
Article:
Chivers, Howard Robert orcid.org/0000-0001-7057-9650 and Hargreaves, C (2011) Forensic Data Recovery From The Windows Search Database. Digital Investigation. pp. 114-126. ISSN 1742-2876

https://doi.org/10.1016/j.diin.2011.01.001

DIGITAL INVESTIGATION 7 (2011) 114–126

available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

Forensic data recovery from the Windows Search Database

*Howard Chivers**, *Christopher Hargreaves*

The Centre for Forensic Computing, Cranfield University, Shrivenham SN68LA, United Kingdom

ARTICLE INFO

Article history:
Received 7 October 2010
Received in revised form
12 December 2010
Accepted 7 January 2011

Keywords:
Digital forensics
Desktop search
Microsoft windows
Database
Carving

ABSTRACT

Windows Search maintains a single database of the files, emails, programmes and Internet history of all the users of a personal computer, providing a potentially valuable source of information for a forensic investigator, especially since some information within the database is persistent, even if the underlying data are not available to the system (e.g. removable or encrypted drives). However, when files are deleted from the system their record is also deleted from the database. Existing tools to extract information from Windows Search use a programmatic interface to the underlying database, but this approach is unable to recover deleted records that may remain in unused space within the database or in other parts of the file system. This paper explores when unavailable files are indexed, and therefore available to an investigator via the search database, and how this is modified by the indexer scope and by attributes that control the indexing of encrypted content. Obtaining data via the programmatic interface is contrasted with a record carving approach using a new database record carver (*wdsCarve*); the strengths and weaknesses of the two approaches are reviewed, and the paper identifies several different strategies that may be productive in recovering deleted database records.

1. Introduction

Microsoft Windows Search[1] provides a database infrastructure and user interface to allow users to search for files, programmes, emails, and Internet history records. As files are created, accessed or modified, they are indexed in a single database. The data recorded includes standard file metadata, such as path and date-time information, content-specific metadata, such as the address of an email or the resolution of a picture, and a summary of file content. The content summary may also allow a significant part of an indexed file's content to be recovered; in the case of small files such as emails, often the whole text is available in the index. The database contains records for all the users in the system, and retains index information about files and folders that are unavailable, for example removable or encrypted drives.

This rich content has the potential to provide the forensic investigator with a valuable additional source of information about the files available to a system; however, despite the potential value there is little prior work on obtaining information from this database, and almost none on the possibility of recovering records that have been deleted.

The Windows Search system stores its records using a Microsoft database technology known as the Extensible Storage Engine (ESE), about which much is documented and for which Microsoft provide an application programming interface (API). Tools designed to manage or query this database provide one means of obtaining data from the search index, and this is the main option which has been available to investigators to date (see Section 2). However, similar to file system forensics where data can be recovered either by

* *Corresponding author.*
E-mail address: h.chivers@cranfield.ac.uk (H. Chivers).

[1] We acknowledge Microsoft copyright in the terms used in this paper to describe Microsoft products, including: Windows, Windows 7, Windows Vista, Windows XP, Windows Search, Extensible Storage Engine (ESE), *esentutl*, Shadow Copy, Encrypted File System, and BitLocker.

1742-2876/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2011.01.001

traversing the directory structure or by carving files from a disk image, so too could database records be carved without the use of database indexing and page structure. Both methods have potential, but the absence of an effective carving tool has prevented a study of their respective strengths and weaknesses. This paper presents such a study, based on practical results from a newly developed carving tool (*wdsCarve*[2]).

This paper describes the two aspects of Windows Search that may assist forensic data recovery:

- How the behavior of the search system provides investigators with opportunities to obtain information about files and folders that are otherwise inaccessible, for example that are on remote, removable, or encrypted drives;
- The possibility of recovering historical search records that have been deleted from the search database.

The paper reports representative experiments that demonstrate the behavior of Windows Search in practice. Experimental evaluation of such systems cannot be exhaustive; however, the results presented here provide the forensic investigator with sufficient information to be able to reason about the effectiveness of different recovery options for individual cases and the likely outcome in terms of forensic value.

The results described here were obtained using Windows 7, and while some of the detail is different for Windows XP and Windows Vista, the overall conclusions are the same.

This paper is organized as follows. Section 2 describes previous work, including related tools and sources of documentation, followed by Section 3 which defines some background terms and concepts. Section 4 describes what data are indexed, the user actions that give rise to indexing, and what is recorded about files and folders. This is followed by Section 5 which describes the prospects for data recovery, supported by the results of practical experiments. Section 6 provides an overview of the database record format, and describes the carving strategy and how this ensures the reliability of carved records; this is followed by a practical example of the recovery of Internet history data in Section 7. A discussion of the main forensic issues is presented in Section 8, and the paper is concluded in Section 9.

Three appendixes describe specific procedures used in the development of this paper: string searching of ESE compressed fields (Appendix A), the process for recovering database files and extracting records using the database API (Appendix B), and the process for extracting records by carving (Appendix C).

2. Related studies, documentation and tools

2.1. Documentation – interfaces and data formats

Both Windows Desktop Search (Microsoft, 2008; Microsoft, 2009), and the Extensible Storage Engine (ESE) (Microsoft, 2007) are documented on-line. The material on Desktop Search is mostly in the form of user and administrator guides; this may be useful to a forensic investigator, since it describes when files are indexed, which may in turn identify Registry keys and other configuration information. In contrast, the information on ESE includes a complete reference guide to its Application Programming Interface (API).

Two substantial studies of Windows Search (then known as Windows Desktop Search) were carried out by Douglas (2009) and Gordon (2009); these are still relevant and contain a wealth of important material, despite recent updates to both ESE and Windows Search. Gordon's work is focussed on documenting the data artefacts used by ESE and Windows Search, including entries in the Windows Registry, a description of transaction files, and an overview of the data formats used to store database records. In contrast, Douglas' work shows how to extract Desktop Search data from the database programmatically, using the Microsoft ESE API.

Detailed documentation about the internal structure of ESE and the database schema used by Windows Search can be found in "Windows Search: Analysis of the Windows Search Database" and "Extensible Storage Engine (ESE) Database File (EDB) format specification", which are available from the *libesedb* project page (Metz, 2010a). Metz has also published an overview of this information (Metz, 2010b), which includes an example of an indexed Windows Mail message demonstrating how the original content is made available in the *autosummary* field.

From a forensic perspective, this documentation provides a wealth of information about the data artefacts created by Windows Search; however, investigators should be cautious about attempting a manual analysis of such data. In particular, the more extensive use of UNICODE in Windows 7 has resulted in many of the file names and paths being compressed to save space within the database, and as a consequence it is now much harder to identify known fields with string searches. (See Appendix A for an approach to this problem.) The Windows Search schema changes as Windows is enhanced, and so the schema described in these documents should not be regarded as definitive for any particular case; an investigator would be well advised to extract and refer to the actual schema for a database under investigation, rather than rely on prior documentation.

2.2. Data extraction tools

As noted above, Douglas (2009) provides the source code for a tool to extract search data using the ESE API, and although this software is no longer current, it does provide a rare worked example of how to use the API. Gordon (2009) describes the use of a viewer tool that was made available by the Microsoft Law Enforcement Portal; this has similar functionality to the tool developed by Douglas: it uses the ESE API to read schema and data entries from the database file. Another viewer has been developed by Woan (2008), which supports both native ESE queries, and schema for Windows Search and Windows Mail; this tool exports data to a comma separated text file for further analysis. A different approach has been adopted by Metz (2010a), who has developed a portable library for accessing ESE, providing a means of programatically accessing ESE and Windows Search data.

[2] *wdsCarve* is a stand-alone tool which runs on Windows XP, Windows Vista and Windows 7; it is available to forensic investigators and researchers from the first author, and is able to recover database records via the ESE API, in addition to record carving.

A related tool is *esentutl*, which is a standard part of Windows distributions, and provides a means for examining and recovering ESE databases, but not extracting actual data. In the context of a forensic workflow its primary uses are to identify required log files and to recover an inconsistent database file. It is necessary to use a version of *esentutl* which matches that of the database, since it uses undocumented features in the ESE API.

All these tools access an ESE database via its API, and rely on an intact database page structure to do so. The limitations of this approach are that there are no means of recovering deleted records from within the database, or of recovering data from fragmentary or badly corrupt databases. As discussed in the introduction, it is also possible to use a carving process to extract records from data fragments or inconsistent databases, which is described in this paper.

3. Background and terminology

Windows Search (formally Windows Desktop Search) uses the Microsoft Extensible Storage Engine (ESE) as a database server, which stores a search index in a single database file, *Windows.edb*. Each file that is indexed has a separate record within the database. The database uses log files to record ongoing changes prior to the database file being updated; this process is described in more detail in Section 5.3 since these log files contain useful historical evidence.

The indexer scope is the set of folders and files that Windows Search will index. The scope is explored further in the next Section (4.1); it typically includes all user folders, and excludes the Windows system folders. The indexer scope can be changed by a user explicitly using the indexer applet via the control panel, or by other actions on files. The indexer also has a number of attributes that control how indexing is carried out, they include attributes for file types and a single attribute that determines if encrypted content is indexed.

The data stored by ESE for Windows Search is defined in a *schema* (or data definition) which sets out all the data items that may be stored in a single database record. In conventional database terminology a complete single record may be described as a 'row', and individual data items as a 'column'; here the terms *record* and *data item* or *field* are preferred, since although over 300 fields are needed for the distinctive attributes of all possible record types (e.g. text, image, email…), each record contains only the few tens of data items relevant to its type.

One important field in the schema is *autosummary*, which is used to store an extract of a file's content.

One generic mechanism for recovering deleted data is the Windows 7 incremental backup service, known as *Volume Shadow Copy Service, Volume Snapshot Service, or VSS*; this records incremental periodic backups of in-scope volumes, which are presented to users as *Restore Points*. This paper uses the term *Shadow Copy* to refer to a retrievable restore point, even though the restore point is not usually a true full copy. In Windows 7 the main system drive is protected by shadow copies on a weekly basis; most user and system data are protected but the pagefile is excluded.

4. Indexed data

This section describes which files are indexed by Windows Search, the user actions that may give rise to indexing, and what data are recorded about each file. The scope of the indexer is described first, followed by an explanation of how the indexing varies with files placed in different parts of its scope, and with different file and indexer attribute settings; a specific experiment explores these aspects of its behavior with respect to encrypted folders.

4.1. *Indexer scope*

The information in this section summarizes information in Microsoft administration guides and other technical resources, and has been also confirmed by experiment.

Indexing of file metadata and content in Windows 7 depends on which files are within the scope of the indexer, file attributes that permit content indexing, and an index attribute that enables the indexing of encrypted content. The default scope of the indexer includes:

- The *Internet Explorer history*, which includes those items that would appear in the history pane of Internet Explorer; this is a far smaller set of records than could be obtained from analysis of the browser history files and caches (*index.dat* etc), but it does record the sites visited, and the date and time of the last visit. Recovery of deleted search database records may sometimes include previous record versions with the date and times of previous visits to the same site.

  More significantly for the forensic investigator, this history also records shortcuts to any computer files that are accessed via Windows Explorer, or via many standard file system actions (e.g. saving from an application). As a consequence this history provides metadata relating to files that are not normally within the indexer scope (see Section 7, below). This includes encrypted files, and may include files downloaded and then explicitly stored by browsers other than Internet Explorer (e.g. Chrome).
- *User emails*; because these are relatively short documents, it is likely that the search index contains the entire contents of users' emails.
- The *start menu* shortcuts to programs on the system, which may provide information about the system configuration and its history.
- *Offline files*, which are those designated as such for synchronization/offline management. The default behavior of Windows Search is to not allow remote indexing of network files, so shared network files cannot usually be added to a library, or selected for indexing in the indexer applet. However, a user may use the context menu (right click) to make a remote network drive available offline, in which case the local copy is within the indexer scope. The indexing of shared network files may be enabled by a group policy, although it would normally be imprudent for a system administrator to do so because of the network traffic that may result.
- All *user files*, which by default are those included in a *\Users* sub-directory, excluding application data (*\Appdata*). On

a live system the search interface restricts search results to the current user; however, the search database contains records for all users. An interesting by-product of this scope is that the user Registry hive (HKEY_CURRENT_USER, stored in ntuser.dat) is within scope, although its record in the search index provides little more than the last time of update.

The experiments described in this paper do not include applications (e.g. browsers, email) other than standard Microsoft components; however, correctly configured applications will store user data in the user's folders, and hence within the indexer scope, and saving files from non-Microsoft browsers may generate history records, as described above.

In general, the record for any file within the indexer scope is persistent within the database; if it is unavailable to the system when a user makes a query, then results are shown as unavailable, but the records are still retained.

Apart from adding a new location to the indexer scope by explicit action[3], or by inheriting an administration policy, a user may permanently add items to the indexer scope in other ways:

- By selecting 'add to index', when prompted to speed up a file content search;
- By adding a folder to a Library.
- By mapping a new device to a drive letter which is already within the index scope.

Windows 7 has a 'Library' construct which is essentially a set of links or shortcuts to popular parts of the user's file system. Adding a folder to a library automatically adds the folder to the indexer scope; however, removing a folder from a library does not remove the folder from the scope. Folders added in this way may include external devices and encrypted volumes, including those on encrypted disk partitions.

If a user adds a drive letter to the index scope, for example to search a datacart assigned to that letter, then that drive letter will remain in scope and any devices that are subsequently mounted to that letter will be indexed. For example, if a user mounts a non-Microsoft encrypted partition (e.g. TrueCrypt, BestCrypt or PGPDisk) to a drive letter within the index scope, this mistake is likely to result in full content indexing of the encrypted data, since the file attribute that allows content indexing is usually set by default.

All these mechanisms may lead to users unwittingly creating persistent records of remote, removable or encrypted devices.

Forensic investigators should also be aware that at present Windows Search does *not* index the content of files brought into scope via a reparse point; for example, if an encrypted TrueCrypt partition is mounted to a *\Users*\sub-folder, its contents would not be indexed, despite that fact that the sub-folder is within the indexer scope.

Detailed Registry analysis is beyond the scope of this paper, but for completeness, the most significant Registry settings are under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\:

- Attribute to control indexing of encrypted content:
  \Windows\Windows Search\Preferences
- Location of Windows.edb and other database files:
  \Windows Search\Databases\Windows
- Scope of indexer:
  \Windows Search\CrawlScopeManager\Windows\SystemIndex

4.2. *Indexer behavior*

As noted above, the behavior of the indexer is controlled by three main factors:

- If the data are within the indexer scope;
- A file attribute that permits content indexing (accessed via file properties);
- An indexer attribute that permits the indexing of encrypted content.

These may be modified by further attributes of the indexer that allow indexing of file metadata and file content by file type. The indexer is controlled by these attributes, regardless of whether the file is in a folder on a local, removable or encrypted drive. If such a file is indexed then the database records are retained even when the file is temporarily unavailable, due to the drive being disconnected, or the encrypted folder remaining locked.

Since there is an indexer attribute that is specific to encryption, the behavior of the indexer is more completely explored by evaluating its performance over a range of different encryption scenarios, as described below.

The ways in which a file or folder may be brought within the indexer scope are described in Section 4.1, above.

The file attribute is often set by default, otherwise if content indexing is required it must be set explicitly; the mechanisms that may bring a folder into scope (libraries and agreeing to indexing) do not set this attribute as a by-product. It may be set on the properties sheet of the file[4], inherited via a parent directory, or set by an administration policy.

The indexer attribute that permits indexing of the content of encrypted files is usually disabled by default. Attempting to set this attribute[5] results in the same warning as contained in the Microsoft documentation: that this should be set only if the index database is BitLocker protected, otherwise file content may be recovered from the index.

The behavior of the indexer was explored by experiment. A set of encrypted files was created, both in and out of indexer scope, and using folders encrypted using Encrypted File System (EFS) and also disk partitions encrypted using BitLocker, together with varying the index permission attributes on the folders and the indexer.

The resulting content of the index is summarized in Table 1; if the file is in scope, then metadata will always be indexed regardless of encryption. For EFS encryption both the file attribute that enables content indexing and the indexer

[3] The indexer scope is controlled via Control Panel ⇒ Indexing Options.

[4] File Properties ⇒ Advanced ⇒ Allow this file to have contents indexed in addition to file properties.

[5] Control Panel ⇒ Indexing Options ⇒ Advanced ⇒ Index Settings ⇒ Index Encrypted Files.

Table 1 – Data directly indexed from encrypted files. The text describes how file metadata may also be recovered from indexed Internet history records, providing information about files that are outside the indexer scope.

Index attributes set	EFS encrypted		BitLocker encrypted	
	In Scope	Out of Scope	In Scope	Out of Scope
None	Metadata	–	Metadata	–
File–Index Content	Metadata	–	Content	–
File–Index Content + Indexer–Index Encrypted	Content	–	Content	–

attribute that enables indexing of encrypted content must be set to allow the file content to be included in the index; however, for BitLocker, if the file attribute is set then content is indexed regardless of the indexer attribute.

The surprise here is that BitLocker content may be indexed regardless of the indexer attribute. Even if content is unavailable, metadata is available in a wide range of circumstances, and the resulting file and path names, access times, and other attributes such as picture resolution, may provide valuable information about the content of encrypted folders and volumes.

More significantly, this experiment highlighted the importance of another source of information about encrypted files. When a file on an encrypted volume is accessed via Windows Explorer, then an Internet history record is created in exactly the same way as if the user had accessed a remote Internet site; this history record is a 'shortcut' to the file and is separately indexed regardless of the encryption status, file attributes, or indexer scope of the original file. Within the index database there are therefore two sources of file metadata: the record of the original file and the record of the Internet history file which is created when the original file is accessed. This further increases the likelihood of recovering file information from encrypted or otherwise unavailable folders.

Note that the date-time information for these two file records is complementary: the metadata for a directly indexed file reflects the metadata stored in the file system, while the indexed history record includes the last access time, even if this has not been updated in the file metadata. As noted above, if an encrypted volume is not made available (e.g. a BitLocker partition that is not unlocked) then the data remains in the search index.

5. Recovery of deleted database records

When files are deleted from a system, the Windows Search service is notified and deletes the corresponding record in the search database. An important question for a forensic investigator is therefore: to what extent can records deleted from an ESE database be recovered? This section reviews why such data may remain available, and as a consequence where it may be found.

The information in this section was obtained by experiment, by directly writing, reading, and deleting sequences of database records using the ESE API, allowing direct manipulation of database contents. Each sequence of tests was reviewed by:

- Analyzing snapshots of *Windows.edb* taken after each action, to observe how database pages are managed;
- File Carving against the whole system image to determine where records may be found.

These experiments identify where deleted records may be discovered; the importance of the sources of data identified here will, of course, vary from case to case. The results of these experiments will follow a brief summary of the internal structure of an ESE database.

5.1. *Overview of the ESE database structure*

The unit of storage allocation in an ESE database is a page. Different versions of Windows Search use different page sizes; the current Windows 7 version uses a page size of 32 kbyte. Database records are stored within pages, and with the exception of long records (see Section 5.3) all records and references must fit within a single page.

ESE database records are stored in a structure known as a B-tree, the structure of which is shown in Fig. 1. The 'B' stands for balanced and this signifies that the path length from the root through internal nodes to every data record is the same; in other words finding any record via the tree structure involves the same number of decisions. The form of

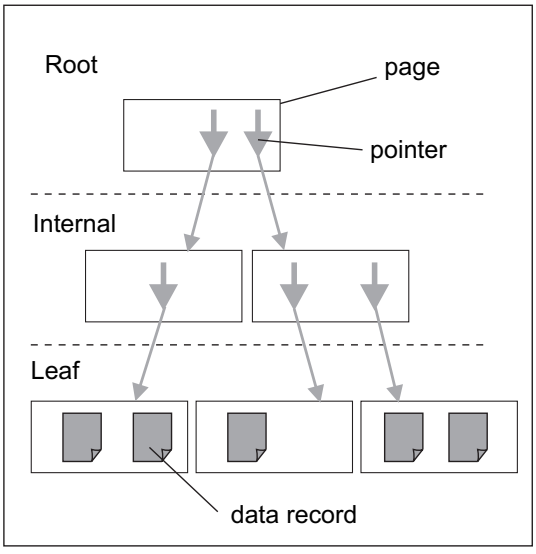

Fig. 1 – A Balanced Tree. A balanced tree consists of a root page, zero or more layers of internal pages, and leaf pages. Pointers in the root and internal pages form a tree structure allowing navigation to data records stored on leaf pages.

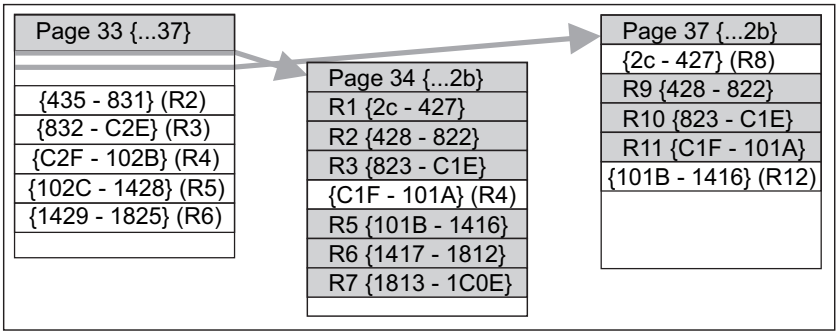

Fig. 2 – Adding and Deleting ESE Records. Three database pages after 12 records have been added and three (4,8,12) deleted. Shaded records are those that can be accessed by the database API, unshaded records are those that remain in unallocated space.

B-tree used in the ESE also allows records to be navigated sequentially, from page to page, so the order of primary keys in leaf pages is ascending.

In order to maintain this structure, when records are added that overflow a page, the page may be split, resulting in an additional pointer at the internal node, this may also be split, and so on. If the changes involve adding a new internal layer, the tree is adjusted to ensure that it remains balanced. The process of rebalancing is not carried out for every record entry or deletion, because root pages can store many records and intermediate pages can store a large number of pointers; however, a balanced tree is constantly re-structured, and the resultant copying can both preserve older copies of records in unallocated space, and also overwrite deleted records (see Section 5.2).

The B-tree shown here allows navigation to records using the primary data key, which in the case of Windows Search is the record identifier (ID). Generally record IDs are incremented every time a new record is added.

A database table may be thought of as a series of records (rows) with a consistent schema (set of column definitions), but actually several trees are used to implement and manage a table; these include a space tree which records spare pages, usually interspersed between leaf pages, and a *large record tree* which stores data items that are too big to fit in a standard record. An ESE table may also include secondary indexes.

5.2. *Database page and record management results*

This section reports the results of the experiments described above: adding and deleting database records.

Fig. 2 shows the state of a database table having added 12 records, then deleting three. The root page for this table is 33, and for the sake of experiment small database pages and records are used. The record ID (e.g. R1) is given for each record, together with the byte range occupied by the record (the bracketed hexadecimal range). Shaded records are accessible via the database API, in other words they remain part of the database, while unshaded records with a bracketed record ID remain in unallocated space, and can potentially be recovered by carving. A small amount of space at the start of each page is consumed by a header and short tag. Page 33 carries two short records which point to other pages in the structure.

When the first 6 records were added to this table, they were placed in page 33. Records do not flow continuously between pages, but must be held within a single page, so adding the 7th record required a new page. However, the tree structure requires that pages are either leaf nodes, which contain data records, or internal nodes, that provide the tree structure. As a consequence, when the 7th record was added it was necessary to copy all the existing the records to page 34, in order to assign page 33 to an internal node. (In this case 7 records will fit on a leaf page, but not a root page, because the latter has a bigger header.)

At this point page 33 contained a single reference to page 34. As more records are added, references to new leaf pages are required; these references are very short, so most of the original records are not overwritten. It is common, even in large systems that have been subject to change over a period, to find that a few very early records still remain in internal pages in this way.

When page 34 was full, page 37 was allocated as a new leaf, and a second reference added to page 33. The pages between 34 and 37 are allocated to a *space tree* which allows space for future changes within the table.[6]

Records 4, 8 and 12 were deleted from the database, and although they are no longer part of the database index, they remain in unallocated space.

If further records were added to the system starting from the state shown in Fig. 2, then the first new record overwrites the space occupied by the deleted record 12, and is numbered from the current highest ID: in this case ID 12 is re-used, as well as the space occupied by the old record 12. Microsoft documentation is unclear about ID re-use, and some commentators claim that IDs are not re-used; however, in these experiments, re-use of IDs within the record set (4 and 8 in this example) was never observed, whereas IDs deleted from the top of the set were consistently re-used.

The space occupied by deleted records 4 and 8 is not re-used until this part of the database is re-organized, for example to re-balance the tree.

In summary, these experiments show that mechanisms exist which result in complete, but unlinked, records persisting

[6] Recall that the leaf pages must be able to be read sequentially, as well as via the database B-tree.

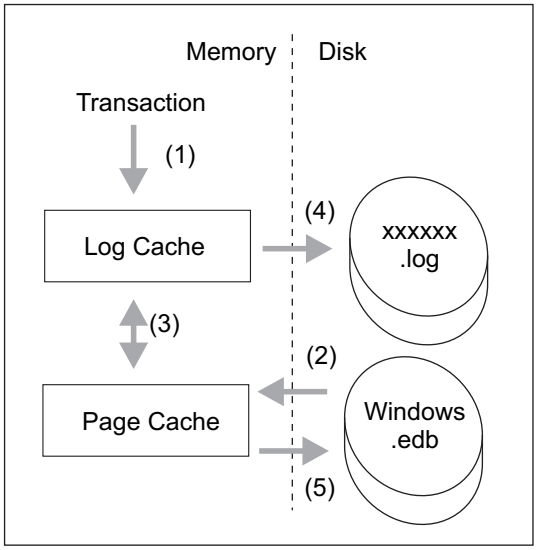

Fig. 3 – The Propagation of Transaction Data into Disk Files. Transactions and updated database pages are cached in memory, then written to disk later.

within the database in unallocated space: in space pages, in unallocated space on pages dedicated to internal pages, and in deleted records within leaf pages. Section 7 below provides a practical example of the recovery of such records.

5.3. *Transaction and cache management*

The Extensible Storage Engine is designed to process high transaction volumes and be recoverable from failures, such as a system crash while data are being written to disk. The use of log files to allow recovery from a crash are well known, but both the transaction logs and updated database pages are held in memory caches for immediate use, and written to disk in slower time (Baher, 2008). A typical transaction sequence is shown in Fig. 3.

An incoming transaction is first held in a memory log cache (1), then any necessary pages are brought into memory (2) and the transaction applied (3), and probably the cached log record updated to reflect the transaction[7]. At this stage the only record of the transaction is in memory, and other database transactions will make use of any cached database pages that have been updated in this way. The writing of cached log records to the log (4), and 'dirty' (i.e. modified) database pages back to the database (5) are not time critical, so these updates take place some time after the transaction. Writing to the log files has a higher priority, since if writing to the database file fails the database file can be recovered to a consistent state by replaying the transactions in the log files from a known checkpoint. The log cache is flushed to file on normal shutdown, whereas the database file (*Windows.edb*) may not necessarily be updated and may be left in a 'dirty' state. Both log and database records use the same format, so records from either can be recovered by carving.

The record carver was used to directly process a series of system images to identify other potential sources of search record data. Database records are found in:

- The database file: *Windows.edb*;
- Associated log files: *MSS.log and MSSnnnnnn.log*;
- Shadow copies: *\System Volume Information*;
- Unallocated space within the file system;
- System memory, if a memory image is available;
- Other memory records. e.g. *pagefile.sys*, crash dumps and hibernation files.

If access to the database via an ESE API is required, then because the *Windows.edb* file is likely to be in a dirty state, it is usually necessary to update the file to a consistent state before it can be accessed; this requires the relevant log files, and also a checkpoint file, as described in Appendix B.

Practical experiments confirm that writing to log files may be significantly delayed: a small number of transactions added to a small index (1000 s of records) produced no change in the associated log files, and daily images taken for a week after the transaction still showed no change, although the records could be carved from memory. Controlled shutdown of the system flushed the records to a log file, but 'pulling the plug' did not. This behavior occurs because there are number of internal log caches, and they are flushed only when a cache is full. If it is critical that all current transactions are gathered, then a memory image, or a controlled shutdown is required. If neither of these is possible, then recent records may be found in the pagefile.

Shadow copies, and records in file system unallocated space are discussed further below.

5.4. *Long data items*

ESE supports two long data types, binary and text, which are used to store data items between 256 and 2GByte long. These are stored in a separate B-tree and referenced by IDs in normal records; the general arrangement is shown in Fig. 4.

- Long data items have little structure, other than that provided by the ESE page; as a consequence, in the absence of the database B-tree and page structure, it is difficult to reliably carve such items.
- The ID used to reference a long data item is simply a 4-byte incrementing number; given the established re-use of record IDs it is possible that long data IDs are treated similarly. If long item IDs are re-used then there is no reliable way to link a long data item with a deleted record.

It is possible for several long items to be stored on a single page, and also possible that a long data item may require several pages; because of this the long item tree indexes long data items by both ID and byte offset.

Long data items are important from the forensic perspective, since the long binary is used by Windows Search to store *autosummary* data, from which file content may be recovered. Unfortunately this storage mechanism poses some difficulties for data recovery, in particular:

[7] This is not certain from documentation, but it is evident that there are two data schema used in the log files, one of which is written immediately, and the second of which is identical to the database record.

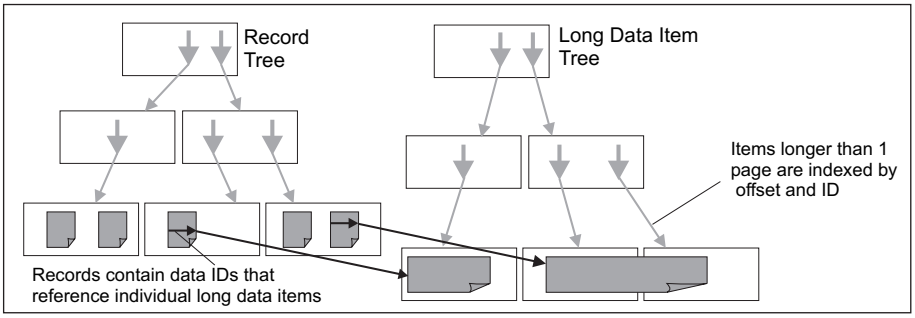

Fig. 4 – Long Data Items. A separate tree is used to store long data items, which are referenced by an ID in the parent record.

These factors suggest that it is likely to be unreliable, and hence inadvisable, to carve long data types, or to link any such types with deleted records for forensic purposes. As a consequence, the best strategy for the forensic investigator is to use a combination of approaches: to maximize the extraction of file content via the database API, then search for metadata that indicates the presence of other files, deleted records, and timeline events, using a record carver.

5.5. *Shadow or backup copies of the database*

The default shadow volume protection provided by Windows 7 includes the Windows Database Search folder in its scope; shadow copies therefore provide a potentially important source of historical search records. As noted above, carving from a disk image recovers some database records directly from shadow copies; however, if possible, recovering the related files then processing these (via API or carving) is likely to produce more complete results than simply carving from *\System Volume Information*:

- Recovery of the database file, *Windows.edb*, may also allow the recovery of file content; and
- Since the shadow volume is an incremental backup, it may fragment some records, and hence not all the records available from shadow copies may be carved directly from *\System Volume Information*.

The required files (*Windows.edb*, **.log* and *MSS.chk*) can be obtained by forensic recovery tools, or by building a clone or virtual copy of the machine under investigation, then recovering the earlier system state, for example by using the properties applet for the folder containing the search files to copy out previous versions. (See Titheridge, 2008; Hargreaves et al., 2008 and Crabtree and Evans, 2010 for methods in detail.)

5.6. *Records in unallocated space in the file system*

It is rare to find database records in unallocated space in the file system because of the way the *Windows.edb* file is managed; records were only discovered in unallocated space after experiments in which the search index was manually forced to re-build.[8]

[8] Control Panel ⇒ Indexing Options ⇒ Advanced ⇒ Rebuild.

Normally, as the database grows, more space is obtained from the file system, which is then managed by ESE as database pages, and is not normally released if the number of records in the database is reduced. Rebuilding the index releases some or all of the old file allocation for *Windows.edb*, and allocates a new default sized file, which is zeroed by ESE as part of its initialization. No old records are therefore found in a rebuilt database, but some may be recovered from unallocated space in the file system.

5.7. *Other data management issues*

ESE has a number of data management features that may restrict the data available to a forensic examiner, including page scrubbing and log file deletion.

The page-scrubbing feature is intended to overwrite records that have been deleted; however, there is no documentation or practical evidence to suggest that it is used by Windows Search. Microsoft Exchange uses scrubbing as part of the backup process: data are first written to backup, then pages in the live database are scrubbed.

There are a number of options for log file management. In practice the log files in Windows Search are deleted after they have been incorporated into the database; usually the only log files found in the Windows Search directory are those necessary to update the current *Windows.edb*.

A discussion of the forensic significance of the sources of data described here is given in Section 8, following a description of the carving strategy and a practical example of data recovery.

6. Carving strategy and reliability

The purpose of record carving is to extract database records from data which are not necessarily structured by ESE B-trees or pages. From the forensic perspective it is important that carved records can be relied on to be to be genuine database records, rather than random artefacts. This section introduces the database record structure, then describes how this facilitates reliable record carving.

6.1. *Record structure*

The structure of ESE records is flexible, allowing for fixed data items, which are normally present and fixed size, variable

data items, which are normally present but of variable size, and tagged items, which vary depending upon the record. In the Windows Search application, tagged items allow a large number of data items to be defined, only a few of which are used for any particular record; for example, a record of an image file may have a data item which records resolution, whereas one relating to an address book contact may record a telephone number. ESE allocates space to store tagged data items only if they are required for a particular record.

An overview of the record structure is given in Fig. 5. The header ends with three fields that define the number of fixed data items, the number of variable data items, and an offset to the first field in the variable record space. Given the number of fixed items and their order, the offset to the variable item space can be calculated from the database schema; however this is also included in the header, presumably to improve record navigation speed.

The variable item space is not described in detail here, since it is not used in the current Windows Search schema, although it appears in earlier versions. It includes a series of offsets that define the lengths of individual data items, followed by the data items themselves.

The tagged item area is divided into two parts, the tags themselves, and the corresponding data items. Tagged items are present only if they are required for the specific record; however, most Windows Search records have in excess of 20 tagged data items since the tagged fields include generic items such as path, folder and file names. Each tag specifies the ID of the data type, which determines how the data should be interpreted, and the offset at which the data item begins. Depending on the ESE version and page size, tags may also include flags, as may the first byte of the data item; generally these flags specify the format in which the item is held, for example if the data item is replaced by a reference to an item stored in the long page tree.

6.2. *Carving strategy*

The navigation information built into the record structure (offsets, etc) can be checked against the data specification for the application, in this case the Windows Search schema. For example, given the number of fixed records specified in the header, their total length can be calculated from the schema and checked against the offset which is also in the record header. Similarly, the IDs of tagged data items must be present in the schema, and their lengths must be consistent with the range of lengths defined for those items.

<table>
<tr><td rowspan="4">Header</td><td>...</td><td></td></tr>
<tr><td>ID Last Fixed Item</td><td>1 byte</td></tr>
<tr><td>ID Last Variable Item</td><td>1 byte</td></tr>
<tr><td>Offset to Variable</td><td>2 bytes</td></tr>
<tr><td colspan="3">Fixed Data Items</td></tr>
<tr><td colspan="3">Variable Data Items</td></tr>
<tr><td rowspan="3">Tags</td><td>Data Type ID</td><td>2 byte</td></tr>
<tr><td>Offset to Next Item</td><td>2 bytes</td></tr>
<tr><td colspan="2">etc</td></tr>
<tr><td rowspan="2">Tagged Data Items</td><td>Data Type ...</td><td>...</td></tr>
<tr><td colspan="2">etc</td></tr>
</table>

Fig. 5 – Record Structure Overview. Three data areas (fixed, variable, and tagged) are framed with content and navigational information.

The carving process is divided into two parts: a primary check and a secondary check. The primary check is for consistency between the schema and the three values at the end of the header; specifically that:

- The fixed record index is valid.
- The variable record index is valid.
- The offset is consistent with the fixed records.

Given the current Windows Search schema, if the data to be carved were random, then the expected false alarm rate would be the product of the likelihood of selecting each of the three fields at random. There are 8 choices for the number of fixed records, 1 possible value for variable records, and 1 for the offset given knowledge of the fixed records. The primary error likelihood is therefore[9] $8/256 * 1/256 * 1/65536 = 1.86 * 10^{-9}$.

The reliability of the forensic analysis does not depend on this figure, because of the additional checks described below; however, its efficiency does, and although practical (i.e. non-random) data patterns produce much higher primary false alarm rates, the carving process remains sufficiently efficient.

The secondary process is to ensure consistency between the schema and the tagged fields while extracting the data. In Windows Search there are currently 389 possible tag data types, and the maximum uncertainty in data length is between 1 and 255 bytes. The maximum possibility of an error from a single tag field is therefore: $389/65536 * 255/65536 = 2.3 * 10^{-5}$.

The total likelihood of carving a record in error is this figure to the power of the number of tags in the record, which normally exceeds 20. Even allowing that practical machine data is far from random, and that these figures are slightly simplified because of the presence of flags within tags in earlier versions of Windows Search, it is clear that any carved record with even a small number of tagged data items is very unlikely to be carved by chance.

These calculations relate to the correct carving of ESE records; *wdsCarve* is also able to carry out further consistency checks against the Windows Desktop schema.

7. A practical example

To provide a concrete example of what data recovery is possible, this section describes a simple experiment where a collection of files are indexed, deleted, and a range of record recovery strategies attempted. The files were generated by browsing the Internet using Internet Explorer, in a system which had previously indexed approximately 18,000 user files. The browsing was carried out over three days, after which the index database was allowed to stabilize until 'Indexing Complete' was reported

[9] The divisors of 256 and 65536 are the number of possibilities for a byte and word respectively.

by the indexer applet. The files associated with the index were copied out, recovered using *esentutl* and 281 'iehistory' records were extracted as the reference set.[10]

The Internet History was then deleted via the browser tool menu, and the indexer quickly reported a corresponding decrease in the total number of items indexed. The system was then shutdown normally, and the database files recovered. In this case a normal shutdown was chosen since the cache is flushed to the logs on shutdown and this ensured the availability of log files, allowing evaluation of the extent that bringing the database to a consistent state removed previously deleted data.

Two sets of files were recovered: those that could be obtained directly from the image, and those that could be recovered via a shadow copy, the restore point for which was taken close to the end of the browsing session to maximize the possibility of recovery via this mechanism.

The results for record recovery are given in Table 2. Carving data from the *Windows.edb* file as recovered allowed recovery of all the index records relating to the deleted Internet history; it is clear that this file had not been updated since the Internet history was deleted. When the file was manually recovered using *esentutl* then access via the database API resulted in only a single remaining history record: essentially the whole Internet history had been deleted from the database. However, it was still possible to carve a high percentage of the original records from this file, so in this case, re-organizing the database tree had resulted in the loss of only 7% of the deleted records.

Since the shadow copy had been taken immediately before the Internet history was deleted, recovering files from the shadow, recovering the database, and then accessing via the API recovered 100% of the deleted records, as was expected.

Another approach to recovering lost records is to simply carve from log files, rather than the database; these results show that this is relatively successful. Carving from log files recovered via shadow produced 76% of the original records, whereas carving from those left after deletion recovered only 25%. This is because log files are produced in sequence, and deleted after they are no longer needed; the first two logs in sequence had been deleted by the time that the history was deleted, but could be recovered via the shadow copy mechanism.

These results should not be taken as an expectation of the level of recovery that could be expected in practice, since the timing of events of interest, seizure, and shadow copies, will significantly change the likelihood of record recovery. However, it does demonstrate three relatively independent strategies for the recovery of deleted search records:

- *Using the database API to read files from a recovered Windows.edb file which is obtained by restoring a shadow copy*. Assuming the availability of a well-timed shadow copy, using the API has the benefit of providing content summaries (*autosummary*) as well as metadata.
- *Carving data directly from a recovered Windows.edb file*. Even files that have been updated, either recovered manually or updated by the system, retain records that may be obtained by carving.
- *Carving from a collection of log files*. This may be particularly effective in obtaining a system history if a full set of log files can be recovered via Shadow copies.

[10] The process for file preparation and recovery is given in Appendix B.

Table 2 – Recovery of deleted records.

Process	Records	% of Total
Files remaining after deletion		
Windows.edb via carving	281	100%
Recovered *Windows.edb* via API	1	0.3%
Recovered *Windows.edb* via carving	261	93%
*.log files via carving	70	25%
Files recovered from shadow copy		
Recovered *Windows.edb* via API	281	100%
*.log files via carving	214	76%

8. Discussion

The results presented in this paper are indicative of the likely behavior of Windows Search and ESE. The search index provides a source of information about parts of the system that may otherwise be unavailable to an investigator, and about the history of the system and the user's behavior. This section will review some important issues from the perspective of a forensic investigator.

8.1. *Unavailable files*

Files may be unavailable to an investigator because they are encrypted, or because they are on a device which is unobtainable, such as a remote network drive or a removable hard disk. Windows Search may retain an index of these data sources even when they are not available to the system, potentially providing a unique source of evidence.

Windows protects the content of files (as opposed to the file metadata) from inadvertent indexing by requiring a file attribute to be set to permit content indexing; in the case of encrypted files an indexer attribute must also be set. However, there are many situations in which file content or metadata will become available to an investigator. Importantly, users may not be aware of the consequences of improving their search performance or making shared network folders available offline, or that deleting folders from libraries does not remove them from the indexer scope. Users may also re-use drive letters in such a way as to make encrypted content available within the search index (see Section 4.1).

It is notable that for encrypted partitions using BitLocker, and for non-Microsoft encryption products that are inadvertently brought within the indexer scope, the indexer encryption attribute fails to protect the content from indexing.

A further important source of metadata about unavailable files is the Internet history, which records files as they are accessed. It is important to note that it is the history that is indexed, not the original files or folders, so the time date information has a different significance, but nevertheless this provides a further source of information about otherwise unavailable files, and the user's behavior in accessing those files.

8.2. Deleted files or folders

Records relating to deleted files will quickly be removed from the index that is available to a user, and eventually removed from the database itself. The rate at which these records are actually destroyed by re-organization of the database obviously depends upon the rate of change in the system; in the experiment reported in Section 7 approximately 7% of the carveable deleted records were lost when the database was recovered; the likelihood therefore is that recently deleted files can be recovered, but deep histories are unlikely to persist in any volume within the database.

8.3. Data recovery strategy

The best strategy for recovering database records relating to deleted files will be case-dependent; however, the example in Section 7 demonstrates that under favorable circumstances it is possible to extract a high percentage of deleted records by three different strategies: using the database API to obtain data from files retrieved from shadow or backup records; carving deleted records from an existing database; or carving from recovered log files. The relative value of these three approaches will vary from case to case, depending upon the availability of previous database copies and the time period of interest.

For most investigations, extracting records via the database API will be the method of first choice, since it allows the possibility of recovering file content as well as metadata. The source database can be the current system, or old versions retrieved via shadow or backup recovery mechanisms. Note that accessing data via a recovered database can occasionally reveal records that are not available by carving, since the recovery process is able to read records in the log files that are not in the database record format.

Record carving may be the priority if the investigator is concerned to probe very recent activity – perhaps in the 30 min prior to seizure, where the only records may be in memory or pagefiles, or where there is suspicion that significant file deletions, perhaps including forensic overwriting, have taken place. Another situation where record carving may be indispensable is if the Windows Search index has recently been rebuilt, in which case carving from the disk image may retrieve records from unallocated file system space.

8.4. Seizure considerations

Delayed writing to log and database files (see Section 5.3), means that the database file is never guaranteed to be in a clean state; however, cache data are flushed from memory into the log files by executing a normal system shutdown. ESE is designed to survive system crashes, so if the seizure method is to 'pull the plug' then there is a small risk that *Windows.edb* will be impossible to recover, but more likely that recent data will be lost.

One other seizure opportunity is to obtain a memory image; given such an image it is possible to carve current database records from the in-memory copy of ESE, thus preserving what would otherwise be lost.

Almost every aspect of the search system can be configured by the user or extended by a programmer, including file types, locations, and indexer scope; in a non-standard installation an investigator may need to review the Registry settings to determine the location of the index database and its log files.

8.5. Future work

Further work is needed to understand data formats used in log files, since it is clear from compressed field searches that index data are first written to these files in a format which is different from a standard database record, and is hence not recovered by the current carver.

It may also be possible to revisit the feasibility of carving content; the conclusion here, that such carving could not be reliably linked to file metadata, is likely to be true in general, but a full understanding of how and when log data IDs are reused may allow the recovery of some data that remains in place within the database after a recent deletion.

Finally, ESE is used for other important applications, including Microsoft Exchange; *wdsCarve* can accommodate different schema, so extending this work to other applications is a potentially worthwhile avenue of research.

9. Conclusion

This paper has reviewed the recovery of data from Windows Search from the perspective of a forensic investigator. Obtaining records from the search database, either via carving or via the Extensible Storage Engine API, provides a potentially valuable source of evidence about files or folders that are inaccessible because they are encrypted, or on unavailable removable storage.

Important new results concern the feasibility and value of record carving: finding database records in unstructured data. Carving allows the discovery of records from a range of new sources, including corrupt database files, log files and memory. Importantly, carving is shown to provide new strategies for the recovery of deleted database records, providing three relatively independent strategies to the forensic investigator for the practical reconstruction of historical information about files and folders in a system.

Appendix A. String searching

Most of the strings stored in ESE records in Windows 7 are compressed; however, short strings use a simple form of compression which allow investigator to carry out string searches, provided that the required text occurs at the start of the string. All that is needed is to search for the compressed text. This attachment gives a simple example to make the compression process explicit.

For example, if the required text is '/2010', then the process is as shown in Table 3. The ASCII characters are regarded as 7-bit values by truncating their most significant bit, then packed into an 8 bit stream as shown. To find the string '/2010' the investigator would search for the hex values '2F 19 2C 06'.

In Windows 7, most of the paths are represented in UNICODE, and there are two options for compressing this text; if the upper byte of the 16 bit UNICODE representation is zero throughout the string, then the string may be regarded as ASCII and 7-bit ASCII characters compressed, as described above.

Table 3 – Simple String Compression used for short strings in ESE.

Letter	Hex	7-bit binary, and split	Reassembled into bytes	Compressed Hex
/	2F	010 1111	0010 1111	2F
2	32	011 001|0	0001 1001	19
0	30	011 00|00	0010 1100	2C
1	31	011 0|001	0000 0110	06
0	30	011 |0000		

Otherwise the 16 bit UNICODE character is converted into two 7-bit characters by truncating the most significant bit of each byte, then the compression above applied. For example, 'Defcon' would be compressed into '44 40 19 60 06 8C 01 …'.

This is not an exhaustive account of compression in ESE, and does not apply to long text, it does however allow a forensic investigator to map many of the database fields manually, if that is necessary.

Appendix B. Processing via the database API

This section describes in detail how processing via the database API was carried out.

In most practical cases the database file (*Windows.edb*) will be in a 'dirty shutdown' state; in other words not all the current pages from memory will have been flushed to disk, and it will first need to be brought to a consistent state before it can be interrogated via the database API.

Appendix B.1. Required files

The normal location for the Windows Search Database files is:

%SystemDrive
%\ProgramData\Microsoft\Search\Data\Applications\Windows

This location can be re-assigned by policy or by the user (see the Registry keys identified in Section 4.1).

The files that must be retrieved from the image are:

- The database file (*Windows.edb*).
- Any log files (*MSS.log* and *MSSnnnnn.log* - where nnnnn is a hexadecimal sequence number).
- The checkpoint file (*MSS.chk*).

MSS.log is the current log file, in other words the file that is currently being written with log records. The *esentutl* utility (see below) may reference this file by the next number in the ascending series of log numbers. This is the number it will be assigned when full, at which time a new *MSS.log* will be started.

Appendix B.2. Recovering the database file

This requires the Microsoft *esentutl* utility, which is a standard component of Windows 7, and is run from the command line.

The first stage is to check if the database file needs to be updated, and if so that the required log files are present:

esentutl -mh <path to database file>

This provides a metadata dump from the database, of which two lines are of particular significance:

State: Dirty Shutdown
Log Required: 192–195 (0xc1–0xc3)

If the state is given as 'Clean Shutdown' no pre-processing is required; usually it is 'Dirty Shutdown', meaning that the *Windows.edb* file must be brought to a consistent state before it can be read via an API.

The hexadecimal numbers of the required logs specify the names of the required log files: *MSS000C1.log*, *MSS000C2.log*, together with *MSS.log* in this example. (Note the comment above: *MSS.log* is the latest log, in this case *000C3*.)

The *esentutl* recovery process is then used to bring the database to a consistent state. Assuming that *esentutl* is run from a directory containing *Windows.edb*, the necessary log files, and the checksum, then the command line is:

esentutl -r MSS -d

Assuming that *esentutl* reports success, the *Windows.edb* file may now be accessed via the database API.

Appendix B.3. Obtaining database records

Given a clean *Windows.edb*, then any of the programs described in Section 2 may be used to obtain records. The authors used *wdsCarve* as follows:

wdsCarve -d -m -c -a -y < format options > <Windows.edb path >

This outputs record into comma separated formatted files for further analysis; four files are output:

- *CurrentData.csv* (-d) The current database contents, including *autosummary* content, that can be retrieved via the database API.
- *CarvedData.csv* Data carved from anywhere in the *Windows.edb* file (-a) which is not an exact duplicate of a record retrieved via the API (-y).
- *Metadata.csv* (-m) The data schema for this database.
- *wdsCarve.log* A processing record, which includes an MD5 hash for the input and output files (-c).

Appendix C. Processing by file carving

File carving may be used to obtain very new records (e.g. from a pagefile), and records that have been removed from the database because the associated files have been deleted. However, as noted in the main text, carving is unable to relink the *autosummary* field with the record, so it will provide file metadata, but not content.

The file carver adds a final field to the output record which provides the source byte offset of the first fixed data item in the carved record, to allow manual analysis if required.

Appendix C.1. Recommended files

Carving can be carried out against any form of image; the most effective approach is to use standard forensics tools to first recover from the image any current or deleted files that are likely to contain Windows Search records. These are:

- The database file: *Windows.edb*;
- Database Log files: *MSS*.log*;
- The system pagefile: *pagefile.sys*;
- A memory image, if available.

It may be worth carving from a whole disk image if the Search index has recently been rebuilt; the gatherer time in the database record is the time that the database entry was made, so it is possible to identify a recent rebuild from the earliest time in the current database.

Appendix B described carving from *Windows.edb* after its recovery to a clean state; this has the benefit of allowing the carver to identify and reject duplicate records that have already been extracted via the API; however, carving from this file before it is recovered maximizes the possibility of obtaining deleted records; the database update associated with recovery will reduce the number of deleted records available (see Section 7).

As noted in the text, the set of logs associated with Windows Search are named *MSS.log* (the current log) and a historical set named *MSSnnnnn.log*, where nnnnn is a hexadecimal sequence number. Log files are deleted when the *Windows.edb* file has been updated beyond their individual scope, so there is benefit in attempting to recover as much of the log sequence as possible, by using forensic tools to recover deleted files, and recovering shadow copies where possible.

The most recent database records may exist only in memory, so if very recent activity is to be investigated, it is worth carving from a memory image, and/or the pagefile.

Appendix C.2. File carving

The options available in *wdsCarve* are beyond the scope of this paper, but a typical command used for carving is:

wdsCarve -r -c <format options> <working directory> <source file>

The -r option is 'recovery' mode, which assumes that the source file does not support the database API or even necessarily a page structure: it is either a dirty database file or some other form of file such as a log file, memory, or disk image. However, the carver needs to obtain a correct schema in order to carve records, since there are several generations of schema between Windows XP, Vista, and Windows 7, all of which can be carved. This mode therefore requires a clean *Windows.edb* file from the same Windows build to be present in the working directory, named *Reference.edb*; the data content of this file is irrelevant, the carver simply needs to read the data definition to configure the record carving program.

The output file (*CarvedData.csv*) will be placed in the working directory, together with a log file (*wdsCarve.log*) which may include an MD5 hash for input and output files (-c option).

REFERENCES

Baher M. Who said that transaction goes from logs to db, http://blogs.technet.com/b/mbaher/archive/2008/01/22/who-said-that-transaction-goes-from-logs-to-db.aspx; 2008 (accessed August 2010).

Crabtree J, Evans G. Reliably recovering evidential data from volume shadow copies in windows vista and windows 7. Tech. rep. QCC Information Security; 2010.

Douglas J. 2009. Forensic artefacts present in microsoft windows desktop search. Master's Thesis, Cranfield University.

Gordon JM. 2009. A forensic examination of windows desktop search (version 3). Master's Thesis, Cranfield University.

Hargreaves C, Chivers H, Titheridge D. Windows vista and digital investigations. Digital Investigation 2008;5(1–2):34–48.

Metz J. libesedb, http://sourceforge.net/projects/libesedb/files/; 2010a (accessed August 2010).

Metz J. Windows search forensics, http://www.forensicfocus.com/windows-search-forensics; 2010b (accessed August 2010).

Microsoft. Extensible storage engine reference, http://msdn.microsoft.com/en-us/library/ms683072(v=EXCHG.10).aspx; June 2007 (accessed August 2010).

Microsoft. Windows search it guides, http://technet.microsoft.com/en-us/library/cc771203(WS.10).aspx; June 2008 (accessed August 2010).

Microsoft. Windows search, browse, and organize administrator's guide, http://technet.microsoft.com/en-us/library/dd744681(WS.10).aspx; September 2009 (accessed August 2010).

Titheridge D. 2008. Microsoft windows vista registry. Master's Thesis, Cranfield University.

Woan M. Esedbviewer, http://www.woany.co.uk/esedbviewer/; 2008 (accessed August 2010).