available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

Extracting Windows command line details from physical memory

Richard M. Stevens [a,*], *Eoghan Casey* [b]

[a] *Information Security Institute, Johns Hopkins University, Baltimore, MD 21218, USA*
[b] *cmdLabs, Baltimore, MD 21218, USA*

ABSTRACT

Current memory forensic tools concentrate mainly on system-related information like processes and sockets. There is a need for more memory forensic techniques to extract user-entered data retained in various Microsoft Windows applications such as the Windows command prompt. The command history is a prime source of evidence in many intrusions and other computer crimes, revealing important details about an offender's activities on the subject system. This paper dissects the data structures of the command prompt history and gives forensic practitioners a tool for reconstructing the Windows command history from a Windows XP memory capture. At the same time, this paper demonstrates a methodology that can be generalized to extract user-entered data on other versions of Windows.

1. Introduction

The Microsoft Windows command prompt (cmd.exe) is often used by perpetrators of computer crime, and being able to reconstruct what instructions were executed on the command line can be important in a digital investigation. Some computer intruders go so far as to place their own copy of the command prompt executable on a compromised system to facilitate their unauthorized activities. The command history maintained by the Windows command prompt can contain valuable information such as what programs were executed with associated arguments, files and folders that were accessed, and unique information such as IP addresses, domain names and network shares (Aculina et al., 2008). If the command history can be reconstructed during an investigation it can provide significant context into how and what occurred on that system (Carvey, 2008). In some cases the command history might contain the only retrievable traces of a deleted file or suspect activity.

The Windows XP command prompt provides command history functionality through the *DOSKEY* command. Unlike UNIX systems that maintain a command history file, the Windows XP command history is normally only accessible while the command window is still open. However traces of these commands can be found in memory and some information can be extracted by interpreting the associated data structures in a memory capture. This paper details how *DOSKEY* stores commands in memory and provides forensic practitioners with a valuable tool that can provide significant insight and context into what actions were performed on a target computer. In addition, this paper demonstrates a methodology that can be generalized to extract user-entered data from a memory capture from newer versions of Windows.

2. Background

Recent research and development in Windows memory forensics has provided the forensic practitioner with several

* *Corresponding author*.
E-mail addresses: rsteve23@jhu.edu (R.M. Stevens), eoghan@disclosedigital.com (E. Casey).
1742-2876/$ – see front matter © 2010 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2010.05.008

valuable tools for investigating a computer crime or intrusion. Memory forensics offers several advantages over a traditional live forensics approach because it minimizes the impact on the target host, the analysis process may be repeated and verified, and the memory capture may be reanalyzed as new techniques and approaches are discovered (Walters and Petroni, 2007). In order to demonstrate the value of memory forensics as an alternative to traditional incident response techniques it is necessary to duplicate or exceed the functionality of the standard incident response commands.

Recent work has made significant progress in providing access to much of this information, including running processes and threads, active network connections, file handles, and open DLL files (Walters and Petroni, 2007) (Betz, 2005). In addition to finding active system objects these approaches have proven successful in identifying system objects that are associated with closed or disconnected processes, files and connections (Schuster, 2006). The ability to identify volatile information that is no longer accessible through the normal operation of the system is of substantial value.

There is a need for more memory forensic techniques to extract user-entered data retained in various Microsoft Windows applications such as the Windows command prompt. Deallocated memory pages can remain in memory for some time and may be a source of forensically useful information even after a program has been closed (Solomon et al., 2007). As a result, memory pages that contain command history information may be recoverable even after the Windows command prompt has been closed.

One approach to finding remnants of the command line history in memory captures is to search for known commands. In practice however there may be no known commands that can be used as a search term and it can be difficult to distinguish the command history from the contents of a file or of a display buffer. This approach also assumes that sequential commands are in the order they are entered, are not from several distinct command sessions, and are commands rather than unrelated command-like strings like a segment of a man page. A more useful approach to reconstructing command line history is to search the memory capture for the signature of the data structures used to store the command history.

Dolan-Gavitt took this approach to identify Registry files stored in memory. This work was able to reconstruct a significant number of Registry keys that might not otherwise be recoverable (Dolan-Gavitt, 2008). Dolan-Gavitt's approach first identifies a Windows Registry hive by searching memory for a known header value. Once a Registry hive has been identified it is possible to identify the *HiveList* and resolve the remainder of the Registry hive objects. Kornblum's work also shows that the structures surrounding useful objects such as an encryption key can be used to identify object in memory (Kornblum, 2009). Both of these approaches demonstrate that a detailed knowledge of how data are stored in memory by an application is necessary to facilitate its recovery.

3. Windows command line history

Microsoft operating systems originally provided command history functionality through the DOSKEY application. DOSKEY has since been incorporated into the Windows command prompt as the *DOSKEY* command and allows the user to edit commands and access past commands. While using a Window command prompt the user can access past commands using the UP and DOWN arrows to cycle backwards and forwards through a list of saved commands. The entire command history can also be displayed by typing the command DOSKEY/history and the user can press the F7 key within the command prompt to open the DOSKEY command history window as shown in Fig. 1.

By default DOSKEY displays at most the last 50 commands entered in the command prompt. This value can be configured by the user from a minimum buffer size of one command to a maximum buffer size of 999 commands. One interesting feature of the DOSKEY program is that it collapses sequential commands that are identical into a single DOSKEY history element. If the same command is entered several times in a row it will only be displayed once in the command history. DOSKEY also does not store empty commands.

The command history buffer size can be modified by accessing the properties of the open command window. This variable can also be modified for all future command windows by editing the "HKEY_CURRENT_USER\Console\HistoryBufferSize" Registry key. DOSKEY also allows users to delete the contents of the command history by using the DOSKEY/reinstall command, and contains functionality that allows the creation and execution of DOS command macros (Microsoft, 2007).

The DOSKEY command history is stored entirely in memory and is not written to the physical disk. The command history is only accessible when the command prompt is open and is no longer accessible when the command prompt is closed. In practice this makes recovering the command history difficult due to its volatile nature and the low likelihood of finding an open command window during an investigation (Carvey, 2008). These commands however may be recoverable from memory for some time after the window has been closed. In order to extract and reconstruct the DOSKEY command history, it is necessary to examine how DOSKEY stores commands in memory and then attempt to identify a signature or method in which they might be identified from a memory capture.

4. Data structures

The only available information on how Windows command line information is stored can be found in functional

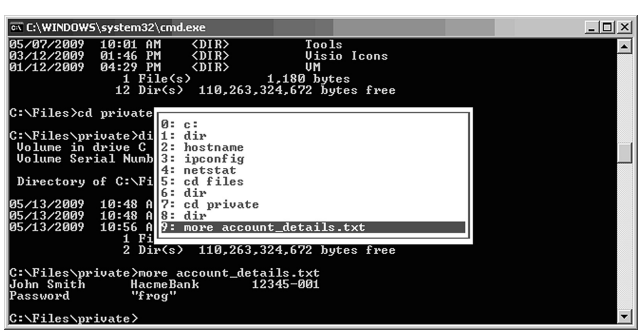

Fig. 1 – DOSKEY F7 History Option.

```
Offset

004F2F38        26 00 65 00 63 00 68 00
                38      &.e.c.h.
                39
004F2F40        6F 00 20 00 55 00 6E 00
                69 00 71 00 75 00 65 00
                o. .U.n.i.q.u.e.

004F2F50        43 00 6F 00 6D 00 6D 00
                61 00 6E 00 64 00 31 00
                C.o.m.m.a.n.d.1.
```

Fig. 2 – A Single Command Element.

explanations of the DOSKEY command on Microsoft TechNet (Microsoft, 2007). In order to understand more fully the functionality of the DOSKEY application and how it stores the command history in memory it was necessary to conduct a number of live experiments. Initial experiments were performed on Windows XP Professional SP3 on VMware Server 1.0.3 configured with 512 MB of RAM. Windows XP was initially used to facilitate validation using publically available memory captures as discussed in the Evaluation section of this paper.

Preliminary experiments revealed various remnants of commands in memory, primarily in Unicode. The majority of these remnants were from the display buffer of the command prompt window, or from messages generated when each command was executed. Each command did have a single string in memory that was associated with the DOSKEY command history. Through an analysis of these results the data structures for the DOSKEY command history element, the command history buffer and several possible methods for their identification were found. For example the command "echo UniqueCommand1" as shown in Fig. 2 is 19 characters long which requires 38 bytes to store as a Unicode string. The short integer value stored at 0x4F2F30 that precedes the command string contains the hexadecimal value of 0x0026 which is equivalent to 38 decimal.

Interestingly, these command elements were not found within the addressable memory of the command prompt executable but rather in the memory space of the Windows XP user run-time environment process (csrss.exe). This process, which is called the "Client/Server Run-Time Subsystem," handles console or character based executables such as the command prompt executable cmd.exe (Russinovich and Solomon, 2005).

Multiple experiments confirmed that the command history element was structured as indicated in Fig. 3.

This data structure is useful for interpreting command history element that has already been located in memory, but is not sufficiently unique to be used as a signature. Fortunately a more distinctive memory structure was found that contains

```
commandElement {

        0x00 short ByteCount;
        // Short, Little-Endian

        0x02 char Command [ByteCount/2];
        // UTF-16

}
```

Fig. 3 – Command Element Structure.

```
commandHistory {
        0x00    short       ElementCount;
        0x02    short       endOffset;
        0x04    short       pointerIndex;
        0x06    short       startOffset;
        0x08    short       HistoryBufferSize;
                …
        0x16    commandHistory*     ?
        0x20    commandHistory*     ?
        0x24    commandElement1*;
        0x28    commandElement2*;
        …
        0x n    commandElement_HistoryBufferSize*;
}
```

Fig. 4 – Command History Structure.

a list of memory addresses, each of which points to a command history element. These memory addresses were found to be stored in sequential order in memory. Through additional experimentation it was determined that the command line history list was structured as indicated by Fig. 4.

The most important element in this structure is the *HistoryBufferSize*. This value has a known default value of 50d or 0x0032 which provides an excellent indicator for a possible signature. The possible values for this field are between 0 and 999, or 0x0000 and 0x03E7. The *ElementCount* stores the number of commands stored in the DOSKEY *CommandHistory* structure. This value must be between 0 and the value stored in the *HistoryBufferSize*.

The history array structure is fixed and has a maximum value of *HistoryBufferSize*. When a user enters more commands than the buffer can hold it overwrites the earliest stored entry with the most recent command. DOSKEY uses *startOffset* and *endOffset* to keep track of where the earliest and most recent commands are stored within the array of *commandHistory* elements. DOSKEY also maintains an index variable *pointerIndex* which stores the index of the *commandElement* currently selected by the user if they have cycled within the list using the UP or DOWN keys. In practice, the

```
DOSKEY Signature {
        0x00    short (between 0 and HistoryBufferSize)
        0x02    short (between -1 and HistoryBufferSize)
        0x04    short (between -1 and HistoryBufferSize)
        0x06    short (between 0 and HistoryBufferSize)
        0x08    short (default value of 0x32)
        0x16    commandHistory* (Valid Address)
        0x20    commandHistory* (Valid Address)
}
```

Fig. 5 – DOSKEY Signature.

Table 1 – Command history variables.

Element Count	End offset	Pointer index	Start offset	History buffer size
4	3	3	0	50

pointerIndex is typically 0. The *startOffset*, *endOffset* and *pointerIndex* are also within the range of 0 and the value stored by the *HistoryBufferSize*.

Of interest are two memory addresses located at offset 0x16 and 0x20 that contain the virtual address of the *commandHistory* object. This study was unable to determine the function of these values. However, research suggests that DOSKEY is capable of storing multiple history buffers, and these fields may be part of a double-linked list to additional *commandHistory* objects. These values may also be related to the DOSKEY commands macro functionality. In practice however these values were found to always indicate the address of the containing *commandHistory* object. The purpose of the values stored between 0x12 and 0x16 are also unknown. Additional study is required to determine the purpose of these fields.

Based on these results, the following signature was constructed that identifies the DOSKEY command history structure in memory (Fig. 5).

This signature is distinctive enough to locate any possible *commandHistory* objects in a memory capture with no prior knowledge of any possible commands. Once a *commandHistory* object is located, each *commandElement* within the object can be examined. Performing a brute force signature search across a memory capture is effective but time-consuming. Manual analysis to convert the physical to the virtual addresses and rebuild the command structure was relatively straight-forward but time-consuming. This process can be greatly aided by a tool that automatically maps out possible command history structures and their contents.

5. Implementation

The DOSKEY history signature was originally examined through a Perl script that parsed a memory capture for possible DOSKEY signatures and output the results for relatively intensive, manual examination. This process proved to be useful but did not scale well and was time-consuming. To improve the process, a plug-in named *cmd_history.py* was written for the Volatility framework (Schuster, 2009); this plug-in both identified possible DOSKEY signatures and reconstructed the command history array that proved to be significantly faster and more resilient. The *cmdHistoryScanner* module in *cmd_history.py* was inspired by the *cryptoscan* module written by Jesse Kornblum for quickly searching for known object signatures (Kornblum, 2008).

Table 2 – Command History Elements.

Virtual Address	Physical address	Size (bytes)	Command
004E8E88	149cfE88	8	cd \
01283A20	14fbbA20	18	mkdir mem
01283B48	14fbbB48	12	cd mem
004E1FF8	80d6FF8	138	"Z:\emidnight On My Mac \Downloads\mdd_1.3.exe" -o sv-laptop-memo
Add. Command			
012839C0	14fbb9C0	88	(vxfer.exe X:\Secretplans \secretplans1.jpg
01283AE8	14fbbAE8	–	.....exe X:\Secretplans \secretplans.....
01283B48	14fbbB48	12	cd mem
01283BA8	14fbbba8	84	svxfer.exe X:\Secretplans \secretplans7.jpg
004E1FA0	80d6FF8	?	?

Table 3 – Command History Variables.

Element Count	End Offset	Pointer index	Start offset	History buffer size
20	13	13	0	50

6. Evaluation

The signature and extraction method detailed in the previous section were validated using memory dumps made publically available by the Digital Forensics Research Workshop (DFRWS) and the National Institute of Standards and Testing (NIST).

6.1. DFRWS 2008 Rodeo

The Digital Forensics Research Workshop (DFRWS), 2008 Forensics Rodeo created a scenario where a trusted insider accessed confidential information without authorization. As part of the scenario a memory capture of the suspect laptop is provided. The signature-based approach identified a DOSKEY

Table 4 – Command history elements.

Virtual Address	Physical address	Size (bytes)	Command
004E1F90	de7fF90	4	dd
004E2CB8	193ecCB8	6	cd\
004E2D18	193ecD18	4	dr
004E2D28	193ecD28	6	ee:
004E2D38	193ecD38	4	e;
004E2D48	193ecD48	4	e:
004E2D58	193ecD58	4	dr
004E2D68	193ecD68	4	d;
004E2D78	193ecD78	4	d:
004E2D88	193ecD88	4	dr
004E2D98	193ecD98	4	ls
004E2Da8	193ecDa8	14	cd Docu
004E2DC0	193ecDC0	68	cd Documents and.....
004E2E58	193ecE58	4	dr
004E2E68	193ecE68	4	d:
004E2E78	193ecE78	12	cd dd\
004E2E90	193ecE90	34	cd UnicodeRelease
004E2Ec0	193ecEc0	4	dr
004E2ED0	193ecED0	6	dd
004E4100	19 588 100	132	dd if = \\.\PhysicalMemory of = c: \xp-2005-07-04-1430.img conv = noerror

Table 5 – Command history variables.

Element Count	End Offset	Pointer index	Start offset	History buffer size
7	6	6	0	50

command history object at offset 0x152F9DB8 within the memory capture. The command history object indicated that there are four commands in the array (Table 1). On examination the array structure appeared to contain ten command element pointers, several of which resolved to valid command element objects as shown in Table 2. Several of the recovered command elements contain partial commands or are missing the *ByteCount* variable.

The additional command elements indicate that the slack space within the command history buffer could be a valuable source of digital evidence. An examination of these additional commands led to the identification of several commands pertinent to the scenario. Further examination of the slack space at the end of each command element revealed several indications that earlier command elements were overwritten.

6.2. *NIST reference data Set*

The DOSKEY history signature was also tested on Windows XP Memory images from the NIST Computer Forensics Reference Data Set project. These datasets are provided as reference data with documented contents that can be used to test tools and methodologies (NIST). Two memory images are provided for analysis: xp-laptop-2005-07-04.img and xp-laptop-2005-06-25.img. Both images were created using a Toshiba laptop running Windows XP.

xp-laptop-2005-07-04.img: The signature-based approach identified a DOSKEY command history object at offset 0x19588D98. The command history object indicated that there are twenty commands in the array (Table 3). Each of the command elements appeared to be whole and intact (Table 4).

xp-laptop-2005-06-25.img: The signature-based approach identified a DOSKEY command history object at offset 0x14408DA8. The command history object indicated that there are seven commands in the array tabulated below (Table 5). The seventh command entry was only partially recovered. The *ByteCount* variable however suggests that this command is the *dd* command used to take the memory sample (Table 6). While testing the initial signature it was found that the command initiating the memory sample was often only partially recoverable.

The final command element is an excellent example of the value of the ByteCount variable. In several instances throughout this study the ByteCount value remained intact but was followed by a partial command. It may be possible to infer the value of the command history element from the length stored in the ByteCount field and the fragment of the history item. In this case the ByteCount variable indicates the command is 44 characters long and started with "dd" – it is a reasonable assumption that this command relates to the dd command used to acquire the memory capture.

Table 6 – Command History Variables.

Virtual address	Physical address	Size (bytes)	Command
004E2D28	14400D28	4	d:
004E1F78	dcbfF78	12	cd dd
004E2CC8	14400CC8	6	dir
004E2E00	14400E00	34	Cd UnicodeRelease
004E2CB8	14400CB8	6	dir/
004E1F90	dcbfF90	6	dd
004E1FF8	dcbfFF8	88	dd ….. (presumably dd memory image command)

7. Interesting behaviors

During the research a number of scenarios were examined to determine how the DOSKEY command stored useful information and how that information might be recovered. Several of the scenarios proved to be quite interesting and significantly improved the studies understanding of the DOSKEY command.

A test scenario was created that tested the effect of the DOSKEY/reinstall command. This command "installs a new copy of doskey" and effectively clears the current DOSKEY command history buffer (Hill, 1998). The test environments utilized a number of known commands, executed the DOSKEY/reinstall command and then examined the resulting memory capture. In this case the signature failed to identity a valid DOSKEY command history object within memory.

A manual search for commands within the test command group failed to identify any intact command history elements for any of the search terms. A test virtual machine was built to examine this case. It was determined that when the reinstall command was performed, the DOSKEY command history values below are reinitialized to Fig. 6.

This places the *endOffset* and *pointerIndex* values outside of the values specified within the initial search parameters. Unfortunately this command also appears to reinitialize the pointers within the *commandHistory* array and the command history elements within the array.

In subsequent tests a semi-intact command history and command history elements were successfully identified after the reinstall command was performed. In these cases it appears that portions of memory were reallocated so that orphaned fragments of the original objects were left intact. On examination these values appear to be the default values created when

```
commandHistory {
    0x00    short    ElementCount = 0x0000;
    0x02    short    endOffset = 0xFFFF;
    0x04    short    pointerIndex = 0xFFFF;
    0x06    short    startOffset = 0x0000;
    0x08    short    HistoryBufferSize;
                     (Default 0x32)
```

Fig. 6 – Default Command History Signature.

a new command window is opened, so the presence of a "blank" DOSKEY command history buffer is not necessarily indicative of the reinstall command being performed.

It was also noted that when multiple command prompts were opened, an independent DOSKEY command history buffer object was created for each command prompt. This was verified in a test scenario by opening two command prompts, entering unique commands into each command prompt, taking a memory capture and analyzing the resulting memory dump. The signature correctly identified two command buffer history structures which contained the command history of each command prompt respectively. In cases where the contents of multiple command prompt sessions can be found in memory, an analysis of the buffer would allow the commands for each session to be identified and separated.

Several test environments were also created to test the recovery of DOSKEY command history buffer structures after a command window had been closed. This scenario involved opening two command prompts and entering unique commands into each command prompt. A baseline memory capture was then taken. One of the two command prompts was then closed, followed by a second memory capture shortly afterwards. This allowed for the comparison of allocated memory both before and after the command prompt had been closed and tested the recovery of a command history buffer in an ideal situation. In the initial tests of this scenario the entirety of the command history buffer was successfully recovered along with the majority of the associated command elements. In several cases the signature search was unable to find a valid command history structure because the structure had partially overwritten in memory. In these cases a manual analysis revealed partial fragments of the original DOSKEY history structures but was unable to recover the entire command history.

8. Conclusions

Data structures such as those used by the DOSKEY command provide an excellent example of how memory forensics can provide forensic practitioners with valuable insight into what occurred on a compromised or suspect system.

The presence of a preceding short value equal to the length of the following string in bytes could indicate that the string is a DOSKEY command. While this signature is useful in verifying that a suspect DOS command is a command history element it is not sufficiently distinct to be easily identified in a memory capture. It might be possible to combine this signature with a frequency analysis of DOS commands (such as looking for a greater than normal incidence of the ":" and "\" characters) but it would be difficult to distinguish commands from surrounding Unicode values.

The DOSKEY command stores the command history in a relatively simple data structure that exhibits sufficiently distinct characteristics to be readily identified within a memory capture. This study has shown that using this data structure, it is possible to recover the command history from a memory capture using this signature. The results of this study also demonstrated that the DOSKEY command buffer can contain slack space that often contains commands from earlier command windows or after the DOSKEY history has been cleared using the DOSKEY/reinstall command.

Both the *commandHistory* and *commandElement* objects contain metadata that could prove to be of forensic value, even when it is not possible to recover the entire command history structure. Metadata within these data structures could allow the forensic examiner to determine the number of commands typed, determine the presence of missing commands, determine the order in which commands were entered and in some cases suggest the contents of a partially recovered command.

The top–down approach of first identifying the command history buffer object proved to be highly effective when the command history structure was intact in memory. In practice however this structure may be partially overwritten when the command window is closed. Future approaches based on the signature of individual element should incorporate a level of tolerance to identify partial structures where one or more fields have been overwritten. It is still possible to use a bottom–up approach in these cases if an examiner is able to successfully identify a command element object.

The success of the DOSKEY signature is based on the assumption that the examiner can determine the history buffer size. In most cases this will be the default value and in many cases the actual value can be determined by examining the systems registry. Reducing the dependence on the history buffer size might prove to be useful but could lead to a higher incidence of false positives. Identifying the purpose of the memory addresses stored at offsets 0x16 and 0x20 may also further refine the effectiveness of this signature.

Future work in this area should find substantial value in examining similar data structures in other operating systems, and in particular Windows Vista, Windows 7 and the PowerShell command prompt replacement. As this work expands it may be possible to implement a tool capable of carving a wide variety of such objects from memory. The process of extracting user-entered data from memory captures would be aided by the development of a unified signature database containing details about various data structures in memory.

REFERENCES

Aqulina JM, Casey E, Malin CH. Malware forensics: investigating and analyzing Malicious Code. Burlington, MA, USA: Syngress Publishing; 2008. pp 60.

Betz C. DFRWS 2005 Challenge Report, Digital Forensic Research Workshop 2005 Memory Analysis Challenge; 2005.

Carvey H. Windows forensics analysis: incident response and Cybercrime investigation Secrets. Burlington, MA, USA: Syngress Publishing; 2008. pp. 39.

Digital Forensics Research Workshop. 2008 forensics Rodeo. accessed from, http://dfrws.org/2008/rodeo.shtml; 2008.

Dolan-Gavitt B. Forensic analysis of the windows registry in memory. Digital Investigation 2008;5(Suppl. 1). The Proceedings of the Eighth Annual DFRWS Conference.

Hill T. The windows NT command Shell, windows NT Shell Scripting. accessed from. MacMillan Technical Publishing, http://technet.microsoft.com/en-us/library/cc750982.aspx#XSLTsection126121120120; 1998.

Kornblum J. CryptoScanner TrueCrypt Volatility plugin. accessed from, http://jessekornblum.com/tools/volatility/cryptoscan.py; 2008.

Kornblum J. Practical Methods for Dealing with Full Disk Encryption, Presented at Department of Defense Cyber Crime Conference; 2009.
NIST (n.d.). Computer forensic reference data Sets, memory images. accessed from, http://www.cfreds.nist.gov/mem/Basic_Memory_Images.html.
Russinovich MR, Solomon DA. Microsoft windows Internals, Microsoft windows Server 2003, windows XP, and windows 2000. 4th ed. Redmond, WA, USA: Microsoft Press; 2005. pp 53.
Schuster A. Searching for processes and threads in Microsoft windows memory dumps. In: Proceedings of the 2006 digital forensics research Workshop (DFRWS); 2006.
Schuster A. Windows Memory Forensics with Volatility (course slides), presented at FIRST 2009. accessed, http://computer.forensikblog.de/files/talks/FIRST2009-Windows_Memory_Forensics_with_Volatility.zip; 2009.
Solomon J, Heubner E, Bem D, Szeynska M. User data persistence in physical memory. Digital Investigation June 2007;4(2).
Walters A, Petroni Jr NL. Volatools, integrating volatile memory forensics into the digital investigation process. Black Hat DC; 2007.

Eoghan Casey is an Incident Response and Digital Forensic Analyst, responding to security breaches and analyzing digital evidence in a wide range of investigations, including network intrusions with international scope. He has extensive experience using digital forensics in response to security breaches to determine the origin, nature and extent of computer intrusions, and has utilized forensic and security techniques to secure compromised networks. He teaches at the Johns Hopkins University Information Security Institute and is the author of the widely used text book Digital Evidence and Computer Crime now in its second edition, is editor of the Handbook of Computer Crime Investigation. He is also the editor-in-chief of Elsevier's Digital Investigation journal.

Richard Stevens is a Senior Systems Security Analyst for T. Rowe Price, where he is responsible for digital forensics, incident response and malware analysis. Richard holds a Bachelor of Computing (Honours) from the University of Tasmania, a Post-graduate Diploma in Information Security & Intelligence from Edith Cowan University and a Master of Science in Security Informatics from Johns Hopkins University. Richard's research is primarily focused on how memory forensics and malware analysis can be applied within the enterprise.