Digital Investigation xxx (2016) 1–7

Contents lists available at ScienceDirect

Digital Investigation

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

Modern windows hibernation file analysis

Joe T. Sylve [a, b, *], Vico Marziale [a], Golden G. Richard III [b]

[a] Blackbag Technologies, Inc, San Jose, CA, USA
[b] Department of Computer Science, University of New Orleans, New Orleans, LA, USA

ARTICLE INFO

Article history:
Received 1 September 2016
Received in revised form
22 October 2016
Accepted 16 December 2016
Available online xxx

Keywords:
Microsoft windows
Memory analysis
Memory forensics
Hibernation file
Digital forensics

ABSTRACT

This paper presents the first analysis of the new hibernation file format that is used in Windows versions 8, 8.1, and 10. We also discuss several changes in the hibernation and shutdown behavior of Windows that will have a direct impact on digital forensic practitioners who use hibernation files as sources of evidence.

Introduction

Starting with Windows 2000, Microsoft introduced a hibernation facility that allows a system to be powered down, while still preserving its volatile state. This is accomplished by saving the contents of RAM and the processor context to disk in a file called *hiberfil.sys* prior to shutdown. When the computer is again powered on the volatile state is restored and the system continues from the saved state.

Hibernation files are a good source of information for digital forensic practitioners, because they store ephemeral data from the contents of RAM to non-volatile storage without the need to run specialized tools on the target device.

Memory analysis frameworks like Volatility (2007–2016) and Rekall (2013–2016) make it easy to analyze hibernation files in much the same way as you would a raw memory dump; however, these tools are not compatible with hibernation files from the most recent versions of Windows. This is because, while the original hibernation file format is well understood, Microsoft changed the hibernation format with the release of Windows 8.

In this paper we will provide an overview of the legacy hibernation file format as well as present the first analysis of the new format that is used in Windows 8, 8.1, and 10. We have implemented support for the new format in our experimental memory analysis framework. We also discuss several changes in the hibernation and shutdown behavior of Windows that will have a direct impact on digital forensic practitioners who use hibernation files as sources of evidence. It is our hope that with this knowledge, existing memory analysis tools can be instrumented to support the new hibernation file format.

Related work

The Windows XP hibernation file format was first publicly documented by Nicolas Ruff and Matthieu Suiche in their PacSec 2007 presentation (Ruff and Suiche, 2007). They were also the first to note the that when hibernation is resumed only the hibernation file header is zeroed, retaining the hibernation data until the next hibernation event.

In 2009 Brendan Dolan-Gavitt introduced support for the hibernation file into the Volatility memory analysis framework in Dolan-Gavitt (2009).

Microsoft announced the release of Windows 8 in 2012 (Microsoft, 2012). This release changes the format of the hibernation file, breaking all existing analysis tools.

During the course of our research effort, Matthieu Suiche was simultaneously studying the format of modern windows hibernation files. In May 2016 Suiche sent an email to the Volatility Users mailing list, announcing a beta version of Hibr2Bin which supports

* Corresponding author.
E-mail addresses: joe.sylve@gmail.com (J.T. Sylve), vicodark@gmail.com (V. Marziale), golden@cs.uno.edu (G.G. Richard).

http://dx.doi.org/10.1016/j.diin.2016.12.003
1742-2876/

Windows 8, 8.1, and 10 hibernation files (Suiche, 2016a). Hibr2Bin is a tool that converts Windows hibernation files to raw memory images so that they can be analyzed by memory analysis tools that do not natively support parsing hibernation files. The updated Hibr2Bin was released publicly in late September 2016 (Suiche, 2016b).

Hibr2Bin does not allow the direct analysis of hibernation files and Suiche has not publicly released any description of the new format. We have also found that Hibr2Bin fails to properly handle hibernation files from the latest versions of Windows for reasons we explore in Section Validation against Hibr2Bin.

Windows XP-7 hibernation format

Windows hibernation files provide non-volatile storage of the system's processor state and physical memory in order to resume a powered-off system to its previous state. While this feature has been available in some capacity since Windows 2000, there are currently two variations of hibernation files that are commonly encountered by investigators today.

The first variation we will discuss is the well-known format used by Windows XP, Vista and 7 (hereafter referred to as *Windows XP-7*).

Windows hibernation files can be found at the root of the system drive in a system-protected file called *hiberfil.sys*. An overview of the layout of *hiberfil.sys* on Windows XP-7 can be found in Fig. 1. Each of the relevant structures will be discussed in this section.

Windows XP-7 hibernation files generally exist in one of two states: *hibernated* or *restored*. A file is considered in the hibernated state when the hibernation process is completed and the system has powered off for the first time. When the system is powered back on, the contents of the hibernation file are restored to memory and the stored processor context is loaded. The first 4096 bytes of the hibernation file are zeroed and the file is then considered in the restored state. The file will not be modified again until the next time a system hibernation is performed.

File header

Hibernation files in the hibernated state begin with a `PO_MEMORY_IMAGE` header. While the exact structure of the header varies slightly among OS versions, it is defined publicly in the kernel's debugging symbols. By using the `dt` command of the Microsoft Kernel Debugger tool, WinDbg (Microsoft, 2016a), we can learn the exact structure of the header used in a given version of Windows. Fig. 2 shows the definition of the header from the 64-bit version of Windows 7 SP1.

Hibernated *hyberfil.sys* files contain the ASCII value of `hibr` or `HIBR` in the `Signature` field. While the system is in the process of resuming from hibernation, this field is changed to the value of `rstr` or `RSTR`. Once the system successfully resumes from hibernation, the header is lost when the first 4096 bytes are zeroed.

Processor context

In order to resume execution after hibernation, *hiberfil.sys* contains a stored copy of all the processor register values. This context is stored in an OS-version-specific `_KPROCESSOR_STATE` structure, whose definition can be discovered with WinDbg. The offset of this structure inside of hiberfil.sys also depends on the OS version. The offsets for all the relevant versions can be found in Table 1.

Knowledge of the processor context can be valuable during the memory analysis process. For example, during hibernation the `CR3` register encodes the physical address of the system's page tables, and on 32-bit systems the `CR4` register contains a flag which tells whether or not page address extensions (PAE) are enabled. These values can be discovered by reading the `SpecialRegisters.Cr3` and `SpecialRegisters.Cr4` members of the `_KPROCESSOR_STATE` respectively. Both of these properties must be learned in order to successfully perform address translation during analysis.

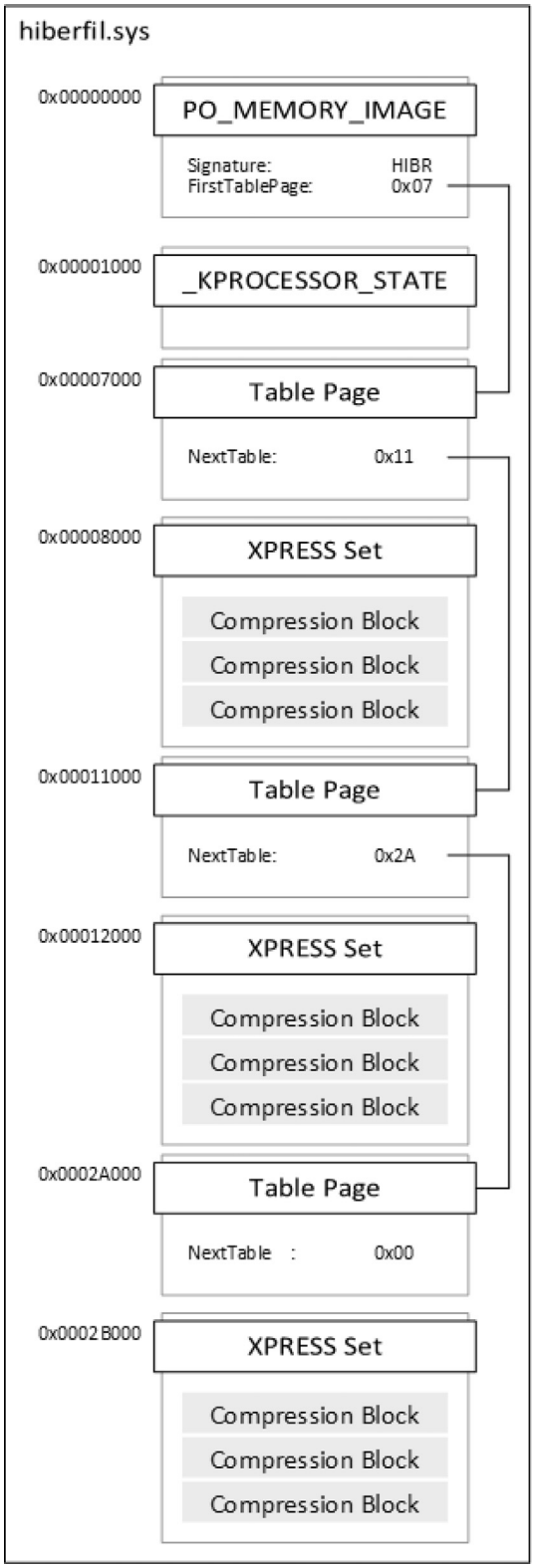

Fig. 1. Example Windows 7 *hiberfil.sys* layout.

```
kd> dt PO_MEMORY_IMAGE
nt!PO_MEMORY_IMAGE
   +0x000 Signature        : Uint4B
   +0x004 ImageType        : Uint4B
   +0x008 CheckSum         : Uint4B
   +0x00c LengthSelf       : Uint4B
   +0x010 PageSelf         : Uint8B
   +0x018 PageSize         : Uint4B
   +0x020 SystemTime       : _LARGE_INTEGER
   +0x028 InterruptTime    : Uint8B
   +0x030 FeatureFlags     : Uint4B
   +0x034 HiberFlags       : UChar
   +0x035 spare            : [3] UChar
   +0x038 NoHiberPtes      : Uint4B
   +0x040 HiberVa          : Uint8B
   +0x048 HiberPte         : _LARGE_INTEGER
   +0x050 NoFreePages      : Uint4B
   +0x054 FreeMapCheck     : Uint4B
   +0x058 WakeCheck        : Uint4B
   +0x060 FirstTablePage   : Uint8B
   +0x068 PerfInfo         : _PO_HIBER_PERF
   +0x0c0 FirmwareRuntimeInformationPages : Uint4B
   +0x0c8 FirmwareRuntimeInformation : [1] Uint8B
   +0x0d0 NoBootLoaderLogPages : Uint4B
   +0x0d8 BootLoaderLogPages : [8] Uint8B
   +0x118 NotUsed          : Uint4B
   +0x11c ResumeContextCheck : Uint4B
   +0x120 ResumeContextPages : Uint4B
```

Fig. 2. Definition of the `PO_MEMORY_IMAGE` structure (Windows 7 SP1 x64).

Table 1
Starting offsets of the stored processor context.

Windows version	File offset
XP	0x2000
Vista	0x4000
Vista SP1	0x1000
Vista SP2	0x1000
7	0x1000
7 SP1	0x1000

Physical memory

Along with processor context, physical memory pages must also be restored before resuming from hibernation. The majority of space in the *hiberfil.sys* is used to store a compressed copy of the active physical memory pages of the system.

As shown in Fig. 1, *hiberfil.sys* contains a linked list of *table pages* that are each followed by a set of compressed blocks that we will refer to as an *XPRESS set*. Together, table pages and XPRESS sets contain the meta-data and data needed to reconstruct the state of physical memory that is required to resume execution. This section will discuss both of these data types.

Table pages

Table pages contain `_PO_MEMORY_RANGE_ARRAY` structures that list, in order, which physical memory pages are associated with the data contained in the next XPRESS set. These structures are defined in Fig. 3. Each table page starts with a `_PO_MEMORY_RANGE_ARRAY_LINK` structure followed by a number of `_PO_MEMORY_RANGE_ARRAY_RANGE` structures as determined by the `EntryCount` field of the link.

Each range defines a set of physical pages that are to be restored with the data stored in the next XPRESS set. The pages defined in the first range are associated with the first pages of XPRESS data. The pages defined in the second range are associated with the next pages of XPRESS data, and so on.

Since a table page can not exceed 4 KiB in size, only a limited number of page ranges can be stored in a single array, thus the `NextTable` field of the link may contain a page number of the next table page in the hiberfil.sys. This page number, multiplied by 4096, gives the offset in bytes of the next table page. The last table page contains a `NextTable` value of zero.

XPRESS sets

A 4 KiB aligned XPRESS set of *compression blocks* follows each table page. Each compression block starts with an `IMAGE_EXPRESS_HEADER` as defined in Fig. 4. `Signature` should be the 8-byte value `\x81\x81xpress`. `UncompressedPages` encodes the number of pages (minus one) in the block. `CompressedSize` encodes the size of the block data. If `CompressedSize` is equal to the size of the number of 4 KiB pages in the block, then the data is stored uncompressed; otherwise, the data is compressed using the Plain LZ77 XPRESS algorithm as described in Microsoft (2016b). The block data immediately follows the header.

Until all pages in the ranges defined in the previous *table page* are accounted for, another compression block exists in the set immediately following the block data.

First table page

In order to analyze the physical memory pages stored in the hibernation file we must first locate the *first table page* (FTP). For hibernated *hyberfil.sys* files we can simply look up the value in the `FirstTablePage` field of the `PO_MEMORY_IMAGE` header. This field will contain a page number which can be multiplied by 4096 to calculate the offset of the FTP.

Restored hibernation files no longer contain the `PO_MEMORY_IMAGE` header, so we must apply a scanning method to locate the FTP. We start by scanning on 4 KiB boundaries from the start of the file for the XPRESS header signature of `\x81\x81xpress`. Once we locate our first signature then we know we have found our first XPRESS set. Since XPRESS sets always follow table pages, the FTP should be located 4096 bytes before this.

Windows 8+ hibernation format

The next variation of hibernation files was introduced with the release of Windows 8 in 2012. As of the time of this writing that new variation of hibernation file is still used in the latest Windows release, Windows 10 v1607 (Windows 10 Anniversary Edition). To the authors' knowledge this variation has never been publicly documented and there are no publicly available tools for analysis.

In this section we will describe for the first time how the new format differs from the legacy XP-7 format. For brevity's sake we

```
typedef struct _PO_MEMORY_RANGE_ARRAY_LINK {
        uintptr_t   NextTable;
        uint32_t    EntryCount;
} PO_MEMORY_RANGE_ARRAY_LINK;

typedef struct _PO_MEMORY_RANGE_ARRAY_RANGE {
        uintptr_t   StartPage;
        uintptr_t   EndPage;
} PO_MEMORY_RANGE_ARRAY_RANGE;
```

Fig. 3. Definition of Table Page structures.

```
typedef struct _IMAGE_EXPRESS_HEADER {
    char        Signature[8];
    uint32_t    UncompressedPages : 10;
    uint32_t    CompressedSize : 22;
    uint32_t    CheckSum;
    uint8_t     Reserved[16];
} __attribute__((packed)) IMAGE_EXPRESS_HEADER;
```

Fig. 4. IMAGE_EXPRESS_HEADER structure definition.

Please cite this article in press as: Sylve, et al., Modern windows hibernation file analysis, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.12.003

will refer to the set of all windows versions from Windows 8 through the current version as *Windows 8+*.

An overview of the Windows 8+ *hiberfil.sys* can be found in Fig. 5. Each of the relevant components will be discussed in this section.

File header

As with the prior version, the Windows 8+ hibernation file begins with a `PO_MEMORY_IMAGE` header structure. Many new fields have been added to this structure, and as before the structure slightly differs between OS versions. A partial definition of `PO_MEMORY_IMAGE` from the 64-bit version of Windows 10 v1607 is shown in Fig. 6.

There are now four valid `Signature` values: `HIBR` for the hibernated state, `RSTR` for when the system is actively being resumed from hibernation, and `WAKE` for after hibernation has been successful. Windows Embedded supports an additional value of `HORM`. This is to support a feature known as Hibernate Once/Resume Many (HORM). When enabled, HORM allows the system to always resume from the last saved hibernation file (Microsoft, 2015). Non-Embedded versions of Windows do not support HORM, thus this signature value is less commonly seen by practitioners.

Unlike the previous version, restored hibernation files retain their headers, but everything after the first 4 KiB of data is now zeroed once the system successfully resumes from hibernation. Because of this, restored hibernation files no longer contain processor contexts or physical memory. This has implications to the forensic process that we will discuss in Section Forensic implications windows 8+.

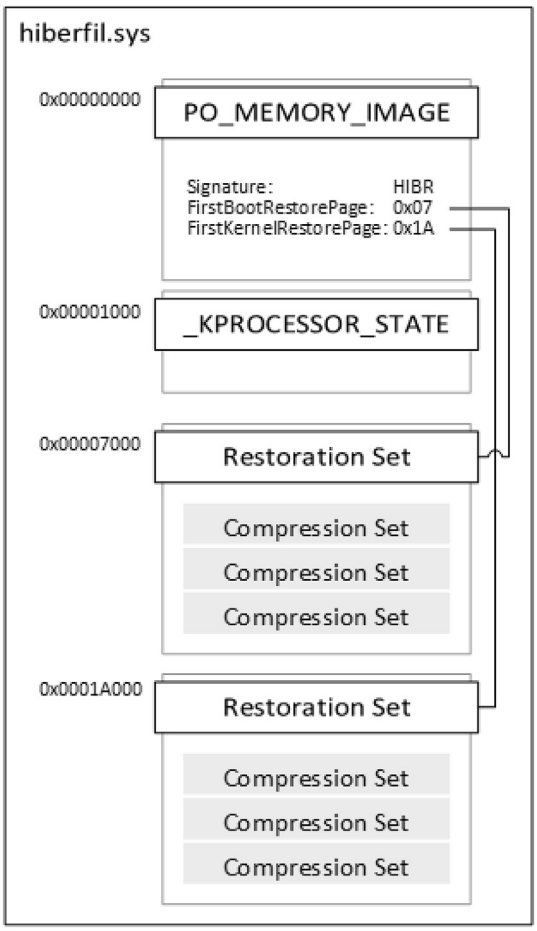

Fig. 5. Example Windows 8+ *hiberfil.sys* layout.

```
kd> dt -r1 PO_MEMORY_IMAGE
nt!PO_MEMORY_IMAGE
   +0x000 Signature        : Uint4B
   [...snip...]
   +0x020 SystemTime       : _LARGE_INTEGER
   [...snip...]
   +0x058 NumPagesForLoader : Uint8B
   +0x060 FirstSecureRestorePage : Uint8B
   +0x068 FirstBootRestorePage : Uint8B
   +0x070 FirstKernelRestorePage : Uint8B
   +0x078 FirstChecksumRestorePage : Uint8B
   +0x080 NoChecksumEntries : Uint8B
   +0x088 PerfInfo         : _PO_HIBER_PERF
      [...snip...]
      +0x188 SecurePagesProcessed : Uint8B
      +0x190 BootPagesProcessed : Uint8B
      +0x198 KernelPagesProcessed : Uint8B
      [...snip...]
   [...snip...]
```

Fig. 6. Relevant members of `PO_MEMORY_IMAGE` (Windows 10 v1607 x64).

Processor context

The processor context is stored in the same fashion as described in Section Processor context with all current OS versions storing the `_KPROCESSOR_STATE` structure at offset `0x1000` from the beginning of hiberfil.sys.

Physical memory

The most significant change made in the Windows 8+ hibernation format deals with how the physical memory pages are stored. The notion of chained table pages, followed by XPRESS sets of compressed blocks of data has been abandoned in favor of an easier-to-parse approach, which will be described in this section.

Restoration sets

Windows 8+ groups physical pages of memory into one or more sets, which we are calling *restoration sets*. Each restoration set is stored in the hibernation file using the same on-disk structure, but are loaded into memory by different stages of the hibernation restoration process.

The `FirstBootRestorePage` member of the `PO_MEMORY_IMAGE` header contains the page number of the first restoration set stored in the image. Multiplying this number by the page size of 4096 gives the offset of the `BootRestorePages` in bytes. This restoration set is restored to memory in the first stage of the resume process by the kernel's bootloader, `winresume.exe`. The total number of pages in this restoration set is stored in both the `NumPagesForLoader` and `PerfInfo.BootPagesProcessed` fields of the file header.

An additional restoration set may also be present. If the `FirstKernelRestorePage` member of the header is non-zero, its value gives the page number of the start of the `KernelRestorePages`. This restoration set is restored by `ntoskrnl.exe` after the bootloader turns over control. The total number of pages in this restoration set is stored in the `PerfInfo.KernelPagesProcessed` member of the file header.

With the release of Windows 10 a `FirstSecureRestorePage` member was added to the `PO_MEMORY_IMAGE` header. This suggests that there may be a third potential restoration set, but we have not yet encountered a hibernation file in which this member has a non-zero value. While it is still uncertain when this restoration set is used, if ever, it may be a part of the new Secure Kernel

Please cite this article in press as: Sylve, et al., Modern windows hibernation file analysis, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.12.003

Mode (SKM) facility described by Ionescu (2014). Further research is needed to determine if this is the case and if the on-disk structure is the same as the two known restoration sets.

Compression sets

A restoration set contains one or more *compression sets* (our terminology), each of which stores the data from at most sixteen 4 KiB physical memory pages (64 KiB of data). A compression set starts with an undocumented 32-bit little-endian `compression_set_header` as defined in Fig. 7. The first 8 least significant bits encode the number of page descriptors that follow the header, if this value is either zero or greater than 16, the image is considered corrupt and hibernation is aborted. The next 22 least significant bits contain the size (in bytes) of the compressed data that follows the descriptors. The most significant bit indicates the compression algorithm used.

The `compression_set_header` is followed by `NumberOfDescs` page descriptors. These undocumented little-endian `page_descriptor` structures are 8 bytes long on 64-bit versions of Windows and 4 bytes long on 32-bit versions of Windows. The definitions of these structures can be found in Fig. 8. Each descriptor defines a contiguous page set of 4 KiB pages. The four least-significant bits encode the number of contiguous pages in the set. The actual number of pages in the set can be found by adding one to `NumPages`. The rest of the bits encode the physical page number of the first page in the contiguous page set. The first page's physical address can be calculated by multiplying `PageNum` by the page size of 4096 bytes. The sum of the number of pages in each of the contiguous page sets determines the total number of pages in the compression set.

The set of `page_descriptor` structures are followed by `SizeOfCompressedData` bytes of data. If `SizeOfCompressedData` is equal to the size of the total number of pages in the compression set then the page data is stored uncompressed, otherwise the data is compressed. If the `HuffmanCompressed` bit is set in the `compression_set_header`, the data is compressed using the LZ77+Huffman XPRESS algorithm as described in Microsoft (2016b); otherwise the Plain LZ77 XPRESS algorithm is used.

The uncompressed data consists of the concatenated pages of the contiguous page set in the order that they were defined. Until all of the pages in the restoration set are accounted for, the compressed data is immediately followed by the header for the next compression set.

Verification

To verify that our analysis of the undocumented Windows 8+ hibernation file format is correct, support for analyzing the format was added to our experimental memory analysis framework by creating a new *address translation layer*. In our framework an address translation layer is similar to an *address space* class in Volatility. It provides a layer of abstraction that translates a given physical address to an offset inside of the uncompressed data of the relevant compression set and returns the number of bytes requested. This allows higher level address translation layers to request data from the physical address space without direct knowledge of the underlying data storage format.

```
typedef struct _compression_set_header {
    uint32_t NumberOfDescs : 8;
    uint32_t SizeOfCompressedData : 22;
    uint32_t Unknown : 1;
    uint32_t HuffmanCompressed : 1;
} compression_set_header;
```

Fig. 7. Definition of the page descriptor header (Windows 8+).

```
// 64-bit page descriptor
typedef struct _page_descriptor {
    uint64_t NumPages : 4;
    uint64_t PageNum : 60;
} page_descriptor;

// 32-bit page descriptor
typedef struct _page_descriptor {
    uint32_t NumPages : 4;
    uint32_t PageNum : 28;
} page_descriptor;
```

Fig. 8. Definition of the 32 and 64-bit page descriptors (Windows 8+).

Hibernation files were then acquired from each of the relevant versions of Windows and processed through our framework. Each of our analysis plugins were able to execute successfully. As an example, Fig. 9 shows the partial output of our process listing plugin ran against the *hiberfil.sys* file from a hibernated Windows 10 v1607 system. While the majority of the output shown in Fig. 9 is parsed from `_EPROCESS` structures that reside in the kernel's address space, the final displayed property of the processes' path is read from the `_PEB` structure stored in each process's own address space. This shows that both kernel space and user space memory are successfully parsed from the hibernation file.

Additionally, since the framework powers the memory analysis capabilities in our popular forensics tool, *BlackLight* (Blackbag Technologies, 2016), it was possible to process each of the hibernation files through a development version of *BlackLight 2016 R3*. Each of the hibernation files produced expected results and all of the memory analysis features worked properly.

In order to test against other memory analysis tools it was necessary to first convert the hibernation files into raw memory dumps. Our memory analysis framework was instrumented to write to a file, at the appropriate offsets, the decompressed contents of each page stored in the hibernation file. The missing pages were padded with null bytes. The converted images were then processed through both the Volatility[1] and Rekall[2] memory analysis frameworks. Each of the relevant plugins in both tools gave reasonable results for all images tested.

Validation against Hibr2Bin

Once support for modern hibernation files became available in Hibr2Bin[3] it was possible to validate our methodology against its output. Hibr2Bin was used to convert a sample of *hiberfil.sys* files collected from both 32-bit and 64-bit versions of Windows 8+ systems to raw memory dumps. Each raw memory dump was then hashed using the MD5 and SHA1 algorithms and compared against hashes from dumps produced by our analysis framework as described in Section Verification.

The majority of the images created with Hibr2Bin were identical to those created using our methodology; However, images from the latest versions of Windows, Windows 10 v1607, produced drastically different images.

As an example a *hiberfil.sys* was taken from a Windows 10 v1607 x64 system and converted to raw with Hibr2Bin. Hibr2Bin reported that it decompressed a total of 53,906 pages, while our analysis tool reported that it decompressed a total of 367,296 pages. 53,906 of these pages were from the `BootRestorePages` and 313,390 were

[1] Volatility 2.5, commit 534374da57679dc353c974de45d27a42b81931ec.
[2] Rekall 1.5.3, commit 434458fb117c7ca56503491c89cb1b55b9acf908.
[3] Hibr2Bin 3.0.109.20161007.

Please cite this article in press as: Sylve, et al., Modern windows hibernation file analysis, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.12.003

Name	PID	PPID	Threads	Handles	Start Time	End Time	Path
System	4	0	144	----	Fri Aug 19 20:35:11 2016	----	----
smss.exe	340	4	2	----	Fri Aug 19 20:35:11 2016	----	----
csrss.exe	432	404	9	----	Fri Aug 19 20:35:15 2016	----	C:\Windows\system32\csrss.exe
wininit.exe	500	404	3	----	Fri Aug 19 20:35:15 2016	----	
services.exe	632	500	6	----	Fri Aug 19 20:35:15 2016	----	C:\Windows\system32\services.exe
lsass.exe	640	500	5	----	Fri Aug 19 20:35:15 2016	----	C:\Windows\system32\lsass.exe
svchost.exe	752	632	20	----	Fri Aug 19 20:35:15 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	820	632	6	----	Fri Aug 19 20:35:15 2016	----	C:\Windows\system32\svchost.exe
sppsvc.exe	876	632	0	----	Fri Aug 19 20:35:15 2016	Fri Aug 19 20:36:10 2016	----
svchost.exe	960	632	62	----	Fri Aug 19 20:35:16 2016	----	C:\Windows\System32\svchost.exe
svchost.exe	988	632	29	----	Fri Aug 19 20:35:16 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	356	632	6	----	Fri Aug 19 20:35:16 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	612	632	12	----	Fri Aug 19 20:35:16 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	1096	632	13	----	Fri Aug 19 20:35:20 2016	----	C:\Windows\System32\svchost.exe
svchost.exe	1148	632	8	----	Fri Aug 19 20:35:20 2016	----	C:\Windows\System32\svchost.exe
svchost.exe	1196	632	16	----	Fri Aug 19 20:35:21 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	1256	632	11	----	Fri Aug 19 20:35:21 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	1304	632	16	----	Fri Aug 19 20:35:21 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	1408	632	13	----	Fri Aug 19 20:35:21 2016	----	----
svchost.exe	1416	632	21	----	Fri Aug 19 20:35:21 2016	----	C:\Windows\system32\svchost.exe
svchost.exe	1660	632	5	----	Fri Aug 19 20:35:21 2016	----	C:\Windows\System32\svchost.exe
WUDFHost.exe	1808	1096	7	----	Fri Aug 19 20:35:21 2016	----	C:\Windows\System32\WUDFHost.exe
svchost.exe	2020	632	9	----	Fri Aug 19 20:35:22 2016	----	----
WUDFHost.exe	1116	1096	7	----	Fri Aug 19 20:35:22 2016	----	C:\Windows\System32\WUDFHost.exe
spoolsv.exe	2084	632	11	----	Fri Aug 19 20:35:22 2016	----	C:\Windows\System32\spoolsv.exe

Fig. 9. Process listing from a Windows 10 v1607 x64 hibernation file.

from the `KernelRestorePages`. The image produced by Hibr2Bin could not be processed by either Volatility or Rekall, while the image produced via our methodology worked flawlessly in both. This is likely because the majority of pages were missing from the Hibr2Bin image.

Since the number of `BootRestorePages` matched the total reported number of collected pages from Hibr2Bin we surmised that Hibr2Bin must only decompress the first restoration set of pages that are restored by the boot loader, ignoring the second set of kernel-restored pages. In all images in which Hibr2Bin produced exact output as using our methodology, the `FirstKernelRestorePage` member of the `_PO_MEMORY_IMAGE` header was set to zero, meaning that the `KernelRestorePages` were unused. For the remaining images, where the second restoration set was in use we hashed each of the `BootRestorePages` individually and compared them across images produced by both tools. In all cases each of the hashes matched exactly.

Since both tools were developed completely independently and produce comparable output we can surmise that the methodology described in this paper is correct. Although the version of Hibr2Bin available at the time of this writing fails to restore the `KernelRestorePages`, it is likely that these pages have simply been overlooked and could be restored by future versions of the tool using the same methods used to restore the `BootRestorePages`.

Analysis

In order to determine the effect on the hibernation file from different ways of powering off the system, we performed the following analysis.

For each version of Windows listed in Table 2 the following procedure was followed:

Table 2
Windows versions and their release dates.

Windows version	Release date
Windows 7 SP1	February 2011
Windows 8	August 2012
Windows 8.1	August 2013
Windows 10	July 2015
Windows 10 v1511	November 2015
Windows 10 v1607	August 2016

1. Install the OS on a newly formatted hard drive.
2. Hibernate the computer with the `shutdown /h` command.
3. Remove the hard drive and collect the *hiberfil.sys*
4. Replace the hard drive and turn the computer back on, either restoring the machine to the previously hibernated state (from Step 2), or going through the standard startup process (from Step 6).
5. Collect the *hiberfil.sys* from the live machine using FTK Imager (Access Data, 2016).
6. Shutdown the computer and repeat Steps 3–5.

For Windows 7, Step 6 was repeated by shutting down in the following ways:

- `shutdown /s` command
- shutdown from the GUI

Windows 8 introduced an additional "hybrid" shutdown mode. This mode is like full shutdown in that user sessions are terminated, but like hibernation in that the running kernel is hibernated and not fully shut down (Sinofsky, 2011). This allows for quicker boot times since the kernel space will already be initialized. Hence, for Windows 8+ Step 6 was repeated an additional time using the hybrid shutdown mode:

- `shutdown /s` command
- `shutdown /s /hybrid` command
- shutdown from the GUI

Windows 7 observations

Though they are well understood, we present the Windows 7 analysis results here for ease of comparison to the Windows 8+ results below.

When hibernated as in analysis Step 2, the resulting hibernation file acquired in Step 3 contains the complete file header and all expected hibernation data for the entire running state of the machine – kernel and user space; However, the hibernation file collected after resuming from hibernation (Step 5) has the first page zeroed out by Windows during the resume process. The hibernation file can still be analyzed by using the methods described in Section First table page. Both methods of shutting down the system in Step 6 left the hibernation file unchanged from the state where the first page is zeroed.

Please cite this article in press as: Sylve, et al., Modern windows hibernation file analysis, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.12.003

Windows 8+ observations

As with Windows 7, the Windows 8+ hibernated `hiberfil.sys` files from Steps 2 and 3 contained the complete file header and all expected hibernation data, albeit in the new format.

The other collected hibernation files exhibited different behaviors than that of their Windows 7 counterparts. The hibernation files collected in Step 5, after resuming from hibernation, still contained the file header, but with the header signature changed from `RSRT` to `WAKE`. Additionally, all data after the first page was zeroed, rendering the hibernation file effectively useless for analysis; no memory data or processor state remains. Performing a full shutdown from the command line using the `shutdown /s` command makes no changes to the (nearly empty) hibernation file.

Both shutting down the machine using either the `/hybrid` switch and using the shutdown button in the GUI results in a "hybrid" hibernation mode. In this mode, all userland sessions are closed as during normal shutdown, and only the running kernel is saved to the hibernation file. A valid hibernation file is produced; however, a much smaller subset of memory is available for analysis. The resulting file can be analyzed for kernel data (e.g., loaded modules and `_EPROCESS` structures that track processes) but without much of the process context that normally resides in userland memory. Even in the hybrid hibernation case, any method of restarting the machine leaves the header signature as `WAKE` and all other data zeroed.

Forensic implications windows 8+

The results of our analysis have several important implications for forensic practitioners that analyze machines running Windows 8+. Due to the changes in behavior introduced with Windows 8, the data lifetime of the stored information in the hibernation file has been reduced to the time between hibernation and the first power-on, while in previous versions of Windows that data would be present until the next hibernation event. This limited lifetime has the following implications:

- The hibernation file is no longer a reliable source of information about a machine's state from "far in the past". In older versions of Windows, hibernation files could contain data from months or even years prior if the machine was not frequently hibernated.
- Collecting the hibernation file from a running machine is now largely useless, as powering on the machine zeroes the bulk of the hibernation file.
- The `shutdown /s` command is commonly used to shutdown remote systems on a network. Systems that are powered down in this fashion will contain no hibernation data. Similarly, shutting down the system by removing the power or "pulling the plug" will leave no hibernation data.
- Presumably, the most common way to shutdown systems is by using the GUI. Systems that are shutdown in this fashion or by using the `shutdown /s /hybrid` command will contain only partial hibernation data. While these images can still contain valuable forensic data, the lack of userland memory limits the analysis to only a subset of the kernel structures that still reside in non-freed pages.
- Powering down the system by forcing a hibernation using the `shutdown /h` command preserves the largest amount of hibernation data.

Conclusion

We have provided an overview of the legacy hibernation file format as well as present the first analysis of the new format as used in Windows 8, 8.1, and 10. We have implemented support for the new format in our experimental memory analysis framework. We have also discussed several changes in the hibernation and shutdown behavior of Windows that will have a direct impact on digital forensic practitioners who use hibernation files as sources of evidence.

While we believe that with the addition of this research, hibernation files will continue to be a valuable source of digital forensic evidence, extra care must be taken when powering down a target computer to ensure that the greatest amount of hibernation data is available.

References

Access Data, February 2016. Ftk Imager Version 3.4.2 (For Use with Version 6 Products and Newer) — Accessdata. http://accessdata.com/product-download/digital-forensics/ftk-imager-version-3.4.2.

Blackbag Technologies, October 2016. Blacklight. https://www.blackbagtech.com/software-products/blacklight.html.

Dolan-Gavitt, B., April 2009. Add Support for Inactive Hiberfiles to Hibinfo Volatilityfoundation/volatility@552c1d8. https://github.com/volatilityfoundation/volatility/commit/552c1d813b05a0bf8d3d1ec1f64b3ba5f98403cc.

Ionescu, A., August 2014. Battle of SKM and IUM. How Windows 10 Rewrites OS Architecture. http://www.alex-ionescu.com/blackhat2015.pdf. Presentation given at Blackhat USA 2015.

Microsoft, October 2012. Windows 8 Arrives — News Center. https://news.microsoft.com/2012/10/25/windows-8-arrives/.

Microsoft, July 2015. Hibernate Once/Resume Many (Horm) (Standard 8). https://msdn.microsoft.com/en-us/library/jj980177(v=winembedded.81).aspx.

Microsoft, July 2016a. Debugging Tools for Windows (WinDbg, KD, CDB, NTSD) – Windows 10 Hardware Dev. https://msdn.microsoft.com/en-us/library/windows/hardware/ff551063(v=vs.85).aspx.

Microsoft, July 2016b. [MS-XCA]: Xpress Compression Algorithm. https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-XCA/[MS-XCA].pdf.

Ruff, N., Suiche, M., November 2007. Enter Sandman. http://www.msuiche.net/pres/PacSec07-slides-0.4.pdf. Presentation given at PacSec 2007.

Sinofsky, S., September 2011. Delivering Fast Boot Times in Windows 8 Building Windows 8. https://blogs.msdn.microsoft.com/b8/2011/09/08/delivering-fast-boot-times-in-windows-8/.

Suiche, M., May 2016a. [Vol-Users] Hibr2bin Beta 1. https://www.mail-archive.com/vol-users@volatilesystems.com/msg00053.html (Email sent to the Volatility-Users mailing list).

Suiche, M., September 2016b. Your Favorite Memory Toolkit is Back for Free!. https://blog.comae.io/your-favorite-memory-toolkit-is-back-f97072d33d5c#.xq8az9jv4.

The Rekall Team, 2013–2016. The Rekall Memory Forensic Framework. http://www.rekall-forensic.com/.

The Volatility Foundation, 2007–2016. The Volatility Framework. http://www.volatilityfoundation.org/.

Please cite this article in press as: Sylve, et al., Modern windows hibernation file analysis, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.12.003