Extracting the Windows Clipboard from Memory

By

James Okolica and Gilbert Peterson

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2011 USA

New Orleans, LA (Aug 1st - 3rd)

available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

Extracting the windows clipboard from physical memory

James Okolica[*], *Gilbert L. Peterson*

Air Force Institute of Technology, Wright-Patterson AFB, OH, United States

Keywords:
Windows clipboard
Memory forensics
Reverse code engineering
Windows operating system
Digital forensics

A B S T R A C T

When attempting to reconstruct the events leading up to a cyber security incident, one potentially important piece of information is the clipboard (Prosise et al., 2003). The clipboard has been present in Windows since Windows 3.1 and is the mechanism for transferring information from one application to another through copy and pasting actions. Being able to retrieve the last file copied or the last password used may provide investigators with invaluable information during a forensic investigation. This paper describes the Windows clipboard structure and the process of retrieving copy/paste information from Windows XP, Vista, and Windows 7 (both 32 bit and 64 bit) memory captures with data from applications including Notepad, Microsoft Word, and Microsoft Excel.

1. Introduction

While there has been significant research into memory forensics (Dodge et al., 2010; Dolan-Gavitt, 2008; Okolica and Peterson, 2010; Schuster, 2006; Walters and Petroni, 2007), to date there has not been research into extracting Windows clipboard evidence from a memory capture. Memory analysis is a key element of digital forensics. A computer's memory provides the most up to date snapshot of the machine's state: programs are loaded into memory before executing; configuration information is either loaded from disk into memory or entered directly into memory from the keyboard; and active network connections are stored in memory (Okolica and Peterson, 2010). Leveraging memory to determine the state of the machine at the time of the incident is often critical to success.

In addition to processes, configuration, and network activity, clipboard contents are also critical to forensic analysis (Prosise et al., 2003). Clipboard contents often provides valuable forensic data, including user passwords, copied sections of classified documents, and incriminating urls. However, while there have been a number of tools written to capture process, configuration and/or network activity (Betz, 2005; Okolica and Peterson, 2010; Schuster, 2006; Walters and Petroni, 2007), there have not been any tools written to extract Windows clipboard data from memory dumps. The existing command line incident response tool that does display clipboard information is pclip.exe (Carvey, 2007), which is accessible from sourceforge.net (http://sourceforge.net/projects/unxutils).

The clipboard has been a part of the Windows O/S family since Windows 3.1. Windows uses the clipboard to transfer information between user applications. As a result, it bridges the gap between O/S user functions (handled by *user32.dll*) and O/S kernel functions (handled by *win32k.sys*). This dichotomy differentiates the clipboard from the process, configuration, and network activity, which are O/S kernel functions. As a result, finding and extracting clipboard data requires slightly different techniques.

The remainder of the paper presents an overview of the memory analysis work already done as well as previous research into the Windows clipboard. It then goes on to describe a methodology for extracting the Windows clipboard from multiple Windows operating systems. Finally, it concludes by successfully applying the methodology to several memory dumps from multiple Windows operating systems.

[*] *Corresponding author.*
E-mail addresses: jokolica@afit.edu (J. Okolica), gilbert.peterson@afit.edu (G.L. Peterson).

1742-2876/$ – see front matter © 2011 Okolica & Peterson. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2011.05.014

2. Background

2.1. *Memory forensics*

Historically, the National Institute of Justice (NIJ) recommended digital forensic process was to pull the plug and image and analyze the file systems (NIJ, 2009). While this process is valuable, it overlooks the architecture of computers. Before the computer executes a program, displays a file, or logs an activity, it first loads the information into *memory*. Memory forensics examines a memory capture from the seized computer.

Forensic memory analysis starts with collecting the memory from the target machine followed by parsing the memory dump into meaningful artifacts. Several tools include Schuster's *Ptfinder* (Schuster, 2006; Schuster, March 2, 2006) and Betz's *Memparser* (Betz, 2005). More recently, *Volatility* (Walters and Petroni, 2007) has emerged as an effective tool for parsing Windows XP memory dumps. Unfortunately, what the above tools have in common is a limitation to a particular operating system (and in many cases, service pack). *CMAT* (Okolica and Peterson, 2010) increased flexibility by parsing much of the same information out of a memory dump from any of the Windows NT family of operating systems (including Windows XP, Vista, and 7).

All of the existing tools are limited to operating system structures. While *Volatility* and *CMAT* provide network information, which does not reside in the kernel, the network driver, *tcpip.sys*, is still a system level driver. In contrast, the clipboard is a user-level data structure found in *user32.dll*. While there has been previous work on user-level structures (Stevens and Casey, 2010), it has focused on finding and carving unique signatures for command line history in memory dumps without tracking it back to the originating processes. This is different from the above tools that start with the processes and use information in them to find other forensic artifacts (network connections, open files, registry keys accessed, etc.).

2.2. *Windows clipboard*

The Windows clipboard is the mechanism that Microsoft Windows operating systems use to allow data to be shared between applications. It first appeared in Windows 3.1, although its functionality has greatly increased since then. Table 1 shows the standard formats used by the clipboard (Petzold, 1999). However, Microsoft also provides the ability for "private data formats", formats that are application specific (for example, fonts in a word processing program), and that could be registered so that other applications could transfer data in these formats (Petzold, 1999). Two private data formats that are used extensively are object link embedding (OLE) (0xC013) and dataobjects (0xC009).

For an application to send data to the clipboard, it first allocates a block of global memory using *GlobalAlloc*, *GlobalLock*, and *GlobalUnlock* (Fig. 1). It then opens the clipboard using *OpenClipboard*, empties it using *EmptyClipboard*, places the clipboard data using *SetClipboard* and then closes the clipboard using *CloseClipboard* (Microsoft.com,). While only one piece of

Table 1 – Predefined clipboard formats.

Constant	Value	Description
CF TEXT	0x0001	Text format. Each line ends with a cr/lf combination. Null-terminated
CF BITMAP	0x0002	A handle to a bitmap
CF METAFILEPICT	0x0003	Handle to a metafile picture format as defined by the METAFILEPICT structure
CF SYLK	0x0004	Microsoft Symbolic link format
CF DIF	0x0005	Software Arts' Data Interchange Format
CF TIFF	0x0006	Tagged-image file format
CF OEMTEXT	0x0007	Text format containing characters in the OEM character set. Each line ends with a cr/lf combination. Null-terminated
CF DIB	0x0008	A memory object containing bitmapinfo structure followed by the bitmap bits
CF PALETTE	0x0009	Handle to a color palette. Whenever an application places data in the clipboard that depends on or assumes a color paletter, it should place the palette in the clipboard as well
CF PENDATA	0x000A	Data for the pen extensions to Windows
CF RIFF	0x000B	Represents audio data more complex than can be represented in a CF_WAVE standard wave format
CF WAVE	0x000C	Represents audio data in one of the standard wave formats
CFx UNICODETEXT	0x000D	Unicode text format. Each line ends with a CR/LF combination. Null terminated.
CF ENHMETAFILE	0x000E	A handle to an enhanced meta file
CF HDROP	0x000F	A handle_t type HDROP that identifies a list of files
CF LOCALE	0x0010	The data is a handle to the locale identifier associated with text in the clipboard
CF DIBVS	0x0017	A memory object containing a bitmapvsheader structure followed by the bitmap color space information and the bitmap bits

```
hGlobal = GlobalAlloc (GHND | GMEM_SHARE, iLength + 1);
pGlobal = GlobalLock (hGlobal);

for (i = 0; i <wlength; i++)
*pGlobal++ = *pString++;

GlobalUnlock (hGlobal);

OpenClipboard (hwn);
EmptyClipboard();

SetClipboardData (CF_Text, hGlobal);

CloseClipboard();
```

Fig. 1 – Transferring Text to the Clipboard.

```
OpenClipboard (hwnd)
hGlobal = GetClipboardData (CF_TEXT);
```

Fig. 2 – Transferring Text from the Clipboard.

data can be present in the clipboard at any given time, it is possible to send and store that piece of data in multiple formats by performing multiple SetClipboardData functions. This allows applications that handle data in different ways to all have access to it (e.g., text in Microsoft Word with or without formatting). Once data is in the clipboard, the block of global memory belongs to the clipboard and other pointers to it become invalid. Getting data from the clipboard (Fig. 2) is even easier and involves opening the clipboard, determining which of the available clipboard formats to retrieve (this is an application specific task), retrieving a handle to the data, and then closing the clipboard.

2.3. *Windows data objects*

For copying more complex data than text to the clipboard, Windows makes available several APIs which make extraction much more difficult. The original method for exchanging data between applications was dynamic data exchange (DDE). In 1990, Microsoft released object linking and embedding (OLE) (Allan, 2001) enabling compound files. Compound files have most of the file in a primary format (for example, a Microsoft Word document) and smaller sections in one or more other formats (Microsoft Excel, Microsoft PowerPoint, text, etc.) either linked in (kept in it’s original, separate file) or embedded. Microsoft quickly extended OLE to a Compound Object Model (COM) architecture and then in 1994 released OLE 2.0 that sits on top of COM. OLE 2.0 added, among other things, uniform data transfer (UDT) and Drag and Drop (Microsoft.com). UDT and Drag and Drop enables the functionality used by the Windows Clipboard today to transfer files, images, and other objects between applications. For example, when a file is dragged from Windows Explorer to the Desktop, this is accomplished internally via the Windows clipboard. This functionality has changed over the years, first with the creation of ActiveX and most recently with the advent of the .NET framework. The result is a complex combination of legacy and new functions cobbled together to enable all of the functionality created and changed over the past twenty years to work together.

3. Methodology

Identifying the method required for extracting clipboard information from a Windows memory dump consists of four steps (Fig. 3). First, one or more of the functions in *user32.dll* or *win32k.sys* that accesses the clipboard data is found. The functions described in (Petzold, 1999) provide a good starting point for selecting specific functions. Then, we reverse engineer each function to identify the clipboard structures. The third step adds the ability to search for the structures into a memory analysis program. Finally, testing of the program against memory dumps from multiple Windows operating systems using textual data transferred to the clipboard from several programs verifies the programs functionality.

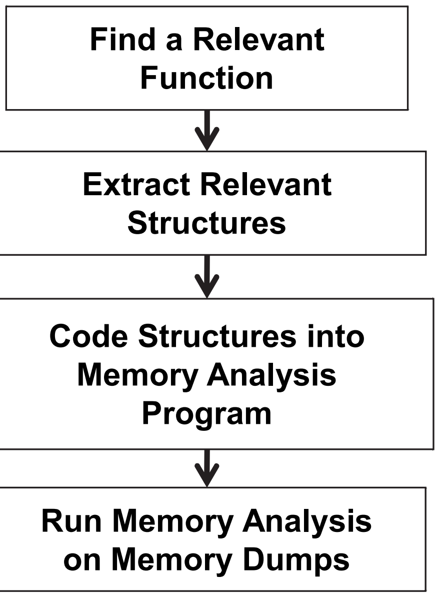
Find a Relevant Function
Extract Relevant Structures
Code Structures into Memory Analysis Program
Run Memory Analysis on Memory Dumps

Fig. 3 – A Process for Reverse Engineering Clipboard Data Structures.

3.1. *Clipboard functions*

While there is a large amount of documentation (Microsoft.com; netez.com; Petzold, 1999) on how to use and access the Windows clipboard via application programmer interfaces (APIs), there is no documentation on the underlying structures used in *user32.dll* to manage the clipboard. Therefore, the reverse engineering process begins with the functions in Fig. 1. It then analyzes the functions statically and dynamically.

The analysis process creates an analysis profile in a virtual machine. Using an application (e.g., Notepad) to transfer data to the clipboard provides an environment with known data. After capturing a memory dump and transferring the dump to the host machine, the dynamic analysis begins. Observe that the transfer of the memory dump is done such that the file is not loaded into the clipboard (e.g., by moving the file within Windows Explorer). Once the dump file is on the host machine and the virtual machine halted, the dynamic analysis process places instruction breakpoints at the beginning of the *GetClipboardData* function. Although not necessary, performing dynamic analysis while having a memory dump allows the investigators to make changes to the program at the same time they are performing reverse code analysis on the clipboard functions.

GetClipboardData begins with the standard saving of registers and resetting of the stack pointer. As shown in Fig. 4, it then makes a call to *NtUserGetClipboardData* to retrieve the pointer to the clipboard data. This is actually an interrupt that moves the process from user space to kernel space. *NtUserGetClipboardData* is a kernel function that begins by retrieving a pointer to the Windows Station object for the

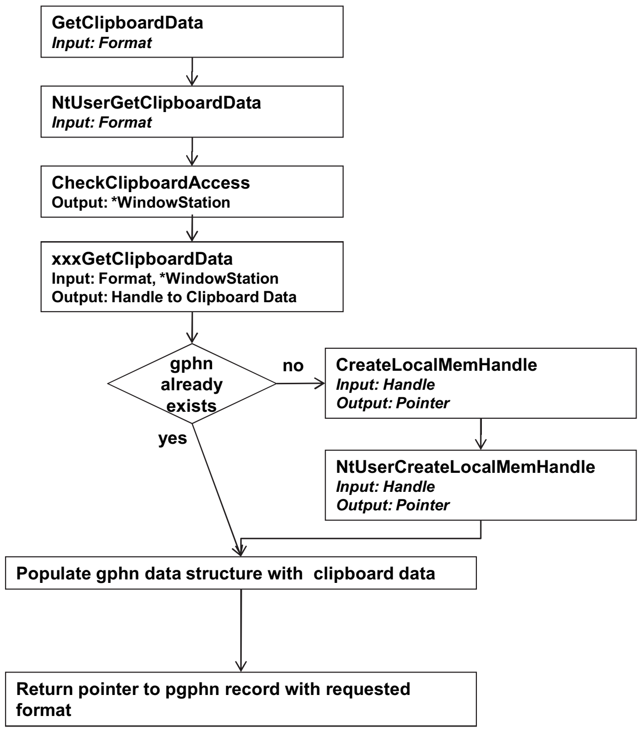

Fig. 4 – GetClipboardData Process Flow.

current thread. It then calls xxxGetClipboardData, passing along the pointer to the Windows Station and the requested clipboard format. *xxxGetClipboardData* then iterates through the different formats stored in the clipboard until it finds the matching one. The clipboarddata is then placed in allocated memory and returned back to *GetClipboardData*. If the clipboard data was already present in the process (either because it originated there or due to a previous paste operation), the symbol *gphn* points to the allocated memory. If not, new memory is allocated and the "pointer" returned by NtUserGetClipboardData is converted into an actual pointer by *NtUserCreateLocalMemHandle*.

gphn is the head of a linked list of clipboard records. There are four variables in each clipboard record (Table 2). The first is a pointer to the next element in the linked list (null if it is the last element in the list). The second, at offset 0x04 (0x08 on 64-bit machines) is the format of the current record. The third, at offset 0x08 (0x10 on 64-bit machines) is unknown. Finally, the fourth at offset 0x0c (0x18 on 64-bit machines) is a handle to the data. The code loops through the linked list until it finds the correct record. Once the data is ready for return to the requesting application, *GetClipboardData* allocates space on the heap. Finally, depending on the requested format, it loads the appropriate data into the space. For text, the handle is a unicode string. The function then sends this handle to the requesting application.

Table 2 – Clipboard structure.

32 Bit Offset	64 Bit Offset	Data type	Field name
0x00	0x00	gphn*	next
0x04	0x08	uint16_t	format
0x08	0x10	unknown	unknown
0x0c	0x18	void*	handle

gphn* means a pointer to the gphn structure and void* means an arbitrary pointer.

3.2. *Tools*

There are two open source tools that the clipboard structure extraction can be implemented in. The first, the Volatility Framework (The Volatility Framework,), is "a completely open collection of tools, implemented in Python under the GNU General Public License for the traction of digital artifacts from volatile memory (RMA) samples." that extract forensic artifacts from memory dumps. The second, CMAT (Okolica and Peterson, 2010) is a C program that extracts forensic artifacts from memory dumps or from a Xen virtual machine (Dodge et al., 2010). Depending on the specifics of the application, either tool would suffice. This study uses CMAT.

In order to reach the clipboard, the process on the user side shown in Fig. 5 iterates through each process sorted by session ID (Okolica and Peterson, 2010). For each process, CMAT first locates the loader table and then iterates through it to find *user32.dll*. If *user32.dll* is not loaded for a given process (e.g., because it has no user interface), then that process does not have access to the clipboard. Once the loader entry for *user32.dll* is found,the PDB file for *user32.dll* is downloaded from the Microsoft symbol server, assuming it is not already resident locally. Next, the offset for the *gphn* symbol is located in the PDB file. This offset provides a location to find *gphn* within the *user32.dll* virtual address space.

It is possible that the user closed the application after copying data to the clipboard. Recall that the clipboard bridges user space and kernel space. While each process has a local copy of the clipboard once it has accessed the clipboard functions, the kernel also has the clipboard. Therefore, until overwritten, clipboard data for a closed process is still

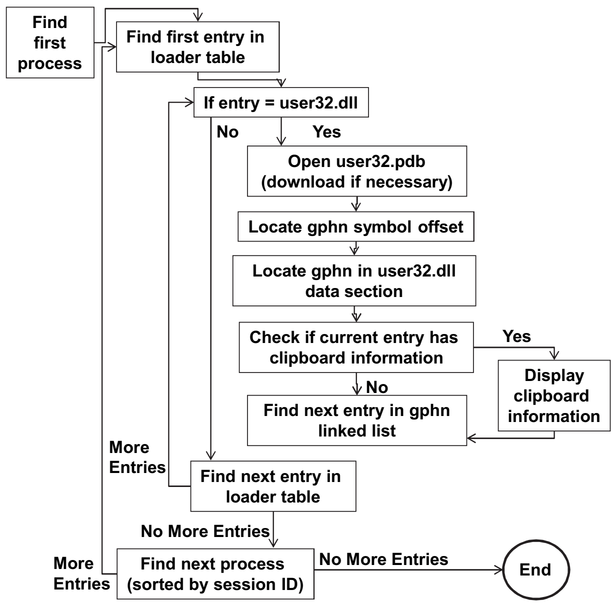
Find first process
Find first entry in loader table
If entry = user32.dll
No
Yes
Open user32.pdb (download if necessary)
Locate gphn symbol offset
Locate gphn in user32.dll data section
Check if current entry has clipboard information
Yes
Display clipboard information
No
Find next entry in gphn linked list
More Entries
Find next entry in loader table
No More Entries
More Entries
Find next process (sorted by session ID)
No More Entries
End

Fig. 5 – Clipboard Extraction Process Flow (User side).

available in the clipboard. Because there may be this data, the kernel module, *win32k.sys* is used to locate and retrieve the clipboard. As shown in Fig. 6, CMAT first locates *win32k.sys* in PsLoadedModuleList and then uses its debug section to retrieve *win32k.pdb*. Using *win32k.pdb*, the location of *gSharedInfo* is found. CMAT then iterates through the table of clipboard formats pointed to by an entry in the Windows Station. Once CMAT finds the appropriate format and its associated handle, CMAT converts the handle (if it was not already) into a memory pointer to the clipboard data. Finally, CMAT retrieves the clipboard.

3.3. *Datasets*

Two publically available memory dumps exist for testing and validating memory analysis tools. The Digital Forensics Research Workshop (DFRWS) 2008 Forensic Rodeo created a forensic challenge involving a trusted insider suspected of accessing proprietary data. As part of the challenge two Windows XP 32 bit memory dumps were created (Digital Forensics Research Workshop, 2008). In addition, the National Institute of Standards and Technology (NIST) developed the Computer Forensic Data Sets (CFReDS) datasets for digital evidence (National Institute of Standards and Technology). Part of this dataset contains memory images from Windows 2000, Windows 2003, Windows XP, and Windows Vista.

In addition to these two publically available datasets, the testing uses additional memory dumps that include known clipboard objects. There are two reasons for this. First, none of the publically available datasets were created with clipboard data in mind. Therefore, there is no way to know if the clipboard contains any information to find. Second, none of these datasets include Windows 7 or any 64 bit operating system. For completeness, we created additional memory dumps for Windows 7, Windows Vista, and Windows XP (both 32 bit and 64 bit). Each of these dumps include clipboard data from Notepad, Microsoft Word, and Excel. For this initial investigation, the only data types implemented are ASCII and Unicode strings. Implementation of additional data types such as OLE objects, and the reverse engineering of the dynamic link libraries and drivers that implement them, is left for future work.

4. Results

As shown in Table 3, the results are very good. Testing of the modified CMAT code on the DFRWS dataset uncovers the statement “pp -B -p -o out.pl file” from the clipboard. This is a command line statement for creating a standalone Perl program. When run on the CFReDS dataset, in two cases, it fails to find any clipboard data. Unfortunately, there is no way to know if there is clipboard data that was not found or if there is no clipboard data. In the third case, a clipboard entry does appear. Unfortunately, instead of being text, it is an object linking and embedding (OLE) private data format.

Testing of the modified CMAT code on the datasets created for testing the clipboard extraction produces perfect results. In

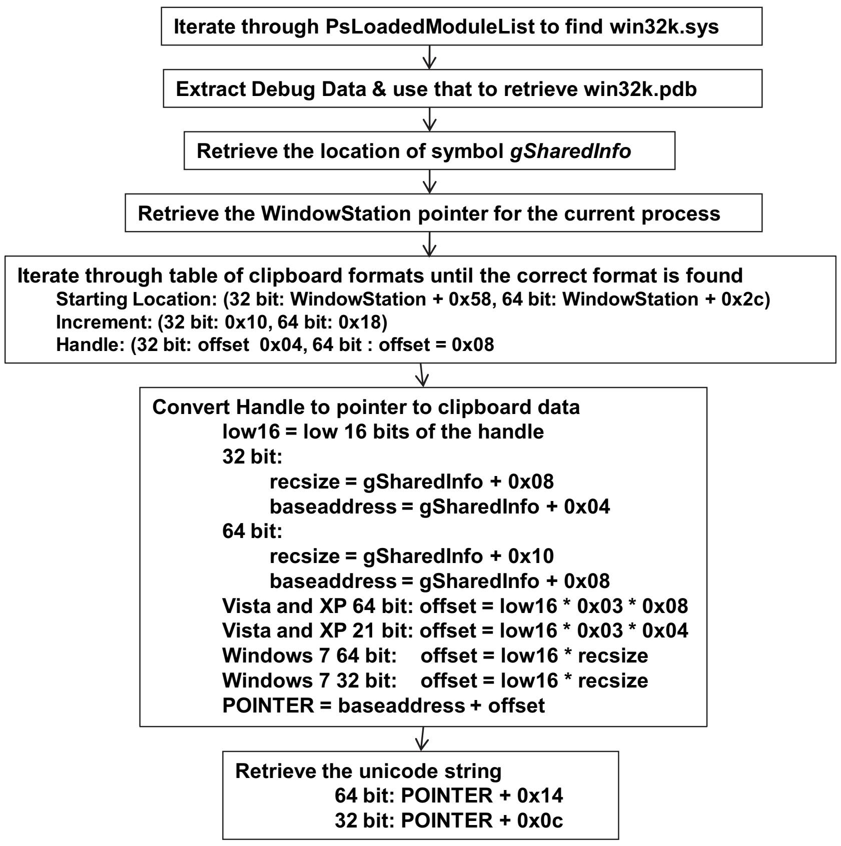
Iterate through PsLoadedModuleList to find win32k.sys
Extract Debug Data & use that to retrieve win32k.pdb
Retrieve the location of symbol gSharedInfo
Retrieve the WindowStation pointer for the current process
Iterate through table of clipboard formats until the correct format is found
Starting Location: (32 bit: WindowStation + 0x58, 64 bit: WindowStation + 0x2c)
Increment: (32 bit: 0x10, 64 bit: 0x18)
Handle: (32 bit: offset 0x04, 64 bit : offset = 0x08
Convert Handle to pointer to clipboard data
low16 = low 16 bits of the handle
32 bit:
recsize = gSharedInfo + 0x08
baseaddress = gSharedInfo + 0x04
64 bit:
recsize = gSharedInfo + 0x10
baseaddress = gSharedInfo + 0x08
Vista and XP 64 bit: offset = low16 * 0x03 * 0x08
Vista and XP 21 bit: offset = low16 * 0x03 * 0x04
Windows 7 64 bit: offset = low16 * recsize
Windows 7 32 bit: offset = low16 * recsize
POINTER = baseaddress + offset
Retrieve the unicode string
64 bit: POINTER + 0x14
32 bit: POINTER + 0x0c

Fig. 6 – Clipboard Extraction Process Flow (Kernel side).

Table 3 – Results.

Dataset	Memory image	Results
DFRWS2008	dfrws	"pp -B -p -o out.pl file" command found
CFReDS	vista-beta2.img	No Clipboard Data Found
CFReDS	xp-laptop-2005-06-25.img	No Clipboard Data Found
CFReDS	xp-laptop-2005-07-04-1430.img	Non-textual Clipboard Data Found (format:0xC013: OLE Private Data)
Generated	32 bit Vista w/Notepad	Notepad Clipboard Data found
Generated	32 bit Vista w/MS Word	Microsoft Word Clipboard Data found
Generated	32 bit Vista w/MS Excel	Microsoft Excel Clipboard Data found
Generated	64 bit Vista w/Notepad	Notepad Clipboard Data found
Generated	64 bit Vista w/MS Word	Microsoft Word Clipboard Data found
Generated	64 bit Vista w/MS Excel	Microsoft Excel Clipboard Data found
Generated	32 bit XP w/Notepad	Notepad Clipboard Data found
Generated	32 bit XP w/MS Word	Microsoft Word Clipboard Data found
Generated	32 bit XP w/MS Excel	Microsoft Excel Clipboard Data found
Generated	64 bit XP w/Notepad	Notepad Clipboard Data found
Generated	64 bit XP w/MS Word	Microsoft Word Clipboard Data found
Generated	64 bit XP w/MS Excel	Microsoft Excel Clipboard Data found
Generated	32 bit Windows 7 w/Notepad	Notepad Clipboard Data found
Generated	32 bit Windows 7 w/MS Word	Microsoft Word Clipboard Data found
Generated	32 bit Windows 7 w/MS Excel	Microsoft Excel Clipboard Data found
Generated	64 bit Windows 7 w/Notepad	Notepad Clipboard Data found
Generated	64 bit Windows 7 w/MS Word	Microsoft Word Clipboard Data found
Generated	64 bit Windows 7 w/MS Excel	Microsoft Excel Clipboard Data found

all cases, the tool recovers the text information in the clipboard. In addition to the text information, the tool identifies several other formats, including OLE private objects and idataobjects. For instance, for Microsoft Excel, the data exists in a total of eleven formats in the clipboard. Although not part of this experiment, parsing these additional formats would have allowed the extraction of additional formatting information.

5. Conclusions and future work

The methodology successfully retrieved the text from several different applications, which included Notepad, Microsoft Word, and Microsoft Excel. There is no reason to believe the methodology would be any less successful at retrieving clipboard text stored from any other application. In addition, the code worked unchanged for Windows XP, Vista, and 7 with only modifications for 32 and 64 bit. This suggests that the underlying clipboard structures have not changed for a long time.

Unfortunately, there are many undocumented structures including *dataobjects* and *OLE Private Objects* and while the methodology worked great for text, it was unable to retrieve information from these data objects. Specifically, when the methodology was applied to copying files to the clipboard, the methodology was unable to determine the file name. Further work is required to determine the format of these structures.

Much of the work in memory forensics to date has focused on kernel structures. While these structures are a prerequisite to any further analysis, they are only a first step. Forensic analysis of user-level structures is a critical next step in memory forensics. Fig. 3 provides a process for reversing Windows drivers and dynamic link libraries to extract the structures needed for analysis of user-level data. Further work is required to develop and generalize this methodology.

Acknowledgements

This work is supported by the Air Force Research Lab/Information Grid Division (AFRL/RIG) and the Air Force Office of Scientific Research/Dynamics and Control (AFOSR/RSL) project #2311/FX. The views expressed in this paper are those of the authors and do not represent the views or policies of AFRL/RIG, AFOSR/RSL, the United States Air Force, or the Department of Defense.

REFERENCES

Allan R. History of the Personal computer: the People and the Technology. Allan Publishing; 2001.

Betz C. memparser, http://sourceforge.net/projects/mem-parser; 2005.

Carvey H. Windows forensic analysis. Syngress Publishing; 2007.

Digital Forensics Research Workshop. 2008 forensics Rodeo, http://www.dfrws.org/2008/rodeo.shtml; 2008 [accessed 19.02.11].

Dodge D, Mullins B, Peterson G, Okolica Simulating J. Windows-Based cyber Attacks using Live virtual machine Introspection Summer computer Simulation Conference (SCS10); 2010. 550–555.

Dolan-Gavitt B. Forensic analysis of the Windows registry in memory, Proceedings of the 2008 digital forensic research Workshop (DFRWS); 2008. 26–32.

Microsoft.com How to add data to the clipboard, http://www.microsoft.com/windowsxp/using/setup/tips/clip-book.mspx, [accessed 8.02.11].

Microsoft.com OLE Background, http://msdn.microsoft.com/en-us/library/aa271002(v=VS.60).aspx, [accessed 14.04.11].

National Institute of Standards and Technology The CFReDS project. http://www.cfreds.nist.gov, [accessed 19.02.11].

netez.com Data Objects and the clipboard, http://netez.com/2xExplorer/shellFAQ/adv_clip.html, [accessed 8.02.11].

NIJ. Electronic Crime Scene investigation: an On-the-Scene Reference for first Responders. US Department of Justice; 2009.

Okolica J, Peterson G. Windows operating systems Agnostic memory analysis. In: Proceedings of the 2010 digital forensic research Workshop (DFRWS); 2010. p. 48–56.

Petzold C. programming Windows: Fifth Edition. Microsoft Press; 1999.

Prosise C, Mandia K, Pepe M. Incident response & computer forensics. 2 ed. McGraw-Hill/Osborne; 2003.

Schuster A. Searching for processes and Threads in Microsoft Windows memory dumps. In: Proceedings of the 2006 digital forensic research Workshop (DFRWS); 2006. p. 10–6.

Schuster A. PTfinder, http://computer.forensikblog.de/en/2006/03/ptfinder_0_2_00.html; March 2, 2006.

Stevens R, Casey E. Extracting Windows command line Details from Physical memory. In: Proceedings of the 2010 digital forensic research Workshop (DFRWS); 2010. p. 57–63.

The Volatility Framework: Volatile memory artifact extraction utility framework www.volatilesystems.com/default/volatilty, [accessed 15.04.11].

Walters A, Petroni N. Volatools: Integrating volatile memory forensics into the digital investigation process. Blackhat Hat DC, www.blackhat.com/presentations/bh-dc./bh-dc-07-Walters-WP.pdf; 2007.