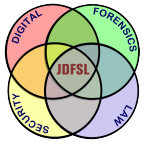

THE JOURNAL OF
DIGITAL FORENSICS,
SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 7 | Number 3

Article 3

2012

Automatic Crash Recovery: Internet Explorer's black box

John Moran
County of Cumberland

Douglas Orr
Special Investigations Unit, Spokane Police Department

Recommended Citation

Moran, John and Orr, Douglas (2012) "Automatic Crash Recovery: Internet Explorer's black box," *Journal of Digital Forensics, Security and Law*: Vol. 7 : No. 3 , Article 3.
DOI: https://doi.org/10.15394/jdfsl.2012.1127
Available at: https://commons.erau.edu/jdfsl/vol7/iss3/3

Automatic Crash Recovery: Internet Explorer's black box

John Moran
County of Cumberland
Portland, Maine
john@jtmoran.com

Dr. Douglas Orr
Special Investigations Unit
Spokane Police Department
Spokane, Washington
dorr@spokanepolice.org

Abstract

A good portion of today's investigations include, at least in part, an examination of the user's web history. Although it has lost ground over the past several years, Microsoft's Internet Explorer still accounts for a large portion of the web browser market share. Most users are now aware that Internet Explorer will save browsing history, user names, passwords and form history. Consequently some users seek to eliminate these artifacts, leaving behind less evidence for examiners to discover during investigations. However, most users, and probably a good portion of examiners are unaware Automatic Crash Recovery can leave a gold mine of recent browsing history in spite of the users attempts to delete historical artifacts. As investigators, we must continually be looking for new sources of evidence; Automatic Crash Recovery is it.

Keywords: Automatic Crash Recovery, ACR, Internet Explorer, IE8, IE9, Browsing history, RecoverRS, Compound files.

1. INTRODUCTION TO AUTOMATIC CRASH RECOVERY

In order to understand the potential value of Automatic Crash Recovery to investigators, some background in to what exactly Automatic Crash Recovery does is required. According to Microsoft, "Automatic Crash Recovery (ACR) is a feature of Windows® Internet Explorer® 8 that can help to prevent the loss of work and productivity in the unlikely event of the browser crashing or hanging" (Microsoft, 2008, p. 3). From the user's perspective, ACR is what provides the option to 'Restore Session' when Internet Explorer closes improperly. Providing this functionality requires Internet Explorer to store numerous pieces of information about the history of the browsing session.

ACR can be disabled by going to 'Tools' -> 'Internet Options' -> 'Advanced' and unchecking "Enable automatic crash recovery" in the 'Browsing' section.

Interestingly, research shows that even with ACR disabled, Internet Explorer will continue to store information for its use. Similarly, research shows that even with InPrivate Browsing enabled, ACR artifacts will still be created.

Several common "cleaning" utilities were tested and not a single utility removed the files created by ACR. It appears that there is currently no way to prevent Internet Explorer from creating ACR artifacts and furthermore that the only reliable way for a user to remove ACR artifacts is to manually delete and overwrite them after each session.

2. ARTIFACTS CREATED BY ACR

The files of interest created by ACR are initially written to the C:\Users\<user>\AppData\Local\Microsoft\Internet Explorer\Recovery\Active directory in Windows 7 or the C:\Documents and Settings\<user>\Local Settings\Application Data\Microsoft\Internet Explorer\Recovery\Active directory in Windows XP.

Internet Explorer creates two types of files in this directory. The first type uses the naming convention 'RecoveryStore.{<GUID>}.dat' and is created when Internet Explorer is first executed. Referred to from this point on as the "recovery store file," only one such file is created regardless of the number of tabs or windows opened by the user (except when using InPrivate Browsing a second recovery store file is created for the InPrivate Browsing session). The second type created uses the naming convention '{<GUID>}.dat'. One of these files is created when Internet Explorer is first executed and one additional file is created for each additional tab or window that is opened. These files will be referred to from this point on as the "tab data files." The globally unique identifiers (GUIDs) created for both the recovery store files and the tab data files are in hexadecimal and display as ########-####-####-############. The format of these GUIDs as well as the information they contain is explained in greater detail in the following section.

When Internet Explorer is closed by the user, the recovery store file and the tab data files are removed from their existing locations and recreated in the C:\Users\<user>\AppData\Local\Microsoft\Internet Explorer\Recovery\Last Active directory in Windows 7 or the C:\Documents and Settings\<user>\Local Settings\Application Data\Microsoft\Internet Explorer\Recovery\Last Active directory in Windows XP with new GUIDs. Any GUIDs stored within the recovery store files and tab data files are also updated unless otherwise noted.

One registry value that may be of interest during an investigation is HKCU\Software\Microsoft\Internet Explorer\Recovery\AutoRecover. A DWORD value of 0x00000000 indicates that ACR is enabled; a DWORD value of 0x00000002 indicates that ACR is disabled. As mentioned previously, ACR files will be created even when this value is set to 0x00000002, however this value may be an indication the user was attempting to hide their browsing

activities.

Another registry key that may be of interest is HKCU\Software\Microsoft\Internet Explorer\Recovery\Active. When Internet Explorer is executed and the recovery store file is created, a new DWORD value is created in this key using the GUID of the recovery store file as the name and 0x00000000 as the value. For example, if the recovery store file created was RecoveryStore.{3519D794-44E1-11E0-8CA1-005056C00008}.dat, a new DWORD value named {3519D794-44E1-11E0-8CA1-005056C00008} would be added to the key. When Internet Explorer is closed properly, this value is deleted from the key. This key appears to be what Internet Explorer checks to see if there are previous browsing sessions that can be recovered; manually adding previous ACR files to the C:\Users\<user>\AppData\Local\Microsoft\Internet Explorer\Recovery\Active directory and adding the GUID of the recovery store file to this registry key caused Internet Explorer to offer to restore the browsing session from the previous ACR files.

Two other registry keys seen in Windows 7 environments are HKCU\Software\Microsoft\Internet Explorer\Recovery\AdminActive, which contains the GUID of the recovery store file currently open in Internet Explorer when run as Administrator and HKCU\Software\Microsoft\Internet Explorer\Recovery\PendingDelete, which contains the GUIDs of the tab data files currently being used by Internet Explorer.

3. ANALYSIS OF ACR FILES

By the very nature of their function, ACR files must store several key pieces of information that can be of use to investigators, such as dates, times, and browsing history that might be otherwise unavailable through other means. In order to get the most from ACR artifacts and more importantly be able to articulate the method by which these artifacts are created and the process of recovering these artifacts, the next several sections will detail the file format and where key evidence may lie.

3.1 GUID Format

The GUID itself can provide some important information and is important to mention, most notably the date and time the file was created. The first eight bytes of the GUID contain the date/time the file was created, or in other words, the date/time Internet Explorer was opened or closed (in the case of a recovery store file) or the date/time an individual tab was opened or closed (in the case of a tab data file). The date/time is stored as the number of 100 nanoseconds since October 15, 1582 in little endian; a very similar format to the filetime format, which begins January 1, 1601.

In order to calculate the date/time from the GUID, we must extract the first eight bytes from the GUID, then change the byte order from little endian to big endian. The first 4 bits of the big endian value represents the version number and are not

part of the date/time and we should ignore them. We should then subtract 0x146BF33E42C000 (5,748,192,000,000,000) to account for the difference in epochs and convert the resulting value to filetime (Parsonage, 2010). A sample calculation can be seen in Figure 1.

Tab data file name:	6E165296-3930-11E0-8FE9-000C29EF1366
Extract first 8 bytes:	6E165296-3930-11E0
Convert to big endian:	0x11E039306E165296
Drop the 1st 4 bits:	0x01E039306E165296
Subtract 146BF33E42C000:	0x1CBCD3D2FD39296 (129422676989285014)
Convert to FileTime:	Tuesday, February 15, 2011 1:21:39 PM UTC

Figure 1 (Sample Date/Time Calculation from Tab Data File)

The last six bytes of the GUID contain a node ID that may also be of interest depending on the nature of the investigation. In most cases, the node ID will be one of the available IEEE 802 medium access control (MAC) addresses on the system. Yet in other instances, a random ID may be used (Leach, Mealling, & Salz, 2005). The remaining two bytes of the GUID not previously mentioned make up the variant and sequence numbers that are of no value in the examination to these particular files.

3.2 ACR File Format

Both the recovery store file and the tab data files are stored in a format called the compound file binary file format file, which will henceforth be referred to simply as a compound file. These files may also be referred to as object linking and embedding (OLE) compound files. Although some level knowledge regarding the compound file format is necessary when discussing carving these files from unallocated space, a complete explanation of the compound file format is beyond the scope of this paper. In fact, a complete explanation of the compound file format has already been issued by Microsoft (2012a), titled *[MS-CFB]: Compound File Binary File Format*. Fortunately, a basic understanding of the compound file format will suffice for examination of these files.

Like many other files, compound files have a common header that can be used to locate and identify these files in unallocated space (discussed below). However, that is where the similarities with other common file formats end. A compound file functions very much like a File Allocation Table (FAT) file system on a disk; it contains a FAT, which tracks all sectors in the file, as well as directory entries and folder- and file-like structures. The folders in a compound file are referred to as *storages* and the files are referred to as *streams*. Like any other file system, a storage can contain other storages or streams. A stream however cannot contain a storage.

There is one other important structure within the compound file - the *property set*. Unlike a stream that can contain Unicode text of any length, a property set

follows a strict format. Once again, a thorough explanation of property sets is well beyond the scope of this paper. However, Microsoft has come to the rescue again with a complete explanation of property sets titled *[MS-OLEPS]: Object Linking and Embedding (OLE) Property Set Data Structures* (Microsoft, 2012b). For the purposes of examining ACR files, it is important to know that property sets contain one or more properties with a unique numeric identifier, a value type (such as date [VT_DATE], four-byte unsigned integer [VT_UI4] or Unicode string [VT_LPWSTR]) and a value.

Making sense of compound files in their raw hexadecimal form can be a daunting task, even with an expert knowledge of the compound file format. While it is possible to identify some text from the file, it is very difficult to attribute context without a great deal of time and effort. Thankfully, there are numerous tools capable of reading the compound file. Several forensics suites, such as the Forensic Toolkit (FTK) and EnCase, are capable of reading the compound file. There are also several free utilities available that will read compound files with varying success. Another product available for reading these files is the Compound File Explorer (CFX) by CoCo Systems Ltd. (http://www.coco.co.uk/developers/CFX.html). CFX is not free but well worth the price of 20 GBP. Unlike most programs, CFX is capable of reading not only the text streams in compound files but also the property sets in an easy to read format. Tools such as CFX are key to the examination of ACR artifacts as they present the information inside the compound file in a much easier to understand way and add a measure of context.

3.3 The Recovery Store File

The recovery store file contains basic information about the browsing session. Only one recovery store file is created by Internet Explorer regardless of the number of tabs or windows the user opens. The one exception to this is when InPrivate browsing is used; when a user selects InPrivate browsing, a new window with the InPrivate Browsing logo opens, and a second recovery store file is created in the \Active directory. If both the original window and the InPrivate browsing window remain open, both recovery store files will remain in the \Active folder. Opening additional InPrivate browsing windows will not create additional recovery store files.

At a minimum, each recovery store file contains three streams: the 'TS#' stream (where # is an integer starting at 0, discussed in greater detail in the next section), the 'FrameList' stream and the '{0B00252A-8D48-4D0B-7B79887F2B96}' stream. A fourth stream, the 'ClosedTabList' stream, may also be present in some recovery store files (Figure 2). The purpose of these streams and the data commonly stored within are described below.

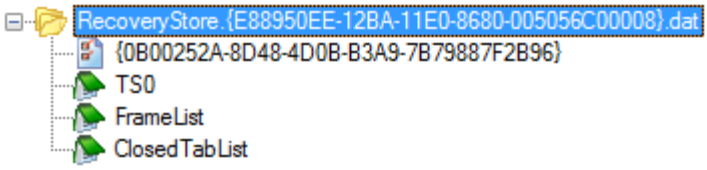

Figure 2 (Recovery Store File Streams as Viewed in CFX)

3.3. 1 The 'TS#' Stream

A 'TS#' stream is created for each tab or window opened by the user. The numbering for the 'TS#' stream starts at 0 and, in most cases, increments by 1 for each new tab or window that is opened by the user, although in a few cases, numbers appeared to be skipped. The 'TS#' stream contains a list of the GUIDs of the tabs or windows that are currently open (or were if the entire session has been closed). The GUIDs are broken in to four sections and are displayed as ########-####-####-############. Figure 3 shows four 16-byte GUIDs that were open in the last browsing session.

```
00000000: EF 50 89 E8 BA 12 E0 11 86 80 00 50 56 C0 00 08  ïP‰è°.à.†€.PVÀ..
00000010: A5 67 CA 16 BB 12 E0 11 86 80 00 50 56 C0 00 08  ¥gÊ.».à.†€.PVÀ..
00000020: A7 67 CA 16 BB 12 E0 11 86 80 00 50 56 C0 00 08  §gÊ.».à.†€.PVÀ..
00000030: 4B 66 0A 23 BB 12 E0 11 86 80 00 50 56 C0 00 08  Kf.#».à.†€.PVÀ..
```

Figure 3 (Sample 'TS0' Stream from Recovery Store File)

The first eight bytes of each GUID are stored in little endian in a group of four bytes, two bytes and two bytes while the last eight bytes of the GUID are stored in big endian. In order to associate the data in the 'TS#' stream with tab data files found on the system, some translation needs to occur. For example, the first GUID shown in Figure 3 is displayed in the TS0 stream as 0xEF 50 89 E8 BA 12 E0 11 86 80 00 50 56 C0 00 08; which translates to 0xE8 89 50 EF 12 BA 11 E0 86 80 00 50 56 C0 00 08. Therefore, the file '{E88950EF-12BA-11E0-8680-005056C00008}.dat' should be associated with this recovery store file.

3.3.2 The 'FrameList' Stream

The format of the 'FrameList' stream is not entirely understood. Each open window is represented by 12 bytes of data in three 4-byte chunks. The first four bytes indicates the window number, shared with the # in the 'TS#' stream. The second four bytes were 0x00000001 in each circumstance. The final four bytes of the first window entry varied between test platforms, whereas the final four bytes of each subsequent window entry remained 0x00000004 across all platforms. These final four bytes of the first window entry may be 0x50000085 on one

computer, while they may be 0x00000005 on another computer under the same circumstances. While this changed between platforms, the final four bytes remained the same in most circumstances throughout recovery store files per computer.

It is possible to detect the use of InPrivate browsing through the 'FrameList' stream by examining the least significant bit of the last 4 bytes of the first window entry. When InPrivate browsing is used, 0x40 (64) is added to the least significant bit. For example, if the last 4 bytes of the first window entry are 0x50000085 (Figure 4), the last four bytes of the first window entry will be 0x500000C5 when InPrivate browsing is used (Figure 5). The 'FrameList' stream created by Internet Explorer 9 Beta also appears to include the GUID of the currently active tab for each window (Figures 6-7).

```
00 00 00 00 01 00 00 00 85 00 00 50
```

Figure 4 (FrameList Stream with Single Window from Internet Explorer 8)

```
00 00 00 00 01 00 00 00 C5 00 00 50
```

Figure 5 (FrameList Stream with Single Window from Internet Explorer 8 InPrivate Browsing)

```
00 00 00 00 01 00 00 00 85 00 00 50 01 00 00 00
01 00 00 00 04 00 00 00 02 00 00 00 01 00 00 00
04 00 00 00 03 00 00 00 01 00 00 00 04 00 00 00
04 00 00 00 01 00 00 00 04 00 00 00
```

Figure 6 (FrameList Stream with Multiple Windows from Internet Explorer 8)

```
00 00 00 00 01 00 00 00 05 00 00 10 68 A5 87 CF
A5 45 E0 11 8B CF 00 50 56 C0 00 08 01 00 00 00
01 00 00 00 04 00 00 00 FB 92 4E D7 A5 45 E0 11
8B CF 00 50 56 C0 00 08 02 00 00 00 01 00 00 00
04 00 00 00 E6 8A 09 DE A5 45 E0 11 8B CF 00 50
56 C0 00 08
```

Figure 7 (FrameList Stream from Internet Explorer 9)

3.3.3 The 'ClosedTabList' Stream

The 'ClosedTabList' stream contains a list of the GUIDs for the tabs used in the browsing session, but were closed prior to closing the entire window. These GUIDs are stored in the same format as those stored in the 'TS#' stream. Even when a tab closed, the associated tab data file remains on the system until the user

exits Internet Explorer (Figure 8).

```
00000000: 64 AE B4 0A BB 12 E0 11 86 80 00 50 56 C0 00 08  d®´.».à.†€.PVÀ..
00000010: B2 35 A6 2C BB 12 E0 11 86 80 00 50 56 C0 00 08  ²5¦,».à.†€.PVÀ..
00000020:
```

Figure 8 (Sample 'ClosedTabList' Stream from Recovery Store File)

3.3.4 The '{0B00252A-8D48-4D0B-7B79887F2B96}' Stream

The '{0B00252A-8D48-4D0B-7B79887F2B96}' stream is a property set that usually contains three properties (Figure 9).

The first common property value in this property set has a numeric ID of 0x00000002 and a type value of VT_UI4 (4-byte unsigned integer). The value of this property is initially set to 0x00000005. When the browser crashes and the files ACR files remain in the '\Active' folder, this value remains 0x00000005. When the browser closes without process failure and the ACR files are moved to the '\Last Active' folder, this value is 0x00000006.

The second common property value in this property set has a numeric ID of 0x00000003 and a type value of VT_CLSID (CLSID). This value should be the same as the GUID of the recovery store file.

The final common property value in this property set has a numeric ID of 0x00000007 and a type value of VT_CLSID (CLSID). When the recovery store file is first created, this value contains a value of the GUID of the recovery store file minus a value of 2 to 4 in the least significant nibble (for example a value of 93E43B49-3931-11E0-8FE9-000C29EF1366 in 0x00000003 may show a value of 93E43B46-3931-11E0-8FE9-000C29EF1366 in 0x00000007), meaning that this GUID was created 200 to 400 nanoseconds earlier than the GUID used as the file name.

As mentioned previously, when Internet Explorer is closed without process failure by the user, the ACR files are removed from the '\Active' folder and recreated in the '\Last Active' folder. When this occurs, the 0x00000003 value will reflect the *new* GUID of the recovery store file, while the 0x00000007 value will reflect the *previous* GUID of the recovery store file as it existed in the '\Active' folder. From these two values, the date/time the browsing session was opened and the date/time the browsing session was closed can be determined.

One other property ID of interest is 0x00000005. If present, 0x00000005 should have a type value of VT_UI4 (4 byte unsigned integer). In testing, the only time this value appeared in the recovery store files was when InPrivate browsing was used and on each occasion, it contained a value of 0x00000001.

ID	Name	Type	Value
0x00000002		VT_UI4	5 (0x00000005)
0x00000003		VT_CLSID	{8B6B1F1D-3932-11E0-8FE9-000C29EF1366}
0x00000007		VT_CLSID	{8B6B1F1A-3932-11E0-8FE9-000C29EF1366}

Figure 9 (The '{0B00252A-8D48-4D0B-7B79887F2B96}' Stream of a Recovery Store File as Viewed in CFX)

3.4 The Tab Data Files

The tab data files contain more detailed information about the history of each tab in a browsing session (Figure 10). As stated previously, one tab data file is created for each tab that is opened within the browsing session. At a minimum, each tab data file contains a minimum of two streams; the 'TravelLog' stream and the '{0B00252A-8D48-4D0B-7B79887F2B96}' stream. Additional streams are created for each page that is loaded within the tab and follow the naming convention 'TL#' where the # is a unique number starting at 0 and incrementing by 1 for each new page that is loaded. A 'TL#' stream is not always created until the next page is loaded. This will be discussed in more detail below.

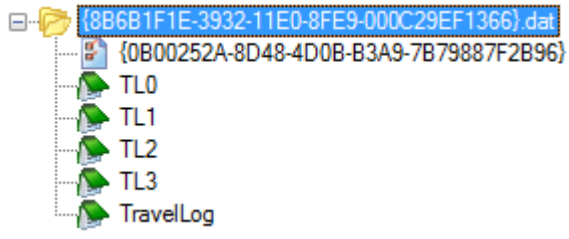

Figure 10 (Tab Data File Streams as Viewed in CFX)

3.4.1 The 'TL#' Stream

The 'TL#' stream contains detailed information about each page that is loaded within the tab. The numbering for the 'TL#' stream starts at 0 and in most cases, increments by 1 for each new tab that is opened by the user, although in a few cases, numbers appeared to be skipped. A 'TL#' stream is not always immediately created when a new page is opened within the tab. Consequently, one may encounter a tab data file that contains one less 'TL#' stream than it appears it should or none at all if only one page was opened. If a 'TL#' stream is not created immediately, once the next page is loaded within the tab, a 'TL#' stream will be created for the previous page. If no 'TL#' streams are present, the URL of the first and only paged opened will still be stored in the property set within the

'{0B00252A-8D48-4D0B-7B79887F2B96}' stream discussed later.

The information stored in these streams varies among pages. At minimum, the full URL and page title are stored at the beginning of the stream. Other data stored inside this stream can include additional frames that are loaded within the page, links to content within the page and default text within text boxes on the page depending on the page content.

Viewing these streams in a hex editor, it is clear the streams contain a mix of Unicode strings and binary data. However, it is the Unicode strings that should interest us. The binary data may be a mix of information stored by Internet Explorer, data stored as part of the compound file format, or slack space within the compound file 'sector'. Because the Unicode and binary data are conflated, a hex editor and CFX are not the most efficient means of examining these streams. FTK does an excellent job of extracting the Unicode data when the stream is viewed using the 'View Files in Filtered Text Format' option. A portion of a sample 'TL#' stream viewed using FTK's 'Filtered Text Format' option can be seen in Figure 11.

```
http://www.cnn.com/
CNN.com - Breaking News, U.S., World, Weather,
Entertainment & Video News
http://www.cnn.com/
http://www.cnn.com/
http://www.cnn.com/
http://www.cnn.com/
http://www.cnn.com/?fb_xd_fragment#?=&cb=f29c9ce8c7e4408&r
elation=parent&transport=fragment&frame=f23f6a82334bd84
http://www.cnn.com/?fb_xd_fragment#?=&cb=f29c9ce8c7e4408&r
elation=parent&transport=fragment&frame=f23f6a82334bd84
#?=&cb=f29c9ce8c7e4408&relation=parent&transport=fragment&
frame=f23f6a82334bd84
http://www.cnn.com/
http://www.cnn.com/?fb_xd_fragment#?=&cb=f29c9ce8c7e4408&r
elation=parent&transport=fragment&frame=f23f6a82334bd84
……
```

Figure 11

(Sample 'TL#' Stream Viewed Using FTK's 'Filtered Text Format' Option)

As shown in Figure 11, when viewed in this manner, the first line of the 'TL#' stream contains the full URL of the website. The second line contains the title of the website. Subsequent lines display additional information about page content as described above.

One additional artifact of interest noted in the 'TL#' streams is the behavior of Internet Explorer when a page is opened from a link on another page. In the example below (Figure 12), the search term 'Forensic Focus' was used in Google

and the first search hit was opened in a new tab by right clicking and selecting open in new tab. The URL http://www.google.com appears twice in the 'TL#' stream containing the information for the tab in which http://www.forensicfocus.com was opened. In addition, the full URL, including the search term used in Google, also appears in the stream.

```
http://www.forensicfocus.com/
Digital Forensics - Digital Forensics, Computer Forensic
Training, eDiscovery
http://www.forensicfocus.com/
http://www.forensicfocus.com/
http://www.forensicfocus.com/
http://www.google.com/
http://www.forensicfocus.com/
http://www.google.com/#sclient=psy&hl=en&q=forensic+focus&
aq=0&aqi=g4g-
o1&aql=f&oq=&pbx=1&bav=on.2,or.&fp=ce4eb09fec0d07a5
….
<a href="http://www.forensicfocus.com"
target="_blank"><img
src="http://www.forensicfocus.com/images/other/forensic-
focus-button.gif" alt="Forensic Focus" border="0" /></a>
http://www.google.com/
```

Figure 12 (Sample 'TL#' Stream Opened in New Tab)

These artifacts also appear if the link is opened within the same tab. While the location of this referring page information seems to vary slightly between pages, the last Unicode string always appears to be the URL of the referring page when the link is opened in a new tab, when appropriate.

3.3.5 The 'TravelLog' Stream

The 'TravelLog' stream contains the tabs back/forward information. Data is stored as 4-byte integers in little endian that indicates the order the 'TL#' information should be displayed in when the user uses Internet Explorer forward or back options. For example, if the user had navigated to three websites in a single tab, 'TL0', 'TL1', and 'TL2 streams should exist and the travel log may appear as it does in Figure 13.

```
00000000: 00 00 00 00 01 00 00 00 02 00 00 00    ............
```

Figure 13 (Sample 'TravelLog' Stream as Viewed in CFX)

As shown in Figure 13, the proper order of the 'TL#' information is 0x00000000, 0x00000001, 0x00000002. If property ID 0x00000004 in the '{0B00252A-8D48-

4D0B-7B79887F2B96}' stream (discussed next) (which contains the currently displayed page number) contained the value 0x00000001, the website from 'TL0' would be displayed in Internet Explorer's 'Previous' menu. The website from 'TL1' would be displayed as Internet Explorer's current page while the website from 'TL2' would be displayed in Internet Explorer's 'Next' menu.

3.3.6 The '{0B00252A-8D48-4D0B-7B79887F2B96}' Stream

The '{0B00252A-8D48-4D0B-7B79887F2B96}' stream is a property set with the same GUID as that stored in the recovery store files that usually contains three properties (Figure 14).

As with the recovery store files, the first common property value in this property set has numeric ID of 0x00000002 and a type value of VT_UI4 (4 byte unsigned integer). When the browser crashes and the files ACR files remain in the '\Active' folder, this value is 0x00000005. When the browser closes without process failure and the ACR files are moved to the '\Last Active' folder, this value is 0x00000006. The second common property value in this property set has numeric ID of 0x00000003 and a type value of VT_LPWSTR (Unicode string). This value should be the current URL of the tab. The final common property value in this property set has numeric ID of 0x00000004 and a type value of VT_UI4 (4 byte unsigned integer). This value should contain the number of the active 'TL#' stream. For example, if the current tab is that stored under the 'TL3' stream, property 0x00000004 should read 0x00000003. Other property IDs (0x00000007 and 0x00000008) were also occasionally seen in testing and were both a type value of VT_UI4 (4 byte unsigned integer). At this time their significance is unknown.

ID	Name	Type	Value
0x00000002		VT_UI4	5 (0x00000005)
0x00000004		VT_UI4	3 (0x00000003)
0x00000003		VT_LPWSTR	http://www.staples.com/

Figure 14 (The '{0B00252A-8D48-4D0B-7B79887F2B96}' Stream of a Tab Data File as Viewed in CFX)

4. FILES OPENED IN INTERNET EXPLORER

Although the most common use for Internet Explorer is web browsing, Internet Explorer can also be used to view files on the local machine. Similar to web browsing, opening files from the local machine causes Internet Explorer to create recovery store and tab data files, although obviously the information stored within varies between local files and web browsing.

One common example of such an action might be opening Multipurpose Internet Mail Extension (MIME) Hypertext Markup Language (MHTML) (.mht) or web archive files. MHTML files allow the user to save an entire web page and its resources to a single file, which can then be accessed offline at a later date or sent to another user. The .mht format is the default format using the 'Save as' function in Internet Explorer.

Notable differences in the tab data file include property ID 0x00000003 in the '{0B00252A-8D48-4D0B-7B79887F2B96}' stream, which will store the full path of the file instead of the URL and the data stored in the 'TL#' stream for the tab in which the .mht file was opened (Figure 15).

```
Users
Users
@shell32.dll,-21813
[[blinded]]
[[blinded]]
Desktop
Desktop
@shell32.dll,-21769
|  Google.mht
Google.mht
Google
Users
Users
@shell32.dll,-21813
john
john
Desktop
Desktop
@shell32.dll,-21769
|  Google.mht
Google.mht
mhtml:file://C:\Users\[[blinded]]\Desktop\Google.mht
file:///C:/Users/[[blinded]]/Desktop/Google.mht
mhtml:file://C:\Users\[[blinded]]\Desktop\Google.mht
…
```

Figure 15 (Sample 'TL#' Stream From .mht File)

As seen in Figure 15, the full path to the file, the page title and the user account along with other information is stored in the 'TL#' stream.

Another instance in which a local file may be opened in Internet Explorer is when Internet Explorer is used as an image viewer. As with .mht files, property ID 0x00000003 in the '{0B00252A-8D48-4D0B-7B79887F2B96}' stream will store the full path of the file and the 'TL#' stream will contain information similar to what is shows in Figure 15.

5. MALWARE

It is not uncommon for malware to open hidden Internet Explorer windows to access malicious sites, open command and control channels or simply increase the hit count of a website. On a test machine, we opened a hidden Internet Explorer window to http://www.google.com using the VB code "Shell Environ("programfiles") & "\Internet Explorer\iexplore.exe http://www.google.com", vbHide". Analysis of the C:\Users\<user>\AppData\Local\Microsoft\Internet Explorer\Recovery\Active directory revealed the same artifacts were generated with the same content as when an Internet Explorer window was opened to http://www.google.com in a traditional manner.

Knowing how and where Internet Explorer stores and verifies ACR files also presents an interesting mechanism for redirecting users to malicious websites. By simply copying ACR files containing a malicious URL to the C:\Users\<user>\AppData\Local\Microsoft\Internet Explorer\Recovery\Active directory and modifying the HKCU\Software\Microsoft\Internet Explorer\Recovery\Active registry key, the user will be prompted to restore the last browsing session to the malicious site.

6. DIFFERENCES BETWEEN INTERNET EXPLORER 8 AND 9

Very little has changed with Automatic Crash Recovery between Internet Explorer 8 and 9. Perhaps the single largest change took place in the 'FrameList' stream of the recovery store file. While the 'FrameList' stream in Internet Explorer 8 only contained a list of the window numbers, the 'FrameList' stream in Internet Explorer 9 also includes the GUIDs of the tab data file active for that window (Figure 16).

```
00000000: 00 00 00 00 01 00 00 00 05 00 00 50 44 44 AB 8A ...........PDD«Š
00000010: FC 52 E0 11 AE CD 00 50 56 C0 00 08 01 00 00 00 üRà.®Í.PVÀ......
00000020: 01 00 00 00 04 00 00 00 2A C4 E0 B9 FC 52 E0 11 ........*Äà¹üRà.
00000030: AE CD 00 50 56 C0 00 08                        ®Í.PVÀ..
```

Figure 16 (FrameList Stream from Internet Explorer 9)

The only other significant change took place in the '{0B00252A-8D48-4D0B-7B79887F2B96}' stream of the recovery store and tab data files. While property ID 0x00000002 was initially set to 0x00000005 in Internet Explorer 8 and only reset to 0x00000006 when Internet Explorer was closed normally, this property appears to be set to 0x00000006 at all times in Internet Explorer 9.

7. ACR FILES IN UNALLOCATED SPACE

Only the most recently closed session information will remain in the '\Last Active' folder. Once a more recent session is closed properly, the corresponding ACR files will be moved from the '\Active' folder to the '\Last Active' and the previous ACR files in the '\Last Active' will be deleted. In order to obtain the most evidence from ACR files, it is vitally important to be able to find and carve them from unallocated space.

The file header for the compound file is 0xD0 CF 11 E0 A1 B1 1A E1 (Microsoft, 2012a). However since the compound file format is not unique to ACR files, searching only for this header will likely create a large number of false positives when searching unallocated space. Using other static fields in the file header, it is possible reduce the number of false positives. Table 1 lists the static fields following the file signature and their byte offset.

Table 1

Byte Offset	Name	Value
0x0008	Header CLSID	0x0000000000000000
0x0018	Minor Version	0x003E
0x001A	Major Version	0x0003
0x001C	Byte Order	0xFFFE
0x001E	Sector Size	0x0009
0x0020	Mini Stream Sector Size	0x0006
0x0022	Reserved	0x000000000000
0x0028	Number of Directory Sector	0x00000000

Using these static fields, we can build a search string of 0xD0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3E 00 03 00 FE FF 09 00 06 00 00 00 00 00 00 00 00 00 00 00. This 44-byte search pattern will reduce false positives, but will still locate most compound files. In all files reviewed, the first time the Unicode text 'http' appeared in the binary data was 2,500 to 3,500 bytes from the file header. The GUID of the ACR property sets, 0B00252A-8D48-4D0B-7B79887F2B96, appears to be unique to these files and will also help reduce false positives when searching.

Carving a compound file format file from unallocated space can be more complicated and time consuming than other file types because of the random nature of the file format and the fact that it does not contain a file footer. However it is still possible to accomplish using information from the file header and the file's FAT.

Sector 0x1C contains a 2-byte value indicating the sector size used in the compound file. This value should always be 0x0009 indicating 512 bytes (Figure 17).[3]

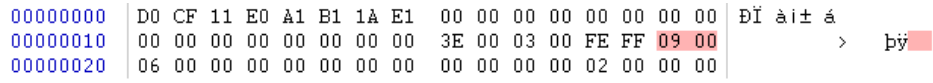

Figure 17 (Sector Size)

Sector 0x2C contains a 4 byte value indicating the number of FAT sectors in the file (Figure 18) (Microsoft, 2012a). Each 512-byte FAT sector can address up to 128 sectors within the file; since each sector is 512 bytes, each FAT sector accounts for up to 65,536 bytes of a file. For example, if sector 0x2C's value is 0x0002, the file must be larger than 65,536 bytes and smaller than 131,073 bytes.

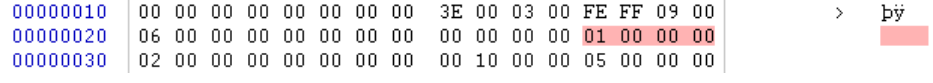

Figure 18 (Number of FAT Sectors)

Sector 0x4C contains a 4-byte value containing the sector number of the first FAT sector (Figure 19) (Microsoft, 2012a). This can be converted to an offset by using (sector number+1) x 512. In this case, the first FAT sector begins at (3+1) x 512 = 2048 or 0x800. Since it has already been determined that this file contains only one FAT sector, the entire FAT must be located from 0x800 to 0x9FF (Figure 20).

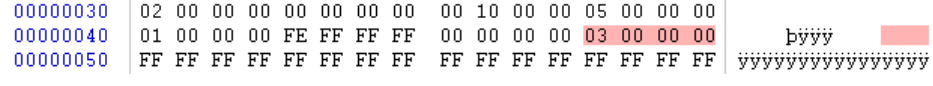

Figure 19 (First FAT Sector Number)

Much like the FAT file system on storage media, the FAT of a compound file contains a linked chain of sectors. Each 4-byte FAT entry will contain the next sector in the chain or reserved value as seen in Table 2 (Microsoft, 2012a).

Value	Description
0x00000000 – 0xFFFFFFF9	Next Sector in Chain
0xFFFFFFFA	Max Regular Sector Number
0xFFFFFFFC	DIFAT Sector
0xFFFFFFFD	FAT Sector
0xFFFFFFFE	End of Chain
0xFFFFFFFF	Unallocated Sector

Table 2

To determine the total size of the file we should count the number of bytes from the beginning of the FAT to the last allocated sector (Figure 20). The file size must be the number of bytes from the beginning of the FAT to the last allocated sector divided by 4 (because FAT each entry is four bytes), plus one (because the header is not included in the FAT) multiplied by 512 (the sector size). In other words, (Number of Bytes / 4 + 1) x 512.

Figure 20 (FAT)

In the example shows in Figure 21, there are 40 bytes from the beginning of the FAT to the last allocated sector, which indicates there are nine allocated sectors in this file we should add one additional sector to include the header and multiply by 512 bytes and the file size should be 5,120 bytes, which is confirmed by Windows. With the total file size known it is now possible to carve the file from unallocated space.

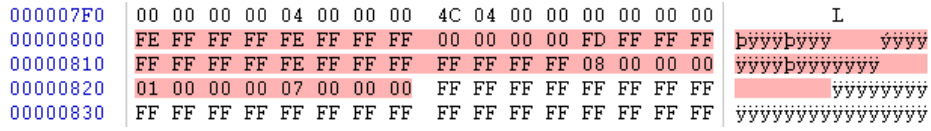

Figure 21 (Allocated Sectors)

If sector 0x2C indicates the file contains more than one FAT sector (Figure 22), the Double-Indirect File Allocation Table (DIFAT) must be used (Figure 23). The DIFAT is a directory of all the FAT sectors in the compound file and their offsets (Microsoft, 2012a). Sector 0x4C, the 4-byte value containing the sector number of the first FAT sector mentioned previously is actually DIFAT[0]. The last 432 bytes of the 512-byte header contain DIFAT[1] through DIFAT[108]. In the case of Active Crash Recovery files, no file should ever come close to requiring 109 DIFAT entries.

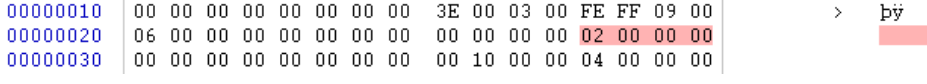

Figure 22 (Multiple FAT Sectors)

Because of the nature of compound files, every sector addressed by a FAT sector must be allocated before a new FAT sector is created. Accordingly, it is safe to assume that each FAT entry in every FAT sector except the last accounts for a fully allocated 512-byte sector within the file. For example, if sector 0x4C indicates there are two FAT sectors, one must be completely allocated. Therefore, the file contains at least 65,536 bytes ((512 / 4) x 512). The important entry in the DIFAT when determining the complete file size is the last entry. Since the header indicated this file contains two FAT sectors, the last entry should be DIFAT[1], which can be confirmed by examining offset 0x50 (DIFAT[1]). This contains a value of 0x0000003B and offset 0x54 (DIFAT[2]) indicates an unused value of 0x0FFFFFFF. 0x3B = 59; using the formula (sector number + 1) x 512, the second and final FAT sector should be located at offset 30,720 or 0x7800.

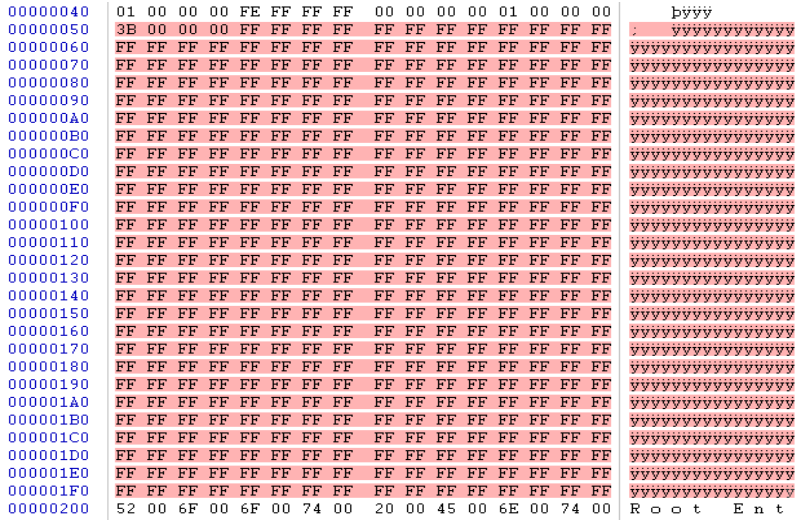

Figure 23 (DIFAT)

Once the last FAT sector has been located, calculating the file size the last FAT sector is done in the same manner as it was with only one FAT sector. We should count the number of bytes from the beginning of the last FAT sector to the last allocated sector (Figure 24), divided by 4 (because FAT each entry is four bytes), plus one (because the header is not included in the FAT) multiplied by 512 (the sector size). In other words, (Allocated Sectors / 4 + 1) x 512.

Figure 24 (Second FAT Sector)

In the example shown in Figure 25, there are eight bytes from the beginning of the last FAT sector to the last allocated sector, which indicates there are two allocated sectors, accounting for 1,024 bytes of the file plus an additional 512 bytes for the header for a total of 1,536 bytes. It was already determined that each prior FAT sector accounts for 65,536 bytes. In this case, there was only one prior FAT sector. So 65,536 bytes can be added to the 1,536 bytes of the last FAT sector and header. The final total in this case is 67,072 bytes, which is confirmed by Windows.

This process can be expressed using the formula (((Total Number of FAT Sectors – 1) x 512 / 4) x 512) + ((Number of Bytes in the Last FAT Sector / 4 + 1) x 512).

Using the previous example, the equation would be (((2 – 1) x 512 / 4) x 512) + ((8 / 4 + 1) x 512) = 67,072 bytes.

Figure 25 (Second FAT Sector)

Since files carved from unallocated space will no longer be associated with their file names (their GUIDs), it will not be possible to associate the tab data files with their respective recovery store files.

8. RECOVERRS

Based on the research of Internet Explorer's Automatic Crash Recovery files, two command line applications were developed called RipRS and ParseRS; collectively, these tools are known as RecoverRS.

RipRS is designed to extract ACR files from a raw disk image using known decimal offsets. A list of known offsets can be obtained by using the search string discussed in the above section (titled 'ACR Files in Unallocated Space') using programs such as EnCase or FTK. Using these known offsets, RipRS uses the methodology discussed in the above section titled 'ACR Files in Unallocated Space' to determine the compound file's size. RipRS first searches the compound file for the GUID that is unique to ACR files then searches the ACR file for strings unique to either recovery store files or tab data files to determine the file type. Once RipRS has determined the ACR file type, the file is written to the output directory using the naming convention RecoveryStore.{offset<offset>}.dat or {offset<offset>}.dat for recovery store files and tab data files respectively.

ParseRS is designed to extract browsing information from ACR files; either those found on the system or those carved from unallocated space by RipRS. As mentioned previously, if ACR files are carved from unallocated space, information linking the tab data files with their respective recovery store files and some date/time information will be lost.

RecoverRS can be downloaded from http://www.jtmoran.com/tools.

9. CONCLUSION

While the information recovered from the Automatic Crash Recovery files may not replace the bounty of information obtained from the cookies and the index.dat files of Internet Explorer, it provides yet another tool for examiners to retrieve valuable evidence. As the Automatic Crash Recovery files seem to be a lesser known source of information, these files may provide valuable data when other

sources are not available as well as to supplement information found in other locations.

REFERENCES

Leach, P., Mealling, M., & Salz, R. (2005, July). *A Universally Unique IDentifier (UUID) URN Namespace* (RFC 4122). Internet Engineering Task Force. Retrieved June 25, 2012, from http://www.ietf.org/rfc /rfc4122.txt

Microsoft Corporation. (2008, March). *Automatic Crash Recovery: Windows Internet Explorer 8 Beta 1 for Developers*. Retrieved June 25, 2012, from http://www.softwaretipspalace.com/whitepapers/microsoft /Automatic%20Crash%20Recovery.pdf

Microsoft Corporation. (2012a, March 28). *[MS-CFB]: Compound File Binary File Format*. Retrieved June 25, 2012, from http://download.microsoft .com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/[MS-CFB].pdf

Microsoft Corporation. (2012b, March 28). *[MS-OLEPS]: Object Linking and Embedding (OLE) Property Set Data Structures*. Retrieved June 25, 2012, from http://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/[MS-OLEPS].pdf

Parsonage, H. (2010, July). *The Meaning of LIFE*. Retrieved June 29, 2012, from http://computerforensics.parsonage.co.uk/downloads /TheMeaningofLIFE.pdf

ABOUT THE AUTHORS

John Moran received his Bachelor's Degree in Computer Forensics from Champlain College in 2011. He holds CFCE, EnCE, CCNA and CEH certifications. John currently works for the County of Cumberland, Maine as a Public Safety Software Specialist and is also a certified police officer.

Douglas A. Orr received his Ph.D from Washington State University in Criminal Justice with a concentration in Political Psychology. He currently serves as an adjunct professor with Chaplain College in their Master of Science Digital Forensic Management Program. Dr. Orr is also a commissioned police detective assigned to the Special Investigations Unit of the Spokane Police Department in Spokane, Washington. He currently serves as their chief computer forensic examiner.