntu. When the file browser is open, any potential source files are added to the thumbnail queue for both the large and normal directories. If there is only a single potential source file in the directory, only a large record is added on the first viewing. If within the file browser there are directories that contain

potential source files, these may also be added to the thumbnail queue; this is discussed further in Section 6.4. The information panel within the file browser is the reason for the creation of sub-records in the large directory, which enables the file to be previewed in a larger size.

Gwenview adds files to the thumbnail queue, when browsing directories in the main window of the application; however it only uses the normal thumbnail cache directory, and therefore the large directory is not modified.

If within Kubuntu, the large thumbnail cache directory is viewed, the files within it are added to the thumbnail queue for the creation and modification of sub-records within the normal directory. These sub-records do not conform to the standard structure; instead they use the same structure as the partial sub-records which are discussed in Section 6.4. Potential thumbnail source files in Kubuntu by default are only image formats. Interestingly, thumbnails for text files are not available in the file browser. If a text file is selected within the file browser, a thumbnail image is rendered in the information panel, but this is not stored in the cache.

In both operating systems, potential source files viewed on the desktop are added to the thumbnail queue for sub-records to be added or modified in the normal directory only. However if the desktop is viewed in the file browser in Kubuntu, sub-records may also be created or modified in the large directory.

6.3 Deleting a record

From the tests conducted on Ubuntu and Kubuntu, no evidence of the deletion of sub-records was recorded. Tests involved deleting and moving files on the hard drive and external storage media. A test was also carried out where a USB stick was mounted on the virtual machines, to enable the creation of thumbnail cache records and then the device was dismounted. The machine was then left for thirty days to ascertain if unused sub-records would be removed, however no entries in the thumbnail cache were altered.

6.4 Anomalies

Within Kubuntu, it is interesting to note that there are circumstances where thumbnail sub-records are unexpectedly created. Partial sub-records within the cache are automatically generated for a selection of files in the first level sub-directories of a directory being viewed in the file browser. An example is shown in Figure 4, which shows the Downloads directory being viewed in the file browser. When this directory is opened, each sub-directory is represented by a series of small thumbnails, such as those shown for 'Test Folder'. These small thumbnails represent files present in the sub-directory, and to create them, the system creates partial thumbnails of files within the sub-directory. If the sub-directory is hovered over with the mouse, the mini-thumbnails rotate through any files within the sub-directory that have thumbnail generation methods.

Therefore from each first level sub-directory in a browsed folder, there may be sub-records created for up to the number of files which have thumbnail generation

methods, even though the user has not browsed the sub-directory. Interestingly, partial thumbnail sub-records have two properties which allow them to be distinguished from sub-records created using standard methods. Firstly, there is a noticeable lack of metadata within the PNG, including no information on the last modified time and URI for the source file. This means there are two main methods of forming a relationship between a partial sub-record and the source file; the image stored in the sub-record and the MD5 hash of the URI which is present in the sub-records filename. The second difference in partial sub-records is noticeable in the IDAT chunks, where upon comparison with a standard sub-record entry for the same file, it is noticeably shorter as the alpha values are missing.

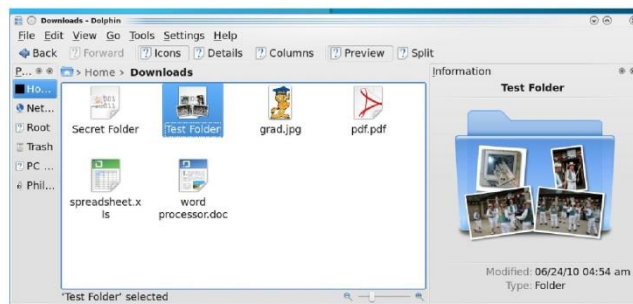


Figure 4: The directory 'Downloads' being viewed in the file browser within Kubuntu



Figure 5: The directory 'Misc Images' being viewed in the file browser within Ubuntu

By default, when viewing directories in Ubuntu, sub-directories are represented by folders, as can be seen in Figure 5. Upon selection of a sub-directory, there is no

visual change in the thumbnail image, and no partial sub-records have been stored in the thumbnail cache.

In Ubuntu a curiosity was discovered when attempting to alter the thumbnail image stored in the IDAT section of a sub-record. In order to successfully fool the system into displaying an incorrect thumbnail, all other metadata within the sub-record must be accurate. This includes the last modification time, as discussed previously, which must be exactly the same as the source file, or the sub-record is rewritten. It is also important to ensure that each part of the chunks within the sub-record are accurate, for example, if an existing sub-record was being altered it would be necessary to alter the size of each chunk and the CRC data to ensure they matched the new information. If the sub-record uses a valid PNG structure, and the last modification date is equal to the source file then, the new image placed in the IDAT section would be viewed instead of a representation of the actual file.

Table 1: A summary of the user activity which may lead to sub-records being present in the thumbnail cache

		Ubuntu	Kubuntu
Sub-records in /fail		- Sub-record creation halted before creation - The file contained an error - An error occurred during sub-record generation	
Sub-records in /normal	Standard sub-record	- Viewing directories in the file browser - Viewing a directory in GIMP using 'open image'	- Viewing directories in the file browser - Viewing directories in Gwenview
	Partial sub-record		- Viewing a folder for a directory in the file browser - Viewing records in the large directory
Sub-records in /large		- Viewing directories in F-spot	- Viewing directories in the file browser
Sub-records in any directory with an additional six character extension		- Sub-records which are in the process of being generated - The remains of a generated sub-record	

However if this 'tampered' record is selected, a new thumbnail image is shown which accurately represents the file. The tampered record is not, however overwritten in the thumbnail cache. Instead, the accurate image is generated when

required, by the system and does not appear to be stored anywhere. If the browser window is refreshed or reopened, the tampered thumbnail is still displayed, until the file is selected.

6.5 Conclusion

This section has shown that the user behaviour which leads to information within the thumbnail cache being altered differ among the two operating systems.

Table 1 shows a breakdown of the user behaviour, which may be responsible for artefacts being present in the thumbnail cache, based on the default packages installed during installation. This section has also shown that Kubuntu makes significantly more thumbnail sub-records, thereby providing a greater range of evidence to a forensic examiner.

7.0 Conclusions

This paper has shown that despite the structure of the thumbnail caches implemented in Ubuntu and Kubuntu being the same, the user behaviour which leads to the creation of artefacts can be significantly different. This highlights the need for forensic education to take careful account of the context of an artefact and not just the structure.

8.0 References

- 1 Paivio A, Begg I (1974). Pictures and Words in Visual Search. *Memory & Cognition* 2, 3, 515-521, 1974.
- 2 Ubuntu: Lucid, www.ubuntu-guide.org/ (visited June 2010)
- 3 Open Office.org – The Free and Open Productivity Suite, www.openoffice.org (visited June 2010)
- 4 Thumbnail Managing Standard, <http://people.freedesktop.org/~vuntz/thumbnail-spec-cache> (visited June 2010)
- 5 R v Porter (Ross Warwick) [2006] EWCA Crim 560
- 6 The Spanner Trust, http://www.spannertrust.org/documents/R_v_Porter.pdf (visited June 2010)
- 7 Chow K, Law F, Kwan M, Lai P (2007). The Rules of Time on NTFS FileSystem, *Systematic Approaches to Digital Forensic Engineering*

C.2

Morris, S.; Chivers, H.; 2011a. An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache. In *Cyberforensics*. Glasgow, UK.

An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache

Sarah Morris¹, Howard Chivers²

Centre for Forensic Computing and Security

Cranfield University, Shrivenham, SN6 8LA

S.L.Morris@cranfield.ac.uk, H.Chivers@cranfield.ac.uk

Abstract. Operating systems such as Windows 7 implement a thumbnail cache structure to store visual thumbnails and associated metadata. There is no standard implementation of a thumbnail cache or its functions, which has led developers to implement their own structures and behaviour. The artefacts present within a thumbnail cache are of interest to a forensic analyst as they can provide information on files within the system which may be of use to the investigation. This research investigates the structure and behaviour of the thumbnail cache implemented in Windows 7 and shows that as well as storing information relating to visual thumbnails the cache also stores the names of networked computers, GUIDs relating to system artefacts and allocated drive letter information. It also shows that due to the behaviour of the cache, information such as records relating to files which are no longer on the system may be available, proving interesting forensic evidence.

Keywords: thumbnail cache, windows 7, forensic computing.

1 Introduction

Microsoft operating systems are installed on a significant proportion of user machines. Due to the likelihood of encountering a Windows based operating system there is significant interest in finding and understanding relevant artefacts within the forensic community. Therefore experiments investigating changes to the thumbnail cache structure and behaviour are interesting and relevant.

This paper shows the structure and behaviour of the operating system thumbnail cache used in Windows 7; understanding the structure can assist in retrieving meaningful artefacts, whilst understanding the behaviour allows the artefacts to be placed in context. This paper begins with an overview of related work and the

methodology employed during this research; followed by a description of the structures implemented within the thumbnail cache. Finally the effect user behaviour has on the thumbnail cache is investigated.

2 Related Work

In Windows XP, thumbnail caches occurred throughout the system, located in every directory which contains potential visual thumbnail source files and has been viewed in thumbnail mode by a user. These thumbnail caches were stored in hidden files named 'thumbs.db' and contain interesting forensic artefacts such as the file name of the original file [1].

The release of Windows Vista prompted speculation from Forensic analysts about the impact of the new Operating system on investigations [2]. One interesting change was the move away from the 'thumbs.db' files, which were present in individual directories, to each user having a single centralised thumbnail cache. The release of Windows 7 therefore prompts the question of identifying further changes; in fact, as shown below, these changes are limited.

Whilst the function of a thumbnail cache has been defined, the structure varies considerably between operating systems. This means that each thumbnail cache structure will have a different forensic value and the data stored within it may not represent the same series of events [3].

Users may not be aware of the thumbnail cache; the media openly discussed the potential privacy violations of its existence in the case of Vosburgh [4, 5]. During this case the defence put forward the argument that thumbs.db were hidden files, created without the defendant's knowledge or consent. The prosecution argued that their existence supported the charges of possessing indecent images of children. This highlights the importance of understanding how such files are created and their relationship to any evidence found, in order to show what actions were taken by the computer's user.

3 Methodology

In order to evaluate the structure and behaviour of the thumbnail cache used in Windows 7 a virtual machine was created with a 30GB hard drive. Windows 7 Ultimate was installed, using an ISO downloaded from the MSDN website. During

the installation, the default settings were selected in order to mimic a typical user system.

Once the installation was complete the baseline machine was cloned and examined to determine the initial state of the thumbnail cache and related files. A clone of the baseline was created for each set of experiments to assist in determining the structure and behaviour of the thumbnail cache. Any experiment specific methodology is described in the appropriate sections in further detail.

4 Identifying the Structure

In order to extract meaningful information from data it is necessary to understand its context; in the Windows 7 thumbnail cache the structure provides a context for the data. This section describes the structure of the various components of the Windows 7 operating system thumbnail cache. It begins with an overview of the thumbnail cache directory structure; this is followed by an overview of the structures contained within the six files which make up the thumbnail cache.

4.1 The Thumbnail Cache Directory Structure

Whilst examining the baseline image, it was determined that each user has a single centralised thumbnail cache which is located at the path below.

[Drive]:\Users\[User_Name]\AppData\Local\Microsoft\Windows\Explorer

This directory contains six thumbnail cache files: an index and five further files. The index holds records, which provide pointers to the location of sub-records in the remaining files. The rest of this section identifies the structures present within the six thumbnail cache files and identifies the records and sub-records.

4.2 Thumbcache_idx.db

The file 'thumbcache_idx' provides an index to the information stored within the thumbnail cache; each source file is represented by a single record in the index. A record contains pointers to the locations of the associated sub-records. The file has a standardised structure for records with each one using 32 bytes. Each record begins with an eight byte unique identification string known as the thumbcache ID. The thumbcache ID can also be found in the record relating to the original source file within the windows desktop search file, Windows.edb. The flags identify the file type. The remaining twenty bytes store pointers to the sub-records for the original source

file; each pointer identifies the starting position in bytes from the beginning of the relevant file.

The thumbcache_idx structure is different to the one used in Windows Vista, which used forty byte records. The last modification time of the source file has been removed, which accounts for the eight byte difference in record size.

4.3 Thumbcache_32, 96, 256 and 1024

The centralised thumbnail cache system implemented in Windows 7 allows for a maximum of four differently sized images to be stored for each source file. The images can be up to a maximum of 32x32, 96x96, 256x256 and 1024x1024 pixels respectively. Each of the four sizes is contained in its own file into which sub-records are stored; the structure of the sub-records can be divided into two separate sections: the header and the image. The sub-record header section contains the metadata for the sub-record and contains a clearly defined structure; the image section follows the standard structure for the image file type implemented, this is generally JPEG, BMP or PNG.

The 32 byte sub-record header structure has decreased by eight bytes from the 40 byte implementation used in Windows Vista; this is due to the removal of 8 bytes which represented the file extension in Unicode. The checksum values provide a method of authenticating the values in each section of the sub-record.

Whilst the thumbnail cache predominately stores visual thumbnail images, the thumbcache_256 also holds non-standard sub-records. Drive letters are stored for available drives, including portable storage devices; the network locations also include the name of the machine on which the thumbnail cache has been created.

4.4 Thumbcache_sr

Whilst this file is present in both the Windows Vista and 7 implementations of the operating system thumbnail cache, experimentation has yet to identify its purpose. The only recorded data for this file is contained in the standard file header.

4.5 Conclusion

This section has described the structure of the record and sub-record implementations used in the Windows 7 operating system thumbnail cache. The cache consists of six files: an index and five files to hold sub-record data. The records stored in the index store pointers to the sub-record data for each source file. The visual thumbnail sub-records consist of two sections, a header and an image. The following section will

show the context in which the artefact structures identified in this section occur on an operating system.

5 Identifying the Behaviour

Three types of action can be observed on the information stored within the thumbnail cache: information can be added to files within the thumbnail cache, it can be modified or it can be deleted. This section identifies the behaviour the Windows 7 thumbnail cache exhibits when a user interacts with the operating system. For the experiments in this section both Microsoft Office and Adobe PDF viewer were installed on the baseline machine; the default settings were selected during their installation. The software was installed to allow common user file formats to be evaluated and manipulated during the experiments.

5.1 The Creation of Records and Sub-records

When a user interacts with a system, there are several ways in which they can create a new file: they can use an application, they can move a file from a storage device, or they can download a file from the internet. They can also view the files using a variety of methods such as in an application or in a file browser. This sub-section identifies the behaviour which results in records and sub-records being added to the thumbnail cache.

5.1.1 Using Different Methods for Adding Files

When a file is created in an application such as Microsoft Word it can be saved to a storage location, which permits the created file to be accessed again at a later date. As part of this research, several files were created using this method; it was noted that a record was not added to the thumbnail cache for these files until they were viewed in thumbnail mode. When a file is viewed in thumbnail mode in file explorer, or in a dialogue Window, like those used for opening and saving documents, a thumbnail record is created or modified as required for each potential thumbnail cache source file. In figure 5.1, several files are being viewed using the standard file browser. Each potential user created thumbnail cache source file will have records added into the thumbcache_256.db and thumbcache_96.db file; whilst system created images such as those used for the directory 'Not Viewed' will only be added to the thumbcache_96.db. If the size of the icons used is increased from medium to large icons, the resulting additions to the thumbnail cache would not change. However if the extra large icons are selected, the directory visual thumbnail would also be added to the thumbcache_256.db file. Sub-records are added to the thumbcache_1024.db file when they are previewed in the preview pane of the file browser; however this only

occurs if the image being previewed is greater than the maximum size image permitted in the thumbcache_256.db file. Experimentation has found that image files, PDF files and directories viewed in thumbnail mode consistently add records to the thumbnail cache; however Microsoft Word files do not always create thumbnails.

When files are viewed on removable storage devices in thumbnail mode, entries are added to the thumbnail cache, experiments have shown that when a file is moved from a storage device to a hard disk drive a new record is added to the system when the file is subsequently viewed in its new location. This shows that thumbnail cache entries for removable storage devices would be present in the thumbnail cache and may provide corroboration that a device has an relationship with a particular machine or user. Network storage viewed in thumbnail cache mode does not create entries in the thumbnail cache until a file is moved to a local drive or device.

Any sub-records added to a thumbnail cache file are appended to the end of that file. However experiments have shown that the index file is a hash table, where entries are assigned a place in the table based on a calculation using the eight byte thumbnail cache ID number.

The index allocates one hundred and one spaces in the thumbnail cache when it is initialised. Experiments have shown that for index sizes fewer than five hundred and five records, each time a clash occurs in the hash table, another one hundred and one spaces are added to the table. For larger index sizes the number of new records added each time increases but is always a multiple of one hundred and one. All the existing records are then recalculated to account for the new table size.

Records added to the thumbnail cache may contain associated data in the Windows.edb file; the record within the windows.edb contains a thumbcache ID which allows a relationship between a source file and a thumbnail cache record to be shown. Figure 5.2 shows a file 'Cake.JPG', and shows the structure of the relationship between the files.

It is important to note that records within the thumbnail cache may not be directly associated with a visual thumbnail image; these records may instead hold information in Unicode in the identifier section of the sub-record header. Experiments have shown these records to contain GUIDs relating to entries in the Windows registry, the letters associated with allocated drives and networked places. The drive letter and network place records are generated when the file browser is used. As can be seen in figure 5.1, the right side of the file browser has icons for drives, folder locations and network places. As these items use default icons, only details are stored in the thumbnail cache. New records are added when locations are added to the right side panel, either by the system or a user and are visible in the file browser; sub-records

relating to locations which are no longer present are not removed. This is a useful source of information for analysts as it provides a list of drives which have been allocated and machines which have been networked to the users machine; it also provides the name of the user's machine as this is shown in the network places list within the panel.

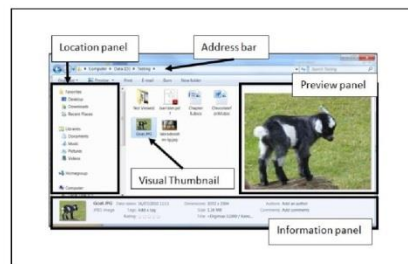


Figure 5.1: The creation of sub-records in thumbcache_1024.db

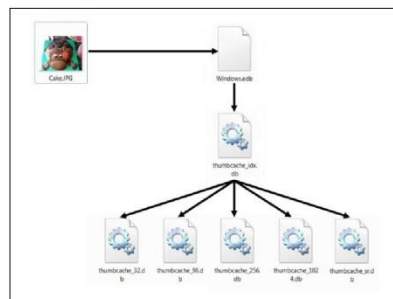


Figure 5.2: The relationship between a file and a record in the thumbnail cache

5.1.2 Circumstances where information is added to the thumbnail cache

A record is added to the thumbnail cache when no record currently exists for a given URI and the file at the URI is a potential thumbnail cache source file.

When a matching record cannot be found in the index a new record can be created under the following circumstance:

- A directory is viewed using the file browser, or the file browser within an application using thumbnail mode
- The directory being viewed contains file types which are compatible with the thumbnail cache
- A file exists within the directory which does not currently have a thumbnail cache record, or a sub-record of the appropriate size.

5.2 The Modification of Records and Sub-records

A user can modify a file in three distinct ways, by modifying the contents, the URI, or deleting the file. This sub-section discusses the experiments performed to ascertain the behaviour of the thumbnail cache when the file is modified; Section 5.3 discusses the deletion of files.

5.2.1 Modifying the Contents of a File

In order to ascertain the behaviour of the thumbnail cache when a user modified a file, several files were modified using their default applications. The files were all located in the file browser, and then selected to open, each file was then modified, saved and closed. The thumbnail cache was examined after the file was closed and no related changes to the thumbnail cache were observed.

The modified files were then viewed using a variety of methods, including through the open command in their default application and through the file browser. The results showed that when a file was viewed in thumbnail view, the record for the source file was updated to reflect the current state of the file. During the experiments it was noted that the thumbcache_1024.db image, where present was not consistently updated when the sub-record was viewed.

5.2.2 Modifying the Files URI

If a source file changes its corresponding URI, a new record is generated in the thumbnail cache and no alteration is made to the existing record. This effect was recorded during a series of experiments where a file was placed at a URI and viewed to ensure a record was added in the thumbnail cache. Once the record was stored in the thumbnail cache, the URI of the file was changed; this involved either altering the file's name or the entire path. The file was then viewed at its new URI and the effect

to the thumbnail cache was analysed. A new record was created in the thumbcache_idx, with sub-records also being created. Aside from the different thumbcache ID value, the record and sub-records were identical to the original test data. This suggests that it may be possible to create a false relationship between artefacts in the thumbnail cache and a source file.

In diagram 5.3 a directory is being viewed using Windows Explorer, the directory contains a sub-directory named 'Not Viewed'; which is represented by a folder and a selection of thumbnails. Experimentation has shown that the thumbnails used to represent the contents of a directory are selected when the thumbnail is first generated. If the images used in the thumbnail are removed from the directory, the directory thumbnail is not always updated, which may lead to an inaccurate representation of the directory's current contents. Only files which are used for the directory's visual thumbnail record have sub-records created for them; therefore there may be records in the thumbnail cache for files which the user has not viewed. Since each user area has its own centralised thumbnail cache, the thumbnails used in the directory thumbnail image must represent source files which have been present in the directory at some point. It is also interesting to note that the standard visual thumbnail in the directory browser and the visual thumbnail in the information panel are not identical. This is because they are both being taken from different sub-records for that entry in the thumbnail cache and the sub-records were generated at different times and were invoked by different processes which were unaware of other sub-records for the thumbnail cache entry.

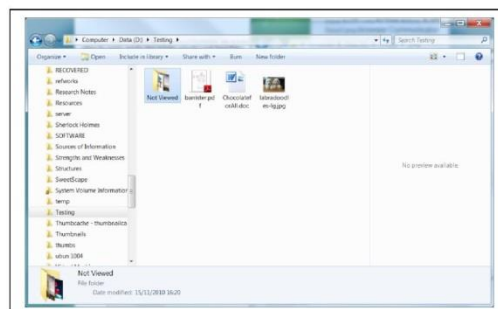


Figure 5.3: A screenshot from a test to see if records were created for files which had not been viewed

5.3 The deletion of records and sub-records

There are several options in Windows 7 which allow the removal of the thumbnail cache; for example the cache could be deleted by the user. However if the cache is simply deleted, the cache will reinitialise and start again, this may remove traces of a file which is no longer on the system from the current thumbnail cache. The disk clean up wizard provides an option to remove the thumbnail cache, this simply deletes the current thumbnail cache; a new cache would be created next time it was called.

The local group policy editor has an option to “turn off the caching of thumbnails in hidden thumbs.db files”, suggesting that Windows 7 can utilise both the directory specific and centralised forms of thumbnail cache. During the experiments conducted in this research, no evidence of a thumbs.db was recorded; however they have been identified on actual systems, suggesting further research into their behaviour is required.

Experiments showed that if a file is deleted from the system, it is not automatically removed from the thumbnail cache, instead the record remains; the cache also does not remove records which have not been accessed for some time. However it is possible to alter the default settings within Windows 7, to limit the size of the thumbnail cache, which experiments have shown causes the cache to remove inactive records when it reaches its maximum size. The cache has been shown to create a sub-directory within the explorer directory named ‘ThumbCacheToDelete’, this directory is where outdated versions of thumbnail cache files are placed prior to deletion. Moving the thumbnail cache files allows the system to regenerate new files; this is particularly used when the index requires extra space. During this research no method tested has led to the secure deletion of thumbnail cache files, therefore once deleted the files are available in unallocated space.

5.4 Conclusion

This section has described the behaviour of the thumbnail cache, it has highlighted the requirement for files to be viewed in thumbnail view for records to be created or modified. However there are situations, such as in the creation of visual thumbnails for directories, where a record may be created for a file which has not been viewed. It has shown the thumbnail cache contains records which do not have visual thumbnail images, these records provide information on which drive letters have been allocated and the names of networked machines. When a file is modified the thumbnail cache is only updated when the file is next viewed in thumbnail mode; however the old record is not deleted. There are various methods for deleting the thumbnail cache and preventing it from regenerating; however the deleted cache is not securely deleted. Finally the allocation of blocks of space to the thumbnail cache provides large

fragments which generally contain several records or sub-records, which may assist with fragment classification.

6 Conclusion

This paper has examined the structure and behaviour of the operating system thumbnail cache implemented within Windows 7. Within Windows Vista, the thumbnail cache implementation changed significantly from Windows XP by using a centralised thumbnail cache for each user, this implementation was continued in Windows 7 with slight differences in the structure of records and sub-records. The thumbnail cache contains visual thumbnail records alongside GUID information, active drive letters and network places. The non-standard sub-records can provide assistance to an examiner by showing the structure of the device, associated devices and associated machines which may require analysis. Records are added to the cache, with a new sub-record being added to the bottom of the thumbcache files. Some records are deleted periodically, however this is application specific. The sub-records contain checksums to assist in authenticating the data; however it is still possible to tamper with parts of the thumbnail cache without it detecting an error.

References

1. Hurlbut, D., "Thumbs DB Files Forensic Issues". AccessData Training Document. 2005. http://www.accessdata.com/media/en_US/print/papers/wp.Thumbs_DB_Files.en_us.pdf
2. Hargreaves, C., Chivers, H. and Titheridge, D. (2008), "Windows Vista and digital investigations", Digital Investigation, vol. 5, no. 1-2, pp. 34-48.
3. Morris, S. (2010) "A comparative study of the structure and behaviour of the operating system thumbnail caches used in Kubuntu and Ubuntu (9.10 and 10.04)". Proceedings from 4th Cybercrime Forensics Education & Training, Canterbury Christ Church University, Canterbury, UK.
4. LiveLeak, "FBI agents posting bogus links and entrapping people". 2009. http://www.liveleak.com/view?i=87d_1233031430
5. McCullagh, D., "FBI posts fake hyperlinks to snare child porn suspects". 2008. http://news.cnet.com/8301-13578_3-9899151-38.html

C.3

Morris, S.; Chivers, H.; 2011b. Forming a Relationship between Artefacts identified in thumbnail caches and the remaining data on a storage device. In *Cybercrime Forensics Education and Training*. Canterbury, UK.

Forming a Relationship between Artefacts identified in thumbnail caches and the remaining data on a storage device

Sarah Morris¹, Howard Chivers²

Centre for Forensic Computing, Cranfield University
Shrivenham, SN6 8LA

¹S.L.Morris@Cranfield.ac.uk

²H.Chivers@Cranfield.ac.uk

Abstract

The primary function of a thumbnail cache is similar between different operating systems; however there is no consistent implementation; because the thumbnails are potentially interesting to forensic analysts it is important to understand the detail of how they are used in a particular operating system. Previous work has shown the importance of understanding the structure and the effect of user behaviour on various thumbnail caches. However, an analyst needs to demonstrate a relationship between artefacts identified in the thumbnail cache and those found elsewhere on the system in order to provide context and corroboration of any evidence derived from thumbnails. A relationship between artefacts can also assist in establishing possible event time lines, and understanding the user behaviour which led to the system being in its current state.

This paper establishes the relationships which are formed between user generated files and information stored in the thumbnail cache; this shows how a forensic analyser can infer relationships between the thumbnail cache and other artefacts identified on the system. This paper provides a description of each relationship between the thumbnail cache and other artefacts; these relationships allow the corroboration of evidence extracted from the thumbnail cache and provide an additional source of evidence of user behaviour. In addition to providing a useful reference for analysts when reconstructing a user's activity, this paper also uses the thumbnail cache as an example to discuss the importance of contextual analysis within forensic computing.

In addition to the relationships shown between standard image thumbnail cache records and the rest of the system, this research also identifies how relationships are formed between the thumbnail cache and system artefacts such as the icons present on the user's desktop, and also allows the identification of devices on the same network as the user.

1. Introduction

The primary function of the operating system thumbnail cache is to store visual thumbnails relating to user generated files in order to save system resources from unnecessarily rendering of the images each time they are requested. An analyst therefore would try to establish a relationship between a user file and an entry in the thumbnail cache which may provide an insight into the user's behaviour.

Morris [2011] identified the structure and behaviour of the Windows 7 thumbnail cache; the work identified the potential context of the records and sub-records within the thumbnail cache. Whilst the identified context of the thumbnail cache can provide an indication of the user's activity and the corresponding historical state of a file, it is necessary for an analyst to form a relationship between this information and a user file

This rest of this paper is structured as follows: Section 2 identifies related work; the methodology employed during this research is described in Section 3. The ways in which a relationship may be formed between the thumbnail cache and a user file are shown in Section 4; Section 5 shows the potential relationships between the remainder of the system and the thumbnail cache. Finally in Section 6 the results of this research are discussed and the paper is concluded in Section 7.

2. Related Work

In forensic computing, admissibility is the term used to describe evidence which is allowed to be presented in court. Generally it is necessary to show a relationship between the evidence being produced and the events that actually occurred in the case. Whilst it may not be possible for an analyst to know exactly what happened, it is their job to produce a report. The analyst's report on the case provides assistance to the court as it is their analysis of the evidence. This can assist in determining the admissibility of evidence and the facts of the case [Kennedy, 2006]. Initially an analyst collects artefacts from a system and it is necessary to form relationships between the artefacts in order to provide context to the evidence. Contextual analysis assists the analyst in understanding the behaviour of both the user and the events which led to the current state of the system. One common method of performing a contextual analysis is by completing an event timeline.

Digital investigations use the same five key phases as those used in traditional physical investigations; preservation, surveying, documenting, searching, reconstruction [Carrier, 2004]. Reconstruction in digital investigations involves analysing the information recovered from a system and constructing events from it which can be ordered to show how the information recovered came to exist in its present state. An event can be defined as an incident that changes the state of one or more objects; in a digital investigation, an event can be thought of as an action that changes one or more of the bits in which the information is stored.

In a case, event reconstruction can assist in proving or disproving guilt by establishing the order in which events occurred and the times within them. Once events have been reconstructed, characteristics can be used to determine whether the actions were committed with intent [Carney, 2004]. For example, analysing times can give an indication of whether an action was the result of a piece of software or the user, since the user will have slower reaction times than the machine, there would need to be larger gaps between events than a machine would require.

The importance of contextual analysis can be seen in the case of Vosburgh [Find Law, 2010], one of the significant pieces of evidence recovered in the case were some indecent images of children found in a Windows XP thumbnail cache (thumbs.db). During the case the prosecution presented the argument that the only reason for the existence of the images to be present in the cache was if they had existed in a directory that the user had viewed. However the defence performed a live demonstration in court showing alternative methods for the evidence being found in the thumbnail cache. Based solely on the evidence within the thumbnail cache both the scenarios suggested by the defence and prosecution may have occurred; however if the rest of the system was examined to put the thumbnail cache evidence in context an analyst can draw up a likely set of user actions which assist in establishing a likely scenario.

3. Methodology

This research was conducted using virtual machines; a baseline image of Windows 7 was created using a standard Windows ultimate ISO downloaded from MSDN. During the installation default options were selected and a single user account was created. After installation of the operating system the base line virtual machine was cloned for each experiment. The experiments were designed to mimic typical user behaviour to identify information which changed on the remainder on the system that could be related to changes within the thumbnail cache. During the experiments potential thumbnail cache source files were added, modified and deleted both on the main and external storage devices. The resulting changes to each virtual machine were analysed using a variety of tools, including Encase and WinHex.

4. Forming Relationships between the Thumbnail Cache and a User File

Establishing a relationship between a file created by a user and the thumbnail cache assists in both corroborating the information identified and in forming an overview of the user's interaction with a file. This section describes the ways in which a relationship can be formed between a thumbnail cache record and a user generated file.

4.1 Visual Thumbnail

A visual thumbnail provides a snapshot of at least part of a file at a point in the file's history; if the visual thumbnail relates to the current state of the file then a visual comparison could be performed to identify if there is any similarity in its content to a user file. If the visual thumbnail and a user file appear to contain the same data then there may be a relationship between the two sets of information; an analyst cannot be certain the two sets of data are related. For example, there may be multiple copies of a file on the user's system, in which case all the copies may visually match with the same sub-record however only one copy would have a relationship with it.

The visual thumbnail may be compared to part of a user file using computer vision techniques; if the two images are the same size then a one-to-one mapping could be applied [Dufournaud, 2000]. However given it is likely that the two images are different sizes and may also be in different image formats the problem involves a one-to-many mapping; the increased requirement for the calculation of mappings may mean the technique is too resource intensive for large data sets and would therefore require the data set to be pre-processed. Like the visual inspection by an analyst a matching algorithm can only narrow down the potentially related sub-records.

The visual thumbnail may not represent the last state of the file, in Windows 7 each record in the thumbnail cache may contain up to four different sized visual thumbnails [Morris, 2011]; it is possible that the visual thumbnails may represent different states of the file. The states represented by the visual thumbnail may not match the last state of a file which would prevent a visual match of the data. It would therefore be necessary to examine each visual thumbnail for a record to maximise the chance of identifying a visual match between the data in the record and the file; it may also assist in identifying previous states of the file which may provide useful information for an analyst.

4.2 Metadata

The removal of time stamps from the thumbnail cache in Windows 7 makes it difficult for an analyst to establish the time a record was last updated within the

thumbnail cache [Morris, 2011]. However it is possible to identify a time period during which a record was added to the thumbnail cache; previous versions of the thumbnail cache files are stored in Windows 7 by the Volume Shadow Service. By right clicking on the directory 'Explorer' where the thumbnail cache is located it is possible to identify previous versions of the directory. Selecting each previous version will show the thumbnail cache files in an earlier state; by comparing the information within the versions of the thumbnail cache it is possible to identify a basic order in which records were added.

Figure 1 shows the windows restore points for an Explorer directory. Since each previous version has a date attached it is possible to work out a time frame during which a record was added. It is also possible to identify the order user created files were added to the thumbnail cache by identifying their position in the individual thumbnail cache files. New sub-records are appended to the end of the file; sub-records relating to user generated content do not alter their relative position; however sub-records relating to other types of data tend to move closer to the top of the file. As a consequence of the movement of some sub-records may mean that the offset to user generated sub-records may alter slightly.

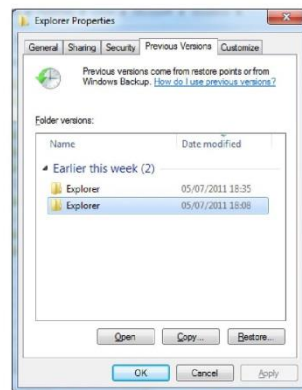


Figure 1: Restore points for the directory Explorer

The type of a file can be identified through the flags stored in the thumbnail cache; each type of file has a specific sub-set of flags; these flags are found in the main record for each file located in the thumbcache_idx file [Morris, 2011]. The flags can be used to identify the type of file which relates to the record, as the file type may not be obvious from a visual inspection.

4.3 Thumbnail cache ID

Each record contains a unique thumbnail cache ID which is made up of sixteen hexadecimal characters. Testing has shown this ID is comprised of data relating to the user file; it has been identified that the thumbnail cache ID is related to the volume, MFT data and the type of the file. In Section 5 the thumbnail cache ID is used to form a relationship through the Windows Desktop Search database.

4.4 Conclusion

This section has identified ways in which a relationship can directly be made between information in the thumbnail cache and a user generated file; whilst matching the visual thumbnail to a file can suggest a relationship it is only possible if the thumbnail represents the last state of a file. A match between a visual thumbnail and a file only provides an indication that the two data sets are related as there may be multiple files which match the thumbnail; however with corroboration from other sources it may be possible to strengthen the relationship. The removal of time stamps from the thumbnail cache makes it difficult for an analyst to provide an accurate time for when information is added to the thumbnail cache; however an event timeline can be constructed using the restore points for the thumbnail cache directory. Finally the thumbnail cache ID can assist in showing the volume, state of the MFT and the type of the file a record relates to.

5. Forming Relationships between the Thumbnail Cache and the Remainder of the System

In Section 4 methods were identified for defining direct relationships between a user generated source file and the thumbnail cache; this section looks at other relationships which can be formed between the thumbnail cache and system artefacts.

5.1 Windows.edb

The database for Windows Desktop Search stores a wide range of information including comprehensive metadata about user generated files [Chivers, 2011]; therefore the database can be used to provide extra and corroborating information on a user file which may be of interest to an analyst. Interestingly, the Windows Desktop Search database contains a field which holds the thumbnail cache ID; this field can be used to form a relationship between a file indexed in the database and a record in the thumbnail cache. However, there may not be a record in the database relating to the file or thumbnail cache ID the analyst is looking for; Chivers proposes several approaches to recovering deleted records in the database, which may recover the record the analyst requires.

5.2 Registry

There are records stored in the Windows 7 thumbnail cache that do not contain visual thumbnails [Morris, 2011]; some of these records contain a GUID. The GUID generally relates to system icons found on the user's desktop; the relationship can be identified by searching for the GUID within the registry. Figure 2 shows the results of searching for a GUID which relates to the Recycle Bin; by selecting the default icon directory shown in Figure 3 it is possible to identify the location of the icons being used. In this example the icons are stored in `imageres.dll` in positions 54 and 55 depending on the state of the recycle bin. In order to identify the icon used on the desktop it is necessary to identify the contents of file containing the icon as the result may be different to the standard icon, for example if the user has a customised theme. As identified in Section 4 the use of restore points can identify when the icons appeared on the desktop; generally system icons are added when a user first logs into their user area, however it is possible to add icons. The sub-records containing GUID's generally move to the top of the thumbnail cache file and do not always contain a reference in the index file. System icons may also be removed by a user, however the reference to the icon will remain in the thumbnail cache which may assist in identifying the behaviour of the user.

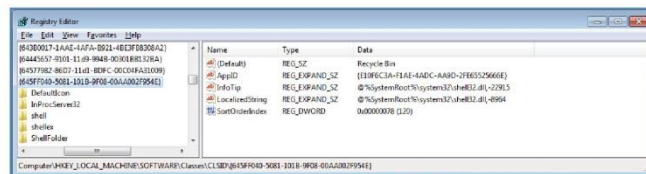


Figure 2: Searching for a GUID

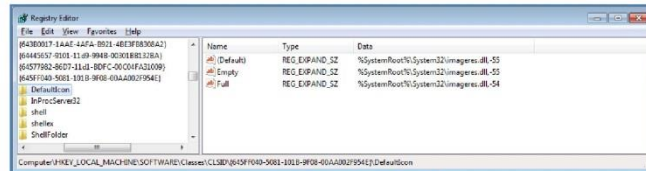


Figure 3: Identifying the Default Icon

involved in its creation. Visual thumbnails may be stored in the centralised cache for any potential source file which is stored on an indexed volume; during testing external media were automatically indexed when connected to the system.

5.5 Encrypted Containers

Encrypted containers such as those used in TrueCrypt [2011] can be mounted as a volume by a user to enable the content to be accessed. Once the content is mounted it is possible to index the container the same as a normal volume. During testing it was found that the selection of records within a FAT32 TrueCrypt volume by a user results in the creation of corresponding records within the thumbnail cache.

5.6 Network Storage and Drive Allocation

There are sub-records stored in the thumbcache_256 file which store names in Unicode. Testing has shown these names to relate to network places and drive letters which have been allocated on the system. Figure 6 shows the drives allocated for the system; it is interesting to note that a list of drives is available in the central part of the explorer window as well as on the left hand side under the Computer icon. Both these areas of the explorer window will create records in the thumbnail cache; the drive letters stored in the thumbcache can be corroborated by checking the mounted devices stored in the registry.

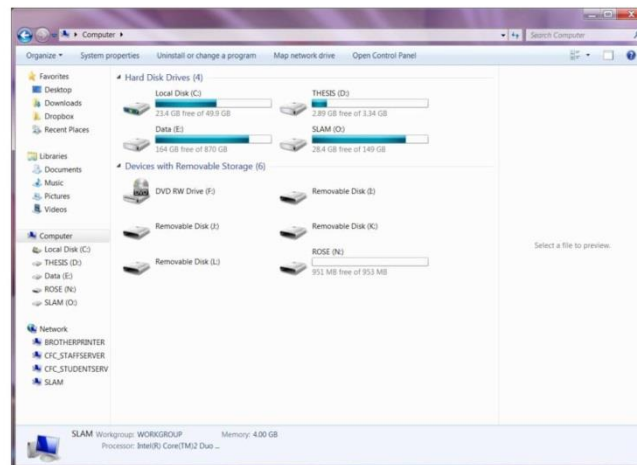


Figure 6: Drives currently available to the user

In Figure 7, a list of networked places can be seen; again the names for these places can be seen in both the main part of the explorer window and in the left hand side frame. Each networked place is stored in Unicode in the thumbnail cache; an analyst could identify the name of the machine being examined in the registry; the remaining networked device names may assist the examiner by providing a list of devices which may be of interest.

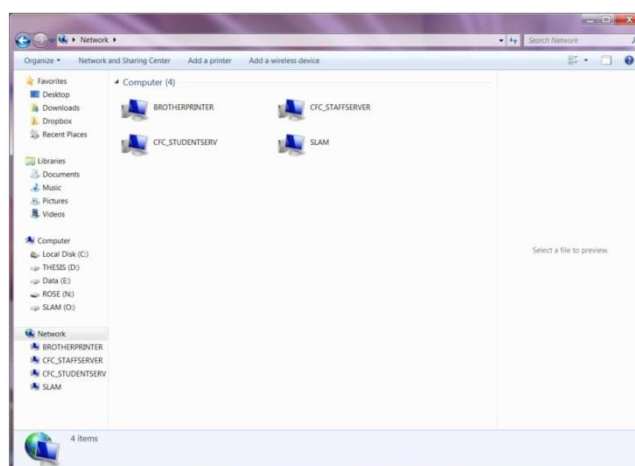


Figure 7: Networked places available to the user

5.7 Conclusion

This section has described the relationships between the thumbnail cache and other potential system artefacts. The Windows Desktop Search database can assist an analyst in forming a relationship between a file and an entry in the thumbnail cache as for user created files it stores a field for the thumbnail cache ID value. It is possible to carve out deleted records from the database, but the record an analyst requires may not be found. The GUID's stored in the thumbnail cache can be found by searching the registry; this will enable the type and default icon to be identified. Information on the icons used for shortcuts is also stored in the thumbnail cache; default icons for shortcuts may also be available in the registry. Thumbnail records for external storage media will only appear in the centralised thumbnail cache if the volume has been indexed by the system; if the volume has been indexed it behaves in the same way as internal volumes. Encrypted NTFS containers behave as a normal volume when mounted; therefore if they are indexed by the system records are created in the standard way. If the volume mounted is

FAT records tend to only be created for selected objects. Finally the names of network devices and allocated drive letters are available in the thumbnail cache, the name of the system being examined can be identified from the registry; the other device names stored may be further potential sources of evidence. The drive letters can be corroborated using the mounted devices key in the registry.

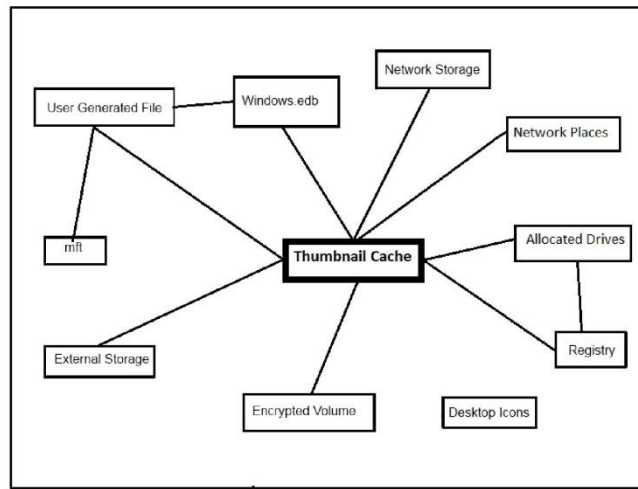


Figure 8: A Diagram of the relationships between the thumbnail cache and other system artefacts

6 Discussion

The relationships between the thumbnail cache and other artefacts on the system have been discussed in this paper; Figure 8 shows the relationships diagrammatically. Whilst the relationships vary in strength this paper has shown a variety of artefacts which can be combined to form corroborating evidence and assist in identifying user behaviour and understanding the context of the evidence. This research also shows relationships with the thumbnail cache can assist in showing the structure of the system through the drive letters allocated; the structure of the network can also be identified through the network places stored in the thumbnail cache records.

Understanding the context of these artefacts assists the analyst in establishing how the system came to be in its current state and what the evidence means. It is necessary to establish the user behaviour and this is commonly done through the creation of event timelines; the removal of times from the thumbnail cache in Windows 7 makes it challenging for an analyst to add the evidence from the cache to a timeline. However the use of system and file restore points allow an examiner to narrow down the time in which evidence was added to the thumbnail cache; thereby creating a basic order of when events occurred. Interestingly a greater understanding of the artefacts and user behaviour develops when the relationships are combined; the collection of evidence produces a greater insight into the system as a whole.

7. Conclusion

This paper has shown that relationships can be formed both directly and indirectly from the thumbnail cache to user generated files and other artefacts on the system. These relationships assist in providing corroboration of the evidence and provide further context of the artefacts identified. Contextual analysis of artefacts identified in a case can assist with understanding the user and system behaviour which led to the current state of the evidence. Encouraging examiners to view evidence contextually would lead to a greater understanding of the artefacts being analysed which allows a more complete portrayal of the evidence to the court to assist in ensuring justice.

References

- Carney, M.; Rogers, M., The Trojan made me do it: a first step in statistical based computer forensics event reconstruction, International Journal of Digital Evidence Vol. 2 Issue 4. 2004.
- Carrier, B. File System Forensic Analysis. Addison-Wesley, PA., March 2005.
- Chivers, H.; Hargreaves, C., Forensic data recovery from the Windows Search Database, Digital Investigation, Vol.7 Issues 3-4. 2011
- Dufournaud, Y., Schmid, C., and Horaud, R. 2000. Matching images with different resolutions. In Proceedings of the Conference on Computer Vision and Pattern Recognition, Hilton Head Island, South Carolina, USA
- Find Law, "No. 08-4702. – UNITED STATES v. VOSBURGH – US 3rd Circuit". 2010. <http://caselaw.findlaw.com/us-3rd-circuit/1522221.html>
- Kennedy, I. "Presenting digital evidence to court". 2006. <http://www.bcs.org/content/ConWebDoc/7372>

Morris, S.; Chivers, H. (2011) "An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache". Proceedings from 1st Cyberforensics Conference. University of Strathclyde, Glasgow, UK.

TrueCrypt, "TrueCrypt – Free Open-Source On-The-Fly Disk Encryption Software for Windows 7/Vista/XP, Mac OS", 2011. <http://www.truecrypt.org>

C.4

Morris, S.; Chivers, H.; 2013. An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache. In *Cyberforensics*. Cardiff, UK.

Thumbnail Cache File Fragment Identification using a Bayesian Network

Sarah Morris¹, Howard Chivers²

¹ Centre for Forensic Computing, Cranfield University, SN6 8LA
S.L.Morris@Cranfield.ac.uk

² Department of Computer Science, University of York, YO10 5GH
Howard.Chivers@York.ac.uk

Abstract.

The increased awareness of digital forensics and privacy has led to a growth in users trying to remove traces of their activity. Therefore information which may be potentially relevant to an investigation may be found in unallocated space. Artefacts found in unallocated space may only consist of a single file fragment. However with the growing size of storage media such artefacts may be difficult to identify without using an automated method. Previous research has highlighted the importance of thumbnail cache artefacts to digital forensics. This research aims to identify single thumbnail cache file fragments using a Bayesian network. The research constructed a Bayesian Network using the information gathered about the structure and characteristics of common operating system thumbnail cache file formats. The Bayesian truth tables were generated and tested using a large corpus of data created during previous research. This research identified at least 62.4% of each of the five classifications. The three classifications which contained structured information had success rates of over 98%.

Keywords: File Carving, Thumbnail Cache, Bayesian, Digital Forensics

1 Introduction

The growing size of storage media and the possibility that only fragments remain of a piece of important evidence suggest that analysts may need to investigate the contents of unallocated space. It is too resource intensive for an analyst to investigate unallocated space in depth for every case. However the ability to identify fragments and reassemble them using an automated method may provide an analyst with further information about the state of the system with minimal resource cost. The likelihood of potentially relevant thumbnail cache artefacts being in unallocated space suggests research into identifying thumbnail cache file fragments would be relevant and may

assist in automating a potentially lengthy process. The process developed will also show that the approach is generalisable as it will have to be adaptable to different implementations of the thumbnail cache. Therefore the approach used in this research could enable the identification of other file types.

This paper provides an overview of the first file fragment identification approach evaluated as part of a comparative study of common file carving approaches adapted for single fragment identification. The study uses the thumbnail cache and a corpus of data to evaluate the strengths and weaknesses of each implemented identification method. One aim of this study is to create a single method for the identification of thumbnail cache file fragments which uses the strengths identified during the comparative study. The identification methods constructed during this study are all implemented in such a way that a forensic analyst could repeat the methods; the methods are constructed in such a way that a typical analyst could implement the methods for other file types and present the results in Court.

This rest of this paper is structured as follows: **Section 2** identifies related work. The methodology employed during this research is described in **Section 3**. The data used during this research is described in **Section 4**. **Section 5** describes the construction of the Bayesian Network. The results of this research are presented in **Section 6**. A discussion of this research is provided in **Section 7**. Finally the paper is concluded in **Section 8**.

2 Related Research

Pal [1] describes the evolution of file carving. File carving is the technique of extracting a file from a storage device based upon its structure. It is commonly used when the directory information is unavailable, to extract as much of the file as possible. Early file carving methods were based on identifying the header and footer of a file and then carving out all the information between these two points; this provides valid results only when the file is: contiguous, not fragmented, and has a footer [2]. Once a file signature is identified the file is carved until a footer is identified. If the file does not have a footer then it is possible to carve the file by extracting a predetermined number of bytes which is larger than the expected file size.

Cohen [3] suggests that identifying positive and negative constraints on files, to establish whether a fragment belongs to the correct file type, can improve efficiency and give a greater level of accuracy. Most algorithms implicitly use this approach to rule out fragments that are not of the correct type. His descriptions of carving PDFs, Zip files, Mpegs and email formats highlight the requirement to understand the structure of the file being carved. This ensures the resulting file can be validated

Fragments can be classified based on pattern recognition statistics. A prototypic histogram is used in a supervised learning environment to attempt to model file types,

and research has shown this to be reasonably successful with a low error rate for HTML and JPEG files [4]. Veenman also establishes the need to combine such techniques with pre-existing carving techniques in order to maximise the information being carved, which may take a greater length of time to run on a given storage device. To date, such techniques have been relatively separate in their research and implementation, and whether developing a tool encompassing several methods would decrease the error rate on more complex file types is still unknown.

3 Methodology

This paper describes using a Bayesian Network to identify thumbnail cache file fragments. The methods described earlier in this paper were based upon the identification of file fragments which consisted of multiple clusters. This research focuses on identifying single fragments of thumbnail cache files, without using any information outside the fragment to assist with its identification. The aim of this research is to identify thumbnail cache file fragments. However this is rather broad and requires the terms fragment and thumbnail cache file to be further defined. For the purposes of this research a fragment will be considered to be a cluster of data. For this research a cluster has been defined as 4096 bytes. There is no standard implementation of an operating system thumbnail cache and therefore trying to identify any thumbnail cache file fragments would be a broad aim. For this research it has been decided that only the centralised operating system thumbnail caches from Ubuntu, Kubuntu and Windows 7 will be considered. This research is comprised of two sections: the generation of a corpus for training and testing (**Section 4**), and the construction of the Bayesian network (**Section 5**).

4 Information to be carved

In previous research Ubuntu, Kubuntu and Windows 7 thumbnail caches have been researched [5, 6, and 7]. As a result of this research a corpus of data based on these three operating systems has been collected. The data corpus also contains a wide selection of user generated files which were collected for use during the previous experiments. This section describes the generation of the corpus and how it has been constructed for this research.

4.1 Corpus Generation

In order to create the data set all the live files were extracted from virtual machines created during research into the structure and behaviour of thumbnail cache files [5, 6, and 7]. The live files were classified into sub-directories of the corpus by file extension. For Windows 7 this generated 4147076 individual files with a total size of approximately 500 GB. For Ubuntu, 2197329 individual files, with a total size of ap-

proximately 140GB. A further 250 GB of user generated content was added to the corpus. These 1095696 files were collected specifically for this research and contained a wide variety of user generated data. The files were either created using software tested during the previous experiments or downloaded from file type specific searches made using Google.

4.2 File Fragment Classification

There is a possibility that a storage device may contain multiple operating systems. Therefore it is appropriate to search for both Linux and Windows thumbnail cache fragments on a device. This led to 6 classes of files for this research.

- H1:**Thumbcache_idx.db file
- H2:** Image thumbcache file and contains metadata
- H3:** Image thumbcache file and only contains JPEG image data
- H4:** Linux PNG visual thumbnail and contains metadata
- H5:** PNG visual thumbnail fragment and only contains image data
- H6:** Not part of a Windows or Linux thumbnail cache

4.3 Training and Testing Sets

The data set contains a wide variety of file types which may be found on user systems during an investigation. As it is not feasible to produce a data set which contains every type of file fragment which may occur on a user's system, a data set needs to be selected which can provide a wide variety of file types which are likely to be on a system under investigation. The corpus created for this research contained a wide variety of system and user generated file fragments. In order to evaluate the identification methods created during this research the data corpus is split into training and test sets.

The training set is used to assist each method with modelling based upon fragments with known classifications. The training set was made up of approximately half the data. Using a random number generator half the fragments from each file type were selected. For the thumbnail cache, half the records for each classification of thumbnail cache data were selected. Where the number of fragments in a category was odd, the number of selected fragments was rounded up. The calculation of the Bayesian Truth tables is described in **Section 5**. The testing set comprised the remainder of the data.

5 Bayesian Network Approach

The Bayesian Network takes into account the probability that an artefact or set of artefacts appearing in a fragment relates to its classification. In order to construct the Bayesian Network originally each artefact relating to each hypothesis was shown in

the network. However this led to the development of a complex network which required refinement.

The nodes for the network were selected based upon characteristics identified during previous studies [5, 6, and 7]; the method for identifying characteristics could be replicated by a forensic analyst for other file types.

5.1 Construction

In order to construct a Bayesian Network it was first necessary to establish characteristics which have a relationship with each of the classifications. The complete Bayesian Networks for the identification of thumbnail cache file fragments are shown in figures 1-3. As can be seen from the figures a selection of structural, syntactical and frequency based statistics were used to identify fragments. The related probability tables and a short description of each node can be found on Morris [8]. It is interesting to note that in the original Bayesian Network all nodes were directly linked to the hypothesis nodes. However upon examination of the statistics it was noted that some artefacts have a greater impact on the validity of a hypothesis when combined together leading to the layers in the network. For each node with dependencies the probability table is created using the formula below.

$$p(X|Z) = \frac{p(X) p(Z|X)}{p(Z)}$$

In this research the Bayesian Network nodes will have Boolean outcomes. A node can resolve to either True or False. The equation provides the probability of a node resolving to True given the data that directly affects it within the network. X represents the node itself, while Z relates to any nodes directly influencing X. For this research, the syntax $p(X|Z)$ means the probability of X being True given Z is True). In the Bayesian network nodes can have multiple parents. In this case Z would represent the state of the parent nodes. For example if a node had two parents, Z could represent one of four states ([True, True], [True, False], [False, True], [False, False]).

After the construction of the probability tables for each node the network could be used for to classify each fragment. In order to calculate the probability tables the training data was used. For each value with no parents the probability of the value occurring was established by calculating the number of fragments the value occurred in compared to the number without the value. For values with parents the probability of each set of parent states occurring while the node value was True was compared with the probability of the parent states occurring with the node being False. Therefore the probability of a set of states occurring in a thumbnail cache file fragment was compared with the probability of the states occurring in a fragment which does not belong to the thumbnail cache.

5.2 Using Bayesian Networks for Classification

The Bayesian Network shown in figures 1-3 and associated probability tables can be used to classify fragments. A program was constructed which identified whether each of the conditions of the independent nodes were True or False. This information was then used to calculate the probability of each of the classifications being true. A classification was deemed to be True if the probability of it being True, given the data, was above 0.75. If multiple classifications resolved to True, the one with the highest probability was accepted as the fragment classification.

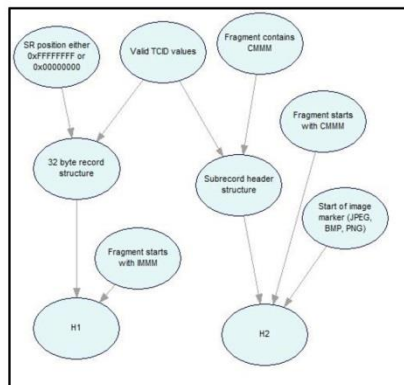


Fig. 1. Bayesian network for H1 and H2 classification

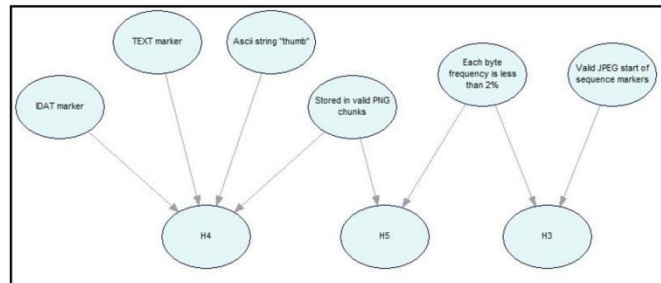


Fig. 2. : Bayesian Network for H3, H4 and H5 classification

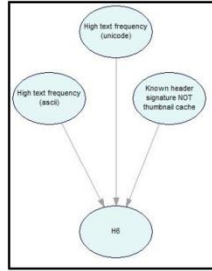


Fig. 3: Bayesian Network for H6 classification

6 Results

Table 1 shows the percentage of success and false positive rates for each thumbnail cache classification using the Bayesian Network on the testing data set.

The success rate represents the percentage of fragments in a category which were correctly identified by the method as belonging to that category.

The false positive rate represents the percentage of fragments falsely classified as belonging to a category. The number of fragments incorrectly identified as a category is divided by the total number of fragments which should not have been classified in that category; this number is then multiplied by 100 to provide a percentage false positive rate.

Table 1. Percentage Success and False Positive rate for the data set

	H1	H2	H3	H4	H5
Success Rate	98.25	100	62.46	100	68.66
False Positive Rate	0.11	2.52	5.33	0.08	4.09

7 Discussion

The potential size of unallocated space on a storage device makes the individual analysis of each cluster infeasible. Therefore a method for the automated identification of information was required. At the time of conducting this research there was little information available about single fragment identification; particularly in cases

where the fragment classification was based solely on the information within the fragment. This suggested that an existing method for the identification of larger chunks of data had to be selected and adapted for single fragment identification.

The Bayesian approach takes the likelihood of a node occurring in both the classification and in other circumstances into consideration to build a statistical model based upon the training data. The use of statistical modelling allows a fragment which does not perfectly fit the structural and syntactical model to be classified. This allows for variations outside the observed data. The Bayesian network is designed using expert knowledge about the domain to construct a model to form relationships between characteristics

The Bayesian Network produced the most inaccurate results when identifying fragments which solely contained JPEG data. This is partly due to the fact it is difficult to distinguish between a JPEG fragment which belongs to the thumbnail cache and a JPEG fragment which belongs to any other file. For this research it was decided that it would be best to ensure all the JPEG fragments from the thumbnail cache were identified. Whilst attempting to minimise the false positives. This proved difficult and most JPEG fragments were classified as thumbnail cache image fragments. Therefore further research is required into methods of identifying JPEG fragments in order to improve the accuracy of their carving.

This research has highlighted the increased identification for classifications of fragments which have been studied and can be formally described. Therefore to identify fragments accurately it is necessary to understand the specific characteristics of the file type. The ability to accurately identify multiple types of thumbnail cache file fragments suggests that the approach documented can be generalised to other types of file. The method tested during this research was not able to identify all the thumbnail cache file fragments without false positives. This suggests there is room for improvement in the identification of single thumbnail cache file fragments. A single method was selected and adapted for this research. A comparative study would attempt to maximise the identification of thumbnail cache file fragments.

8 Conclusion

This paper has described research into the identification of thumbnail cache file fragments. This was achieved using a Bayesian Network which was tested on a corpus of data created during previous research. This research has successfully identified a significant number of thumbnail cache fragments with low false positive rates. The classification of image-only fragments was the least successful. It was difficult to distinguish between thumbnail cache image-only fragments and image-only fragments from different sources. This research used a single method for the identification therefore it is possible that another method may improve the ability to identify thumbnail cache file fragments. Future work in this area involves a comparative study

of common file fragment identification methods. Finally this research has highlighted the success of identifying fragments using a deep understanding of the characteristics of the file format.

9 References

1. Pal, A., Memon, N., 2009. The evolution of file carving. *IEEE Signal Processing Magazine*, 26(2), pp.59–71.
2. Calhoun, W.C. & Coles, D., 2008. Predicting the types of file fragments. *Digital Investigation*, 5(SUPPL.), pp.S14–S20.
3. Cohen, M.I., 2007. Advanced carving techniques. *Digital Investigation*, 4(3-4), pp.119–128.
4. Veenman, C., 2007. Statistical disk cluster classification for file carving. *Information Assurance and Security Symposium*. pp.393–398.
5. Morris, S., Chivers, H., 2010. A comparative study of the structure and behaviour of the operating system thumbnail caches used in Kubuntu and Ubuntu (9 . 10 and 10 . 04). In *Cybercrime Forensics Education and Training*. Canterbury, UK.
6. Morris, S., Chivers, H., 2011a. An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache. In *Cyberforensics*. Glasgow, UK.
7. Morris, S., Chivers, H., 2011b. Forming a Relationship between Artefacts identified in thumbnail caches and the remaining data on a storage device. In *Cybercrime Forensics Education and Training*. Canterbury, UK.
8. Morris, S., 2013. Identatron: Digital Forensics Research. Available at: www.identatron.co.uk [Accessed April 2, 2013]