Article:
Chivers, Howard Robert (2014) Private browsing: a window of forensic opportunity. Digital Investigation. pp. 20-29. ISSN: 1742-2876

https://doi.org/10.1016/j.diin.2013.11.002

Private Browsing: A Window of Forensic Opportunity

Howard Chivers

Department of Computer Science, The University of York, Heslington, York, YO10 5GH, UK

Abstract

The release of Internet Explorer 10 marks a significant change in how browsing artifacts are stored in the Windows file system, moving away from well-understood Index.dat files to use a high performance database, the Extensible Storage Engine. Researchers have suggested that despite this change there remain forensic opportunities to recover InPrivate browsing records from the new browser. The prospect of recovering such evidence, together with its potential forensic significance, prompts questions including where and when such evidence can be recovered, and if it is possible to prove that a recovered artefact originated from InPrivate browsing. This paper reports the results of experiments which answer these questions, and also provides some explanation of the increasingly complex data structures used to record Internet activity from both the desktop and Windows 8 Applications. We conclude that there is a time window between the private browsing session and the next use of the browser in which browsing records may be carved from database log files, after which it is necessary to carve from other areas of disk. It proved possible to recover a substantial record of a user's InPrivate browsing, and to reliably associate such records with InPrivate browsing.

Keywords: Digital Forensics, Internet Explorer, Microsoft Windows, Database, Carving

1. Introduction

The release of Internet Explorer 10[1] marked a significant change in how Internet history and cache data are stored within the file system; the binary historical formats which have been widely documented in the forensic community (e.g. (Jones, 2003)) were replaced by a high performance database technology known as the Extensible Storage Engine (ESE). This database is used to support a range of other applications, including Windows Search, and was the subject of a previous paper in which we described the results of carving for deleted ESE database records from the Search Database (Chivers and Hargreaves, 2011). The carving tool is now known as *ESECarve*[2] and has subsequently been used to assist a number of real investigations.

InPrivate Browsing is an Internet Explorer mode which is launched by the user in a separate browsing window; the claim is that this mode *"prevents local storage on your computer"*(Microsoft, 2012). The prospect of evidential recovery from private browsing is of considerable forensic interest, and several researchers have reported using string searches to identify artifacts of interest; others have used ESECarve to survey residual browsing histories and suggest that such evidence is recoverable (Malmström and Teveldal, 2013).

The prospect of recovering evidence from InPrivate browsing prompts questions, including when such evidence can be recovered, the implications for seizure tactics, where the evidence can be found, and if it is possible to prove that a recovered artefact originated with InPrivate, as opposed to normal, browsing. This paper reports the results of experiments which answer these questions, and also provides some explanation of the increasingly complex data structures used to record Internet activity from the desktop and by Windows 8 Metro Applications. Results of forensic interest include:

- InPrivate browsing artifacts can be positively identified using the *Type* field in cache content records.
- Pull-the-plug seizure may allow the recovery of InPrivate browsing records from the database file (*WebCacheV01.dat*); however, it may also result in a database that cannot be recovered for use with application interface-based tools because log files have not been completely written to disk.
- The window of opportunity for the recovery of InPrivate artifacts from database log files extends to the next time the browser is opened for use. During this window substantial recovery is possible, afterwards these data are securely deleted.
- Browsing evidence may also be recovered from areas of disk apart from normal database files and logs; this may persist for some time.
- The table structure within the database includes separate records for applications, allowing some fine grain distinctions to be made about the use of the computer.

Email address: hrchivers@iee.org (Howard Chivers)

[1] We acknowledge Microsoft copyright in terms used in this paper to describe Microsoft products, including: Windows, Windows 8, Metro Application, Internet Explorer, Windows Search, InPrivate Browsing, Extensible Storage Engine (ESE), and esentutl.

[2] ESECarve is available from the author for forensic investigation, education and research

The remainder of this paper is organized as follows: Section 2 briefly describes the Extensible Storage Engine and Data Storage in HTTP/HTML, both of which are needed to understand the descriptions of database behaviour and browser artifacts that follow; this is followed by a review of publications related to private browsing. Section 3 describes how the experiments used to determine browser behaviour were conducted. The next sections present detailed results; section 4 describes the files that support Internet Explorer and how the database tables are structured, section 5 describes the conclusions of experiments to determine if InPrivate browsing records can be recovered. Findings are further discussed at section 6 and the paper is concluded in section 7. Appendixes describe the restoration of a database to a clean state, and record carving using *ESECarve*.

Terminology. This paper uses the term 'record' to mean a single database record or row. Browsing records include a URL with an associated date and time. They document a single Internet action; examples include a cached response to a HTTP request, a download, a history record of a visit to a domain, or the storage of a cookie. The term 'browsing record' here should not be taken as an implication that it originated from human action.

2. Background

2.1. Extensible Storage Engine (ESE)

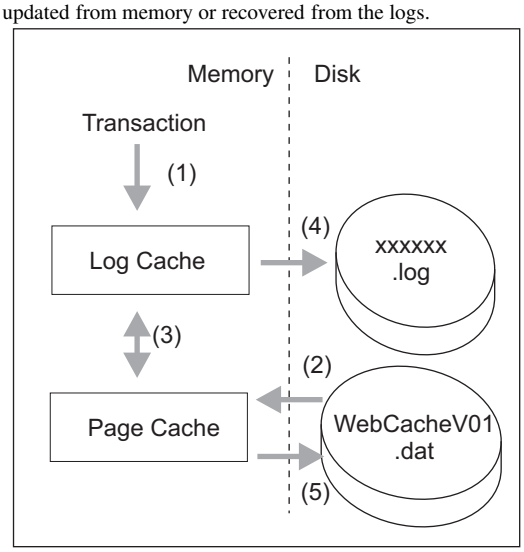
The Extensible Storage Engine is documented on-line by Microsoft (Microsoft, 2013), and details of its internal structure have been published by Joachim Metz (Metz, 2010). A previous paper (Chivers and Hargreaves, 2011) provides an overview of the database and the reliability of records recovered by carving. This section briefly describes transaction processing, as a background to why database records are often found in log files or in cached memory such as the *pagefile*.

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. A typical transaction sequence is shown in figure 1, with the file names currently used by Internet Explorer 10.

An incoming transaction is first held in a memory log cache (1), then any necessary database pages are brought into memory (2) and the transaction applied (3); as soon as possible the updated database record is written to the log file (4). Eventually the database file is updated with the page which contains the new transaction. A database whose file has not been fully updated is known as *dirty*. On a normal shutdown the log cache is flushed to disk, whereas the database file (*WebCacheV01.dat*) may not necessarily be updated and may be left in a dirty state.

If the database is dirty it must first be *recovered* (in ESE terminology), before it can be accessed using the database application interface. This process recovers the database to a consistent state by replaying log transactions from a known checkpoint. The checkpoint is stored in a *V01.chk* file and the logs are recorded in files numbered in a hexadecimal sequence (e.g.*V010009.log, V010000A.log*) together with the current working log (*V01.log*). When the current working log

Figure 1: *The Propagation of Transaction Data into Disk Files.*Transactions are cached in memory and written quickly to log files; the database file is subsequently updated from memory or recovered from the logs.



is full it is renamed to the next name in the hexadecimal sequence, and a new V01.log file is created. Logs that are no longer needed are deleted.

Both database and log records use the same record format, so records from either can be recovered by carving.

It is evident from this process that database records are found in memory and perhaps therefore in the pagefile, in log files, and in the database file. The action of allocating and freeing files for logs may also leave records in unallocated or slack space in the file system. Because this is a high performance database there are deeper layers of caching and lazy writing which may delay the writing or deletion of records.

Some forensic tools require the database to be recovered to a clean state before it can be processed via the database application interface; the *esentutl* utility can be used for this purpose, and for investigating the table structure of an unknown ESE database. This is described in section Appendix A. Forensic practitioners should also note that:

- Some forensic tools automatically recover the database to a clean state before retrieving records via the application interface; they require the .chk, and .log files as well as the database file.
- The recovery process will delete as well as add database records; it is often more productive to carve from a dirty database rather than a recovered copy.
- Pull-the-plug seizure may result in an unrecoverable database because logs files have not been flushed to disk from database caches. This occurred in approximately 40% of the experiments reported here.

2.2. HTTP/HTML Data Storage

Certain data types originating from HTTP protocol transactions or from scripted actions in HTML pages are stored

separately in the file system and result in distinctive database records: *cookies*, *Web Storage* and *Indexed Database* storage.

Cookies are a well known mechanism for maintaining state between HTTP protocol exchanges, for example to allow a web server to identify a request with a previous transaction (Barth, 2011). Cookies are name:value pairs which are returned in HTTP requests to the domain which supplied them. The value is a short text string, which is often a key to information held at the server, such as a user's session information.

The other two data types are managed by scripts running on web pages . *WebStorage* (W3C, 2013b), known as *DOMStorage* by Microsoft allows the storage of name:value data on the client. It is more flexible than the cookie mechanism in the assignment of access rights, lifetime, and size, allowing the storage of data objects of up to 10MByte. DOMStorage may be used for client-side storage, such as cached mailboxes or other files, as well as for session continuity. From the forensic artefact perspective DOMStorage can be thought of as a separately managed data cache, the difference being that DOMStorage is managed by scripts on web pages, unlike cache content which is managed by the browser.

The third type of data storage is the *Indexed Database API* (W3C, 2013a), also known as *IndexedDB*. This provides web pages with the ability to store large arbitrary objects which are flexibly indexed; for example, keywords may be associated with documents and allow the retrieval of a set of documents by specifying a key. This more flexible storage allows the caching of complex objects such as scenes from online games, and also allows them to be viewed offline. This is a relatively new feature which is part of HTML5 and has yet to achieve widespread use. This storage will not be discussed further in this paper, since it stored in a separate database, *Internet.edb*[3]

2.3. Protected Integrity Levels and Application Containers

Browsing records are separated into different database tables by data type (e.g. cookie, DOMStorage, URL cache, download, cached page content) and also by integrity category. Earlier versions of Internet Explorer divided data into two integrity categories, one for sites accessed in protected mode (Internet and Restricted network zones), and the other for data loaded from Trusted Sites, the Local Intranet or the Local Machine. This separation is maintained in Internet Explorer 10; however, the *Metro Applications* introduced in Windows 8 execute in separate *AppContainers*, each of which is a separate integrity partition. This separation can be confusing for a user; for example the desktop Internet Explorer and will not share cookies with the same browser launched as a Metro Application.

For the forensic practitioner the distinction between integrity containers may add information about the user's behaviour, and is needed to understand the large number of Internet caches in the file system.

[3]The *Internet.edb* database is usually located at: *\Users\%USERPROFILE %\AppData\Local\Microsoft\Internet Explorer\Indexed DB*. A demonstration of pictures that use Indexed Data can be seen at *http:\\snapyx.azurewebsites.net* - note the large difference in time to load the images between the first and subsequent visits.

2.4. Related Work

Some general studies have been carried out on private browsing. Aggarwal et al. (2010) provide a thoughtful analysis of threat models and what constitutes private browsing and survey four different browsers. They find weaknesses in all but do not report the possibility of data retention in Internet Explorer 8. In contrast Ohana and Shashidhar (2013), also working with Internet Explorer 8 and other browsers identified recoverable evidence in unallocated and slack space. Similar results were obtained by Said et al. (2011) who noted evidence on disk and in physical memory for InPrivate browsing in Internet Explorer 8.

Mahendrakar et al. (2010) review artifacts in memory and ways they can be obfuscated or recovered; they suggest a tool to reassemble files from memory blocks. The approach taken here is not to attempt reassembly of database files, but to reliably carve individual records. (Satvat et al., 2013) is a more recent paper, but despite quoting Internet Explorer 10 as the target, they describe the analysis of Index.dat files which are associated with earlier versions of the browser. The paper provides interesting suggestions for active attacks; however, these are less relevant to the normal forensic process.

Malmström and Teveldal (2013) experimented with Internet Explorer 10 and suggest that residual evidence may be recoverable following InPrivate browsing; they also document some aspects of the Containers table. This work concludes that the positive identification of InPrivate browsing records is still an open question, and prompts further questions about the extent and reliability of the recovery of such records.

3. Method

The results reported here were obtained using Windows 8 Pro and Internet Explorer Version 10.0.9200.16384. The operating system was run within a VMWare virtual machine, allowing a complete reset of the system state when necessary. To simulate a 'pull the plug' seizure the virtual machine was paused while the browser was still open and an image taken of the resulting system. Each stage in each experiment was imaged using *FTK Imager* into an E01 image file for future study. Recovery of database metadata, current database contents, and carving were carried out using *ESECarve V1.19*, selection and counting of results were carried out using scripted regular expressions in Python, and file system mapping was carried out in *XWays Forensics V17.3*.

Memory images were not captured, since they are unfortunately rarely available to forensic analysts; however, the findings here on pagefile provide a good indication of the potential benefit of a memory image if one were available.

The database schema (metadata) was first extracted and a series of scoping experiments carried out to confirm how ESE data fields are used by Internet Explorer. This allowed confirmation of important factors such as the interpretation of date and time information; the results are summarized in section 4 below.

Each browsing experiment was conducted over 5 days in which the remanence of artifacts resulting from InPrivate browsing on the first day were measured under different conditions of system use. The measurement process was to carve all available database records from the disk image for the day; records of interest were identified and the offsets provided by the carver then mapped to the file system producing a complete map of artifacts of interest. The InPrivate browsing activity was designed to produce an image-rich data set with sufficient images to allow indicative statistics to be obtained. *www.carandclassic.co.uk* provides lists of classic cars for sale with small pictures, and sufficient pages were visited to guarantee that each browsing session had cached over 500 images.

Only recoverable database records were counted as results, in contrast with the alternative of searching the image for distinctive URLs. As would be expected, the URL search would identify a few extra artifacts where the URL was available but the surrounding record was corrupt. In these tests the difference is small (around 7%). In real cases it has proved necessary to recover the contextual information provided by the whole record, not just a URL fragment. This methodology is therefore closer to the needs of forensic practice than string search counting.

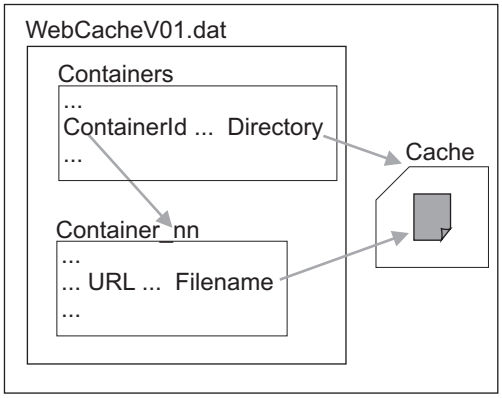
Three InPrivate experiments were carried out: a scoping exercise, a controlled comparison with ample system memory, and finally a mixed-load scenario.

The scoping exercise was used to confirm the likelihood that artifacts would be found. Because it appeared that InPrivate records were added to the database normally, and then deleted, it seemed likely that there was a marker value within each record which either indicated deletion or InPrivate browsing. This was identified and then confirmed by reference to Microsoft documentation. The first experiment also suggested that the availability of records in the database and log files would persist only to the next use of the browser, although records could still be found on other parts of the disk.

The second experiment was designed to confirm that InPrivate browsing records could be uniquely identified, and the point at which data were deleted from database files. Three separate machines were used: *Use, Avoid,* and *Control*. InPrivate browsing was carried out on the first two, after which the Use machine was subject to daily light browsing, and the Avoid machine was subject to daily use without opening the browser. The Control machine was subject to the same use, but the initial session was carried out using normal, as opposed to InPrivate, browsing. These experiments were carried out with ample memory (1 GByte) allocated to each virtual machine; since the browsed images were small in size and there was negligible background machine load it was expected that the browser would be unlikely to page memory.

The results of this experiment confirmed expectations: comparison of the Avoid and Use machines resulted in InPrivate artifacts being deleted on the first use of the browser in the Use machine, but not during the experiment in the Avoid machine. Comparison of the artifacts recovered from the Use machine and the Control machine was unable to find any instance which contradicted the field value which marked InPrivate browsing. Also as anticipated, no artifacts were discovered on disk outside the browser files.

Figure 2: *Overview of Cache Data Structures.*The Containers table acts as an index into a series of Container_nn tables and specifies the directory path to the related cache directory; individual records in the Container_nn table specify the cached filename.



The third experiment was designed to simulate a computer with a heavier workload; a virtual machine was created with 500GByte of memory, and while the test InPrivate browsing session was conducted a video was watched using the *Firefox* browser. The disk map of this experiment is reported in detail below; the results were consistent with those previously achieved and resulted in a large number of long-lived recoverable browsing records outside the database and log files.

4. Browser Data structures

4.1. Overview

Internet Explorer 10 maintains a single database which includes history records and indexes to various caches, the database file *WebCacheV01.dat* is located at:

\Users\%USERPROFILE%\AppData\Local\Microsoft \Windows\WebCache

This location appears to be standard, we have not identified a registry stetting or group policy that allows it to be changed. The same directory includes associated database files, *V01.chk*, *V01.log*, and *V01nnnn.log*, where nnnn is a hexadecimal sequence number.

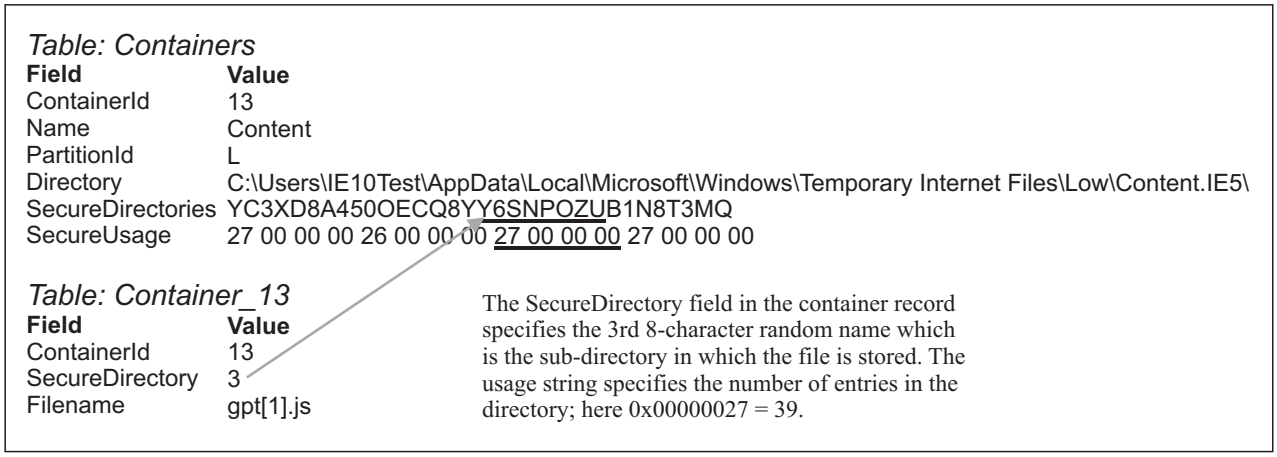
The relationship between the most important database tables is shown in figure 2.

The WebCacheV01.dat database contains a *Containers* table, and records in that table act as an index to actual data containers which are tables with names of the form *Container_nn*. The presence of a record in the Containers table does not guarantee that there is a corresponding Container_nn table, although they are usually present.

Index records in the Containers table specify the directory which contains the actual data items (e.g. cookies, cache), the names of these files are specified in the browsing records in the associated Container_nn table.

The Containers table therefore acts as a master index, its fields include:

Figure 3: *Secure Directories.*Files are placed in subdirectories of the path identified in the Containers table.



- *ContainerId*: which is the number of the associated Container_nn table.
- *Directory*: in which cached or stored files are held.
- *Name*: which describes the type of data in the container (e.g. cookie, content, history).
- *PartitionId*: which specifies the integrity partition: L - Low, H - High, or a specific AppContainer).

Windows 8 systems may have large number of containers; each Metro Application will usually have containers for both cookies and cache content and the desktop Internet Explorer will have similar containers at both M and L integrity levels. Containers are also allocated to History records and other data types. A typical Windows 8 laptop with only the standard Metro Applications and little history is likely to have between 40 and 50 containers. Knowledge of these data structures is potentially valuable to a forensic practitioner, since they each specify a directory with Internet records on disk.

Each record in a Container_nn table includes the following fields:

- *ContainerId*: which references the associated row in the Container table.
- *SecureDirectory*: which is used to index a sub-directory within the cache path.
- *Type*: which may be used to determine if the record originated with InPrivate Browsing.
- *AccessCount*: the number of times a URL has been referenced (but not necessarily selected by the user).
- *Date and Time Information*: fields include Sync, Creation, Expiry, Modified, and Accessed times.
- *URL*: the URL from which the information was obtained, in some instances a response header is also available.
- *Filename*: the name of the cache file used to store the data item.

These fields are included because they are likely to be of forensic significance; they are discussed in detail in the sections below. Other fields have not been diagnosed or confirmed by experiment.

4.2. Secure Directories

Content cache items and certain other types (e.g. DOMStore data objects) are stored in sub-directories of the path specified in the Containers table. These sub-directories are given random 8 character names, similar to cache storage in previous versions of Internet Explorer. The mapping from the database record to the sub-directory name is implemented as shown in figure 3. The SecureDirectories string is a list of 8 character names, indexed starting from 1; the SecureDirectories value in the record specifies which of these sub-directories is used for that particular data item.

The usage record is stored as a long binary string, here shown as a series of hexadecimal values; the values should be read as a series of 4-byte little-endian numbers which indicate the number of files in the corresponding directory. Experimentally it was not always possible to confirm that these numbers were exactly correct, they often differed by a small number from the actual number of files, this is unexplained but may be a result of cached writing.

One feature of the directories listed in the Containers table (not the random named subdirectories) is that they usually contain a *container.dat* file, of zero size. The consequence is where there are records without associated files, such as History records, the directory listed in the Containers table contains only the zero size *container.dat file*. This arrangement seems to have little forensic value, other than the possibility that zero size files of this type may indicate the presence of actual records in a corresponding database table.

Table 1: *Examples of PartitionIds and related directories for Internet Explorer content entries in the Containers table.* The text distinguishes between different integrity partitions: Low, Medium, and those specific to an Application.

ContainerId	*PartitionId*	*Directory*
8	M	C:\Users\IE10Test\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\
7	S-1-15-2-1430448594-2639229838-973813799-439329657-1197984847-4069167804-1277922394	C:\Users\IE10Test\AppData\Local\Packages\windows_ie_ac_001\AC\INetCache\
13	L	C:\Users\IE10Test\AppData\Local\Microsoft\Windows\Temporary Internet Files\Low\Content.IE5\

4.3. PartitionId and AppContainer

Integrity levels and AppContainers were introduced in section 2.3. For desktop applications the PartitionId in the Containers table is L (low integrity) corresponding to data written in protected mode (Internet, or Restricted zones), or H (high integrity) corresponding to data from the Local Machine, Local Intranet, or Trusted domains. Note that some data, such as daily history records, are written by the local machine outside protected mode, despite originating from an Internet access.

For AppContainers the SID (Security Identifier) of the Application is the PartitionId, and the associated directory is specific to that application. The name for Internet Explorer when run as an Application is *windows_ie_ac_001*, and this can be related to the SID via the associated directory path, or by searching for the SID in the Registry (under Classes), and retrieving the associated DisplayName.

Table 1 provides an example of three Content records related to Internet Explorer.

4.4. Date and Time Information

Each browsing record has a number of date and time stamps, although not all will necessarily be set. The meaning of these date-time groups is intuitive and follows previous practice. The summary below has been confirmed by browsing with offset machine times while monitoring network packets as well as the resulting cache files.

All dates and times are recorded in UT (GMT); those derived from the host computer use the same offsets as the file system. The dates and times present in the Content tables have the following significance:

- *SyncTime*: is the most recent time the url content was synchronized, either by downloading a data item to the cache, or by comparing the update time of the on-line content to that of the cached item. If the cached data item is updated as a result of a synch, then the modify time in the file system is the same as the sync time.
- *CreationTime*: is the time that the cached item is first created; it is the same as the file system create time for the corresponding file.
- *ExpiryTime*: is the time set in the protocol data unit by the web server; it is intended to be the time after which the cache must be refreshed. This date-time may be calculated from a maximum age, or directly reflect an expiry time; either or both may appear in the HTTP protocol header.
- *ModifiedTime*: is the modified time set in the protocol data unit by the web server. It is intended to reflect when the on-line data item was last updated. It is not related to file update times in the cache.
- *AccessTime*: is the most recent time the user accessed the cached item.

Records also include *PostCheckTime* and *SyncCount* fields. No values for PostCheckTime were observed. Despite experiments designed to ensure cache synch, which was confirmed in the network trace and in SyncTime values, the sync count did not reflect the activity. It's purpose is undiagnosed. Note that times are changed when history records are rewritten, as described in the next section.

4.5. Data Name

The name field in the Containers table gives the purpose of the associated container and its directory. Familiar names from previous versions of Internet Explorer are *Cookies*, *Content*, and *History*. Downloaded data are given their own container, *iedownload*, as is web storage which is named *DOMStore*. This list is certainly not exhaustive, three other types of named data justify specific mention: MSHIST01yyyymmddyyyymmdd, iecompat, and PrivacIE.

- The name used for history records is also familiar from previous versions of Internet Explorer: *MSHIST01* followed by two dates in year-month-day format. The name signifies the date range over which records have been collected. For example MSHIST01201310022013100 3 contains records between 2nd and 3rd of October, 2013. History records provide a summary of user activity, more detail is found in the Content records that index the Internet cache.
  As with previous versions of Internet Explorer, daily history records are subsumed into weekly records: a new Container_nn table is created, records copied to that container and the old history tables deleted, usually when the browser is next opened. Unfortunately the new records have modified URLs and timestamps. The URL field is

prefixed with the to-from dates of the history container, in the same format as the table name; when the record is rewritten this is replaced by the new date range. The SyncTime and AccessedTime are changed to the time the browsing records were rewritten.

- *iecompat* records are a pre-configured list of Internet addresses where it may be necessary to use a compatibility mode, that is for Internet Explorer to emulate an earlier browser version. They are loaded when the database is rebuilt, so may appear in memory or logs after a delete operation. Apart from perhaps providing evidence that the database has been rebuilt, which in any case is likely to be evident, they appear to be of little forensic interest. These records were filtered from these experiments to avoid overstating the number of recovered records.

- *PrivacIE* records are not related to InPrivate browsing. They are used to record URLs to third party sites, for example advertisers' web pages that are referenced on web pages visited by the user. This information is used in InPrivate filtering to limit indirect access to such sites (Wilson, 2012). Unfortunately it is not just the name that may cause confusion between InPrivate Filtering and InPrivate browsing; the type indicator used to identify InPrivate browsing is also used for PrivacIE records, and the forensic practitioner must be aware of the distinction to avoid ascribing PrivacIE records to InPrivate browsing. See section 5.1, below.

5. InPrivate Browsing

This section describes two important outcomes of the experiments described in section 3: how to identify records, and where and when such records may be found.

5.1. Identifying InPrivate Browsing Records

Because InPrivate browsing records were found to be stored in the same tables as other content, and then later deleted, it seemed likely that there was a marker in each record to identify if its origin was InPrivate browsing. Scoping experiments identified the *Type* field as a likely candidate, and analysis revealed that this field was a bitmask with some obvious features: cookie records always had the 0x100000 bit set, and History records were similarly associated with 0x200000. The bit correlated with InPrivate browsing is 0x20000. This provided sufficient information to search Microsoft Developer information; the specification of these bitmasks is divided between the header files *WinInet.h* and *Wininet.h* in the Windows 8 SDK. Field definitions that are likely to be of interest to forensic practitioners are listed in table 2.

Further experiments were carried out to identify other circumstances where the PRIVACY_MODE_CACHE_ENTRY bit was set, and none were found for cache Content records. This bit was set, however, in PrivacIE records, which are introduced above. In this case the STICKY_CACHE_ENTRY bit was also set, usually giving a type value of 0x20004.

Table 2: *Decoding the Type Field.* This table lists the bit assignments most likely to be of forensic interest; a full list can be found in the Windows 8 SKD headers *WinInet.h* and *Wininet.h*.

Description	*Value*
NORMAL_CACHE_ENTRY	0x00000001
STICKY_CACHE_ENTRY	0x00000004
HTTP_1_1_CACHE_ENTRY	0x00000040
STATIC_CACHE_ENTRY	0x00000080
DOWNLOAD_CACHE_ENTRY	0x00000400
REDIRECT_CACHE_ENTRY	0x00000800
PRIVACY_MODE_CACHE_ENTRY	0x00020000
COOKIE_CACHE_ENTRY	0x00100000
URLHISTORY_CACHE_ENTRY	0x00200000
PENDING_DELETE_CACHE_ENTRY	0x00400000
POST_RESPONSE_CACHE_ENTRY	0x04000000
INSTALLED_CACHE_ENTRY	0x10000000
IDENTITY_CACHE_ENTRY	0x80000000

In conclusion, the agreement of Microsoft SDK documentation with experimental evidence provides confidence that Content records with the PRIVACY_MODE_CACHE_ENTRY bit set originated from InPrivate browsing. However, PrivacIE records do not provide evidence of InPrivate browsing; where the container cannot be reliably identified, records with the STICKY_CACHE_ENTRY bit set should be regarded as PrivacIE, and not InPrivate browsing.

5.2. The Lifecycle of InPrivate Browsing Artifacts

This section presents results of the third experiment described in section 3. This was deliberately carried out in a mixed work environment rather than a 'clean' environment in which the only activity was that under study. Memory was constrained and competing processes were running at the same time as InPrivate browsing.

A full disk map for the results of this experiment is presented in table 3. The map was created by carving the whole disk for Internet Explorer 10 records using ESECarve, filtering those records to extract only InPrivate browsing using the Type field described above, then mapping the resulting records to files. Each stage of the experiment was separately mapped, and the table shows the how the number of records recovered and their file system assignments changed as the experiment progressed.

The term *unallocated* is used for any space within the file system which is not allocated to a file system object or file, this is synonymous with *free space* used by some forensic tools.

The table is divided into two parts, the first set of records map to database files, the second to other areas of the disk; they will be described separately. The counts given in each section are the number of InPrivate browsing Content records recovered.

5.2.1. Database Files

Database log records are first written to the current log file (V01.log), and when this is full that file is renamed to the next in the hexadecimal sequence (in this example V010000F.log) and a new V01.log allocated. This process is evident in the disk map.

Table 3: *Disk Map of Recovered InPrivate Browsing Records.* InPrivate browsing *Content* records were carved from a disk image and mapped to file allocations. The table shows how the availability and mapping of recovered records change with time.

1: Pull Plug		*2: After Shutdown*		*3: After Use 24 Hours Later*		*4: After Use 48 Hours Later*	
Count	*Allocated To*	*Count*	*Allocated To*	*Count*	*Allocated To*	*Count*	*Allocated To*
817	WebCacheV01.dat	0	Records overwritten with 0x4F				
580	V010000D.log	0	Disk space reallocated, overwritten with 0x4F or new records				
624	V010000E.log	624	unallocated	0	Contents overwritten with 0xFF		
254	V01.log	260	V010000F.log	0	Reallocated to a new file and overwritten.		
0	unallocated	23	V01.log	16	V0100010.log	0	unallocated, overwritten 0x4F
984	pagefile.sys	984		984		984	
442	System Volume Information	442		442		442	
Total Carved Records: 3701		2333		1442		1426	
Total Unique Records: 899		812		742		741	

The database file, WebCacheV01.dat, is populated with InPrivate browsing records during the browsing process; these are securely deleted (overwritten) when the browser is closed. No records remained in this file after the browsing session.

Records do remain in the log files after the InPrivate session. In general (see V010000E.log, V01.log/V010000F.log, V01.log/V0100010.log) logs are removed when Internet Explorer is opened, not when it is closed. This is consistent with the behavior that would be predicted for this database (see section 2.1): while it is operation in-memory records and logs are updated and the database file is marked as 'dirty'; before the database can be re-opened for use it is updated from the logs and logs that are no longer needed are deleted. In control experiments in which the test machine was used but the browser not opened the logs were not modified.

The evidence for how these logs are eventually deleted is mixed, but it is clear that no InPrivate browsing records remain. The common behavior observed for several files is for individual records to be overwritten by 0x4F, this is consistent with the treatment of InPrivate records in the database file.

There is one anomaly in these results which is the treatment of V010000D.log. This log file was deleted while the InPrivate session was active, and it is eventually returned to unallocated space and completely overwritten by 0xFF. The anomaly is that it is released to unallocated space in column 2 (at the first shutdown) but not overwritten until the system is next used. Removing file allocation then subsequently overwriting it in free space is a curious system behaviour, it was checked using a different forensic tool with the same result, to ensure that it was not a feature of the mapping in XWays Forensics.

5.2.2. Other Disk Artifacts

InPrivate browsing records were found in large quantities in two areas of disk unrelated to the database files. The first is pagefile, which was expected given the process environment in which the browsing was carried out.

The second area was within a large file in the System Volume Information directory. Comparison of the records found in this directory with the database files captured in previous experimental steps suggest that this content originated from the V010000D.log file.

5.2.3. Recovery Success

Because the same record may be found in several places there will inevitably be duplicates. The total of unique records shown in table 3 is the number of unique combinations of database ID and URL recovered from all sources; this measure will not count revisited URLs, however revisits were not a major feature of this experiment. Because of the comprehensive method of capture, the 899 records captured when the machine was stopped after browsing is likely to be a high proportion of the total records generated while browsing.

There are three distinct sources of browsing records: the database file, log files, and other areas on disk. In this experiment most of the unique results could have been obtained from any of the three sources; at the end of the experiment when no records were available from database files 82% of the available records were still recoverable. The disk map also provides a clear indication of when various InPrivate browsing records are available:

- From the database file: during browsing, or if the machine has been powered off during browsing.
- From database log files: after the browsing session, and before the browser is next opened.
- From other areas on disk: may persist for some time, depending on system activity.

6. Discussion

The primary objectives of this research were to clarify under what circumstances, and to what extent, it is possible to recover InPrivate browsing records from Internet Explorer 10, and to identify a marker that provides evidence that a recovered record originated from InPrivate browsing.

Experimental results confirmed by Windows header files provide a clear marker for InPrivate browsing records: a *Type* field

with bit 0x20000 set. This result is confirmed experimentally for *Content* records, i.e. those that index the cache. *PrivacIE* records, which are not necessarily related to InPrivate browsing also set this bit. Usually confirmation of the data held in a record requires the *ContainerId* to be checked in the *Containers* table; however this may not be possible if that table is not recovered. There is also the usual problem of foreign key references; using a key (i.e. ContainerId) to look up a value in a second table requires evidence that they existed at the same time, similar to the problem of determining what web content a user actually viewed from an old URL. In this case, however, evidence within a record will identify it as PrivacIE data: this name is present in the URL field, and the *Type* is likely to have the sticky cache bit set (0x4). There should therefore be no possibility of confusing PrivacIE records with Content records resulting from InPrivate browsing.

InPrivate browsing records were found in the database file, *WebCacheV01.dat*, in related log files, and in other areas on the disk:

WebCacheV01.dat held recoverable InPrivate records while the browsing session was in force, or if the machine was depowered while the browsing session was still open; when the browsing session closed these records were securely deleted.

Log files provided recoverable records between the closure of the InPrivate browsing session and the next time that Internet Explorer was opened. This was observed in all experiments, and is consistent with the expected database behaviour: while the database is in use in-memory records and logs are updated and the database file is marked as 'dirty'; before the database can be re-opened it is updated from the logs and logs that are no longer needed are deleted. The fate of records that were in deleted logs is not fully resolved; the disk mapping process did not identify a single record remaining after log files were deleted, which suggests secure deletion.

Other disk areas provided recoverable records; the experiment reported above found records in the *pagefile* and in *System Volume Information*. As would be expected, it proved possible to influence the extent that records were saved to pagefile by varying the amount of physical memory and the background workload of the machine. These factors can be expected to influence the recovery of records from this area, and also the length of time that they remain in the pagefile before being overwritten. The records found in System Volume Information derive from a logfile which was deleted while the InPrivate session was active; it has not proved possible to recover this file information using the VSS service (i.e. as a shadow copy), so the function of these records is unknown.

7. Conclusion

This paper reports research into the extent that a user's InPrivate browsing history in Internet Explorer 10 can be recovered, and if such records can be reliably identified as resulting from InPrivate browsing.

None of the records recovered during these experiments could be read using the standard database application interface, the experimental work relied on a carving tool (ESECarve) to find records within a disk image, after which they were manually mapped to the file system presented by a standard forensic tool. In about 40% of the experiments depowering the computer during the browsing session resulted in an unrecoverable database, due to missing log files.

The results indicate that InPrivate browsing records can be reliably identified. Browsing records may be recovered in large quantities depending on seizure timing: if the machine is depowered during an InPrivate browsing sessions records may remain in the database file (WebCacheV01.dat) if Internet Explorer has not been used since such a session then there are likely to be records in database log files, otherwise it is necessary to carve records from the disk. Experimentally, all three areas held a high proportion of the user's browsing session, and even after log file records became inaccessible over 80% of the browsing record remained in other disk areas.

This work has extended and clarified informal results of other researchers, some of whom found considerable historical records and some of whom found little. It demonstrates that longitudinal studies over a series of machine and application cycles provide more information about the remanence of forensic artifacts than one-off usage experiments, as does the explicit consideration of background machine load during forensic experiments.

References

Aggarwal, G., Bursztein, E., Jackson, C., Boneh, D., 2010. An analysis of private browsing modes in modern browsers. In: USENIX Security Symposium. pp. 79–94.

Barth, A., 2011. Rfc 6265: Http state management mechanism. http://tools.ietf.org/html/rfc6265 (accessed October 2013).

Chivers, H., Hargreaves, C., 2011. Forensic data recovery from the windows search database. digital investigation 7 (3), 114–126.

Jones, K. J., 2003. Forensic analysis of internet explorer activity files.

Mahendrakar, A., Irving, J., Patel, S., 2010. Forensic analysis of private browsing mode in popular browsers.

Malmström, B., Teveldal, P., 2013. Forensic analysis of the ese database in internet explorer 10.

Metz, J., 2010. Extensible storage engine (ese) database file (edb) format specification. http://forensic-proof.com/wp-content/uploads/2011/07/Extensible-Storage-Engine-ESE-Database-File-EDB-format.pdf (accessed October 2013).

Microsoft, 2012. Internet explorer 10 privacy statement. http://windows.microsoft.com/en-gb/internet-explorer/ie10-win8-privacy-statement (accessed October 2013).

Microsoft, 2013. Extensible storage engine. http://msdn.microsoft.com/en-us/library/5c485eff-4329-4dc1-aa45-fb66e6554792.aspx (accessed October 2013).

Ohana, D. J., Shashidhar, N., 2013. Do private and portable web browsers leave incriminating evidence? a forensic analysis of residual artifacts from private and portable web browsing sessions. 2012 IEEE Symposium on Security and Privacy Workshops 0, 135–142.

Said, H., Al Mutawa, N., Al Awadhi, I., Guimaraes, M., 2011. Forensic analysis of private browsing artifacts. In: Innovations in Information Technology (IIT), 2011 International Conference on. pp. 197–202.

Satvat, K., Forshaw, M., Hao, F., Toreini, E., 2013. On the privacy of private browsing–a forensic approach.

W3C, 2013a. Indexed database api. http://www.w3.org/TR/IndexedDB/ (accessed October 2013).

W3C, 2013b. Web storage. http://www.w3.org/TR/webstorage/ (accessed October 2013).

Wilson, C., 2012. Privacie entries. http://kb.digital-detective.co.uk/display/NetAnalysis1/PrivacIE+Entries (accessed October 2013).

Appendix A. Database File Recovery using *esentutl*

This section describes database recovery via the *esentutl* utility. Although no InPrivate browsing records could be recovered via the database application interface the process of database recovery may be needed for some forensic tools, and the knowledge of how to use esentutl allows the investigation of new types of ESE database.

In most practical cases the database file (*WebCacheV01.dat*) 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 if it is to be interrogated via the database API.

Appendix A.1. Required files

The Internet Explorer 10 database is normally located at:

\Users\%USERPROFILE%\AppData\Local\Microsoft \Windows\WebCache

The files that must be retrieved from the image are:

- The database file (*WebCacheV01.dat*).
- Any log files (*V01.log* and *V01nnnnn.log* - where nnnnn is a hexadecimal sequence number).
- The checkpoint file (*V01.chk*).

V01.log is 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 hexadecimal log numbers. This is the number it will be assigned when full, at which time a new *V01.log* will be started.

Appendix A.2. Recovering the database file

This requires the Microsoft *esentutl* utility, which is a standard component of Windows, and is run from the command line. The correct version of esentutl must be used; in other words the recovery is best carried out using the version of Windows from which the database was obtained. 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 *WebCacheV01.dat* 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: *V01000C1.log*, *V01000C2.log*, together with *V01.log* in this example. (Note the comment above: *V01.log* is the most recent log, in this case *V01000C3.log*.)

The *esentutl* recovery process is then used to bring the database to a consistent state. Assuming that *esentutl* is run from a directory containing *WebCacheV01.dat*, the necessary log files, and the checksum, then the command line is:

esentutl -r V01 -d

Assuming that *esentutl* reports success, the *WebCacheV01.dat* file may now be accessed via the database API. If the database is unknown it is often useful to begin by obtaining a list of table names, using:

esentutl -mm <path to database file>

Appendix B. Record Carving using *ESECarve*

ESECarve is a recovery tool which is capable of reading clean databases via their application interface, or reliably carving database records. A description of the carving process and its reliability is given in (Chivers and Hargreaves, 2011); the tool is available from the author for forensic investigation, education or research.

In order to run the carving tool it is necessary to provide a target schema; this is built by the tool automatically from a sample clean database: a file named *Reference.edb* is placed in the working directory. This file should be from the same version of windows as the investigation target.

The tool is invoked on the command line with:

ESECarve –IE10 -r <working-directory> <target-file>

The result is a file *CarvedData.csv* which can be opened in a spreadsheet. The program also places a log file *ESECarveLog.txt* which records the processing together with MD5 hashes for input and output files. The fields in the output file are identical to those that would be recovered via the database application interface, with the addition of an offset field which records the address from which the record was carved