TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Shaymaa Mamdouh Mohammed Radwan Khalil 177237IVCM

ANALYSIS OF WINDOWS 10 HIBERNATION FILE

Master's thesis

Supervisor: Hayretdin Bahsi
Ph.D., Research
Professor

Pavel Tšikul
Ph.D. Researcher

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Shaymaa Mamdouh Mohammed Radwan Khalil 177237IVCM

WINDOWS PUHKEOLEKUFAILIDE ANALÜÜS

magistritöö

Juhendaja: Hayretdin Bahsi
Ph.D., Research Professor

Pavel Tšikul
Ph.D. Researcher

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Shaymaa Mamdouh Mohammed Radwan Khalil

19.05.2020

Abstract

Since the reveal of Windows hibernation file structure by Mattieu Suiche late in 2007, the hibernation file became a valuable source of artifacts for digital forensics examiners. Starting from Windows 8, Microsoft changed the hibernation file structure. Therefore, many digital forensics tools are no longer supporting the direct analysis of modern hibernation files. The literature shows that the modern hibernation file almost loses its value once the system is resumed, as all the file's content is zeroed, except the header. Such behavior modification of hibernation file content toward power state changes has caused some digital forensics practitioners to lose interest in including the file in their investigations.

One of the aims of presenting this study is to raise awareness about the forensic value of Windows 10 hibernation file and highlight special considerations to be taken when processing the file. The study analyses the hibernation file structure of Windows 10 versions 1809, 1903, 1909, and provides an updated layout of the file. Moreover, this research documents the impact of different Windows 10 configurations on the hibernation file content. A predefined list of evidence was created to compare the output of hibernation file analysis using BEC, Magnet Axiom, and BlackLight tools. The thesis also evaluates alternative tools that could be used in the analysis of Windows 10 hibernation file.

The study demonstrates that Windows 10 hibernation file is a valuable source of volatile evidence. The results of this research show that Windows 10 hibernation file contains valuable data related to running processes, opened connections, private browsing history, and other types of evidence that might not be found in a disk image. This research recommends considering the hibernation file as a memory image substitute, in case the memory image was not taken from live evidence, memory image corruption, or the device was found in hibernation or shutdown state.

This thesis is written in the English language and is 121 pages long, including 7 chapters, 63 figures, and 30 tables.

List of abbreviations and terms

ACPI	Advanced Configuration and Power Interface
BCD	Boot Configuration Database
BEC	Belkasoft Evidence Center
BIOS	Basic Input Output System
CR	Control Register
DFRWS	Digital Forensics Research Workshop
DHCP	Dynamic Host Configuration Protocol
EXIF	Exchangeable Image File Format
FAQ	Frequently Asked Questions
GDT	Global Descriptor Table
HORM	Hibernate Once/Resume Many
IDT	Interrupt Descriptor Table
MBR	Master Boot Record
N/A	Not Available
OS	Operating System
PC	Personal Computer
POST	Power-on self-test
RAM	Random Access Memory
SKM	Secure Kernel Mode
UEFI	Unified Extensible Firmware Interface
VAD	Virtual Address Descriptors

Table of contents

List of figures

List of tables

1 Introduction

Digital Forensics is a modern science as it all started in the late 80's. During the first decade, digital forensics investigations were focused on data recovery, as users frequently deleted data due to the high cost of storage media. Many commercial tools were designed initially for data recovery and file management, then later adapted for digital forensics use. A few years later, digital forensics examiners were able to see the past by recovering deleted data, emails, and instant messages, months after it happened. Nowadays, more and more commercial and open source forensics tools are developed, targeting different types of systems, and allow individuals with limited training to extract valuable data [1].

Memory forensics is a great source of evidence for forensic examiners and incident responders. The use of memory forensics is not just related to law enforcement. Malware analysts also have a huge interest in memory forensics, as well as red teams, and pen testers. Hibernation (also known as suspend to disk and S4 system power state by ACPI nomenclature) is a feature related to computer power management. The hibernation power state is not related to a specific operating system. This feature is available on Windows (starting from Windows 2000), Linux systems [2], and macOS. However, on Mac devices, this power state is named safe sleep or deep sleep [3]. Once the hibernation command is provided to the system, the operating system will write all system context to a file on non-volatile storage media and leave appropriate context markers [4]. Such functional behavior of the hibernation feature has turned it into a great source of information in the memory forensics investigations.

The hibernation file is a hidden system file located in the root folder of Windows OS drive. Windows uses the Hiberfil.sys file to store a copy of the system memory on the hard disk. Enabling hibernation is easy and does not require system reboot or any special physical requisites [5].

Statistics show that the Windows operating system is the dominant desktop operating system. The statistics of December 2019, published by StatCounter web analysis service, show that Windows OS has a market share of 77,64% of the desktop operating system market share [6]. Statistics for the same month show that Windows 10 consumers are 65.4% of the number of Windows operating system users [7], followed by Windows 7 with 26.79%. The combined rates of Windows 8.1, XP, and 8 are less than 8% of Windows operating system market share. It is to be noticed that Windows 8 and Windows 7 officially end of sales for a few years. The end of extended support for Windows 7, service pack 1 was January 2020. End of support and retired Windows versions would give a huge market share for Windows 10. By the time of writing this thesis, Microsoft has officially released nine versions of Windows 10, starting from July 2015 till the end of 2019. The most recent version of Windows is currently version 1909, which was released in November 2019 [8].

The idea of this thesis first came during the system forensics course provided by Pavel Laptev, who raised the need for reconsideration of Windows 10 hibernation files in digital forensics investigations, as they seem to become useless. We proceeded by creating a survey about the use of the hibernation file in digital forensics investigations and malware analysis. Convincing digital forensics experienced individuals to fill such a survey was a difficult mission. Despite the use of LinkedIn and personal network, only eight feedbacks were collected. The collected feedbacks include four persons with less than one year of experience in the digital forensics field. On the other hand, the other four survey respondents had from one to five years of experience as digital forensics specialists or malware analysts. The survey's feedback shows that individuals with limited experience in digital forensics investigations - less than one year of experience - have never used the hibernation file in their investigations. On the contrary, professionals' answers show that they have used this file at least once during their years of experience. It was motivating to find that six out of the eight received feedbacks were interested in some guidelines to demonstrate how they can include the hibernation file to their investigations.

The results of an interview that we did with two forensics examiners working at the Estonian Forensic Science Institute[1], showed that experienced forensics examiners are aware of the importance of the hibernation file in digital forensics investigations, the two examiners confirmed that they previously used the hibernation file in their investigations, using Magnet IEF tool [9], while they did not get the chance to test the tool on a Windows 10 hibernation file. They clarified that in most of the cases they receive in real life, no memory image is acquired at the crime scene, while it happens that some Windows

[1] https://www.ekei.ee/en

devices are found hibernated on the scene. The examiners stated that they included the hibernation file to their investigations to extract data related to web browsing, and they also used it to recover some files from ransomware infection with CryptoLocker, as they extracted the encryption keys from the hibernation file.

1.1 Research Objectives

For Windows versions before Windows 8, the hibernation file was used as a substitute for memory analysis in many cases. Some examples include the first responder who did not take a memory image for the running computer, the evidence was found powered off, hibernated, or the memory image collected from the evidence was corrupted. For some Windows versions before Windows 8, the hibernation file (even old hibernation file taken for backup) was a great support in forensic investigations. The use of the hibernation file made it possible to recover chat sessions, running processes, login credentials, encryption keys for encrypted devices, web history, program, and email data. It was also helpful in some cases of malware analysis, especially memory-only malware and rootkits [10].

The structure of the hibernation file was dramatically modified starting from Windows 8. For the legacy hibernation file format, when the device is resumed, only the headers of the files are replaced by zeros. The rest of a legacy file content remains as it is, so the file's data might be kept for months or years in some special cases. An example of such a case was a device that was not hibernated for a long period. On the contrary, Windows8+ hibernation file keeps only the headers once the system is resumed, while the whole content of the file after the first page is replaced by zeros [11]. Such behavior makes the new versions almost useless once the device is resumed. When talking about the new structure of hibernation file, used starting from windows 8, we would refer to it as "modern hibernation file", this name was first introduced in the "Modern Windows hibernation file analysis" paper [11].

The goal of this thesis is to answer to the below list of questions:

1. Is a hibernation file created by default by Windows 10?
2. Could we document the known hibernation file structures in a clear mapping/datagram?

3. Is the Modern hibernation file layout [11] still applicable for the latest versions of Windows 10 hibernation file?
4. Do the free tools decompressing Windows 10 hibernation file have the same output file when processing the same input hibernation file?
5. What are the impacts of the modifications of power configurations, as well as power states on a hibernation file content?
6. What is the effect of enabling the “HORM” feature on the hibernation file characteristics?
7. Could we extract artifacts from Windows 10 hibernation files using free tools?
8. Are there any differences found between commercial tools outputs?
9. What kind of artifacts could be collected from a Windows 10 hibernation file?
10. In which Windows 10 power state, a hibernation file would contain the maximum number of artifacts?

1.2 Scope

This thesis topic aims to analyze hibernation files collected from different Windows 10 versions, document the current file structure, propose a list of currently available tools that could support the file analysis, and list some types of artifacts that could be extracted from the file. As the thesis scope is Windows 10 hibernation file, we would use the modern hibernation file layout proposed in [11], as they already did a proof of concept for their findings on older versions of Windows 10. We aim to confirm if the latest versions of Windows 10 still use the same proposed structure in [11] or not.

The limitations of this study should take into consideration that the hibernation file is Microsoft proprietary, and they have not released documents about the structure of this file. Most of the academic papers related to Windows hibernation base their analysis on Suiche reverse engineering for the legacy hibernation file structure [12], and his documents about Microsoft compression. The legacy file structure is not applicable anymore for Windows 10 hibernation file.

The online available system forensics documents are not widely spread. Besides, the available free cheat sheets do not seem to be relevant for Windows 10 hibernation file analysis. The manual analysis of Windows 10 hibernation file was not easily achievable, considering our limited knowledge about the computer's memory structure, and the system forensics field. In the first period of this study, we performed some manual analysis and got blocked in the first steps, so we had to read more about memory forensics, volatile and physical memories as well as Windows 10 operating system. The researcher's background experience was another limitation as most of the tools used during this research were used for the first time. Such limitation is time-consuming to learn how to use each tool and explore its capabilities. Taking into consideration the limited time of trial versions of the commercial tools, and the research period, it was challenging to overcome these technical knowledge limitations.

1.3 Novelty

The number of papers directly related to Windows 10 hibernation files are very limited. The main paper on the modern Windows hibernation file is [11], and it was released with cooperation with Blackbag technologies. The paper introduced the modern hibernation file structure, while also promoting the hibernation file analysis feature added to BlackLight commercial software. As on the 8th of December 2019, Google scholar indicated only one citation for paper [11], in a paper named "Memory forensics: the path forward" [13] that was also released in 2017. The paper [13] did not contain new information about the modern hibernation files structure. Our literature shows that the paper [11] is currently the only paper discussing the modern hibernation file structure. We found some other papers with misleading titles, that describe legacy hibernation file structures as Windows 10 file structure, this point would be discussed in more detail in section 2.4 (Related work). There is a clear research gap in the studies related to Windows 10 hibernation file. Some commercial tools claim that they can extract artifacts data from the modern hibernation files. However, there has not been any research yet to test the efficiency of these tools or compare them. At the time of writing, we have not found a document to confirm if all the previously identified artifacts are still collectible from a Windows 10 hibernation file. There are also no documents that propose a list of tools to be used in Windows 10 hibernation file analysis and compare the performance of these tools.

2 Background

This chapter includes terminologies and concepts that would be used during the research. It provides background about memory forensics, power management, and the hibernation file. The hibernation file sections provide details about the hibernation power state, the hibernation file size, type, usage, legacy and modern file structures, the legacy hibernation file security and a literature review. An introduction to the tools used during this research is also included in this background chapter.

2.1 Memory forensics

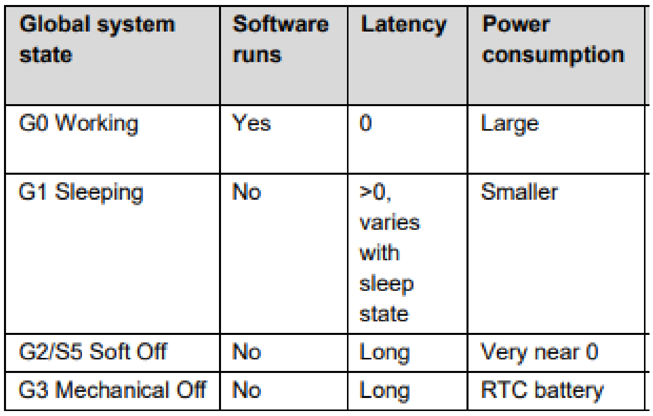
Memory analysis for forensics investigations first appeared in the early 2000s. By this time, digital forensics investigators were using system administration tools like grep, strings, and hex editors in their investigations. This investigation technique was named “unstructured analysis” and was useful for extracting information related to user activity, passwords, and encryption keys. In 2005, the DFRWS[1] released a challenge that requires the analysis of a Windows memory sample. This challenge led to the creation of many memory analysis tools. Since then, many commercial tools and open-source frameworks were developed targeting memory forensics with capabilities of extracting attacker activities, defeating anti-forensics techniques and the scope of some of these tools were extended to detect some types of malware and malicious activities [13].

Analysis of the hibernation file is a part of memory forensics techniques. Memory forensics techniques could extract data that is not available in network or disk forensics. For example, browsers that implement “private browsing”, do not save many artifacts to disk. Another type of artifacts that could be found using memory forensics is the chat sessions related to chat applications that use end-to-end encryption, with chat logging disabled, like Adium and Pidgin [13].

[1] https://dfrws.org/about-us/

2.2 Power Management basics

The Advanced Configuration and Power Interface (ACPI) [14] is an open standard, with two primary roles of device configuration and power management. This standard helps to reduce market fragmentation using its open framework. ACPI defines the global system states (G0-G3), which are applied to the system and visible to the user; it also includes different system Sx states (S0-S5). The sleeping states (S1-S4) are types of sleeping states within the global system state G1 [4]. Figure 1 shows the global system states defined by ACPI.



Global system state	Software runs	Latency	Power consumption
G0 Working	Yes	0	Large
G1 Sleeping	No	>0, varies with sleep state	Smaller
G2/S5 Soft Off	No	Long	Very near 0
G3 Mechanical Off	No	Long	RTC battery

Figure 1. Summary of Global system state as per ACPI standard [4].

Microsoft applied the ACPI standard in Windows systems power management. The hibernation state (S4), first used mainly to extend a mobile device battery life, as it saves the user state by preserving the operating system, applications, and devices states. All the content of physical memory is saved to hiberfil.sys on the primary system drive. The hibernation file should be large enough to ensure that it would save all the content of the physical memory. Once hibernated, the volatile memory does not require refreshing anymore, and it is powered-off, which makes the hibernation state the lowest power-consuming state in the sleep states [15].

The hibernation file is not only used by the hibernation power state. Table 1 shows that the hibernation file is also used in the hybrid sleep state, which is a combination between the hibernation power state and the sleep state. Starting from Windows 8, a full shutdown (S5) occurs when a system restart is requested, while when a system shutdown is requested using the Windows startup menu, the system will transition to a fast startup, which uses the hibernation file [15].

Table 1. Lists the ACPI power states from highest to lowest power consumption [15].

Power state	**ACPI state**	**Description**
Working	S0	The system is fully usable. Hardware components that are not in use can save power by entering a lower power state.
Sleep (Modern Standby)	S0 low-power idle	In this state, the system remains partially running, it can very quickly switch from a low-power state to a high-power state. Systems that support Modern Standby do not use S1-S3, as it is faster than the S1-S3 sleep states.
Sleep	S1 S2 S3	The system appears to be off. Power consumed in these states (S1-S3) is less than S0 and more than S4; S3 consumes less power than S2, and S2 consumes less power than S1. Systems typically support one of these three states, not all three. In these states (S1-S3), volatile memory is kept refreshed to maintain the system state. Some components remain powered so the computer can wake from input from the keyboard, LAN, or a USB device. **Hybrid sleep,** used on desktops, is where a system uses a **hibernation file with S1-S3**. The hibernation file saves the system state in case the system loses power while in sleep.
Hibernate	S4	The system appears to be off. Power consumption is reduced to the lowest level. The system saves the contents of volatile memory to a hibernation file to preserve the system state. Some components remain powered so the computer can wake from input from the keyboard, LAN, or a USB device. The working context can be restored if it is stored on non-volatile media. **Fast startup** is where the user is logged off before the hibernation file is created. This allows for a smaller hibernation file, more appropriate for systems with fewer storage capabilities.
Soft Off	S5	The system appears to be off. This state is comprised of a full shutdown and boot cycle.
Mechanical Off	G3	The system is completely off and consumes no power. The system returns to the working state only after a full reboot.

Fast startup is a type of shutdown that uses a reduced sized hibernation file, to speed up the booting time. The user session is not included in the hibernation file created for a fast startup, as the user is logged off before the creation of the file, while the content of kernel (session 0) is written to the hard disk, which generates a small hibernation file size. Starting from Windows 8, a fast startup is the default response to a system shutdown request [15].

The computer cannot do a direct transition between any states from S1 to S4 (sleeping states). Instead, it always comes back to the S0 state (working state), which is shown in Figure 2. System moving from S1-S5 states to S0 is waking, the system moving from S0 state to S1-S5 is said to be sleeping (in fact, for S5, it would be fully powered off) [16].

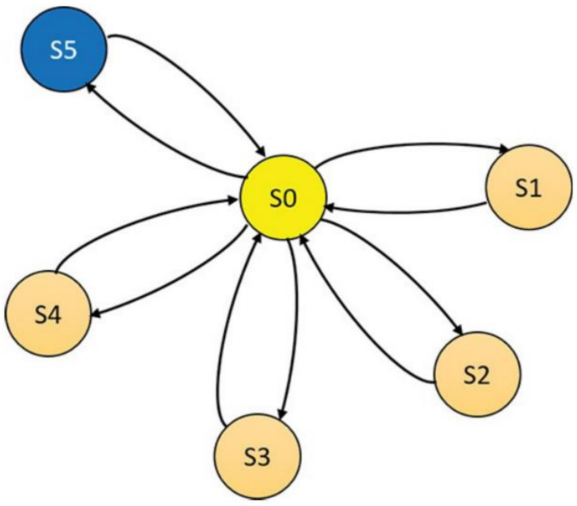


Figure 2. System power state transitions [16].

2.3 Hibernation file

It is important to differentiate between the physical and the virtual memory of a computer system. The virtual memory purpose is to compensate for the limited space of physical memory by reserving a part of the computer hard disk for memory usage. Once the system detects high usage of RAM space, some of the physical memory content is written to the hard drive to free up some memory space. This content is then retrieved from the virtual memory once the system needs to process these data again by the memory [17].

The root directory of a Windows 10 system contains three system files that store memory content. The three files are hidden files. To view these files, users must modify the default

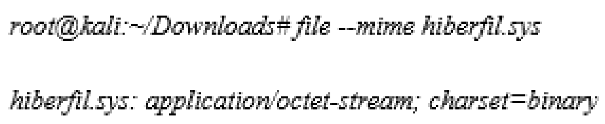
Windows view of folder options, display hidden files, and unmark the “Hide protected operating system files” option. The three files are:

1. “pagefile.sys”: This file is a part of the virtual memory of a computer. *“Page files enable the system to remove infrequently accessed modified pages from physical memory to let the system use physical memory more efficiently for more frequently accessed pages. Page files can be used to "back" (or support) system crash dumps and extend how much system-committed memory (also known as “virtual memory”) a system can support”* [18].

   For Windows 10 Pro and Enterprise editions, there is a group policy option “Shutdown: Clear virtual memory pagefile” that clears the memory Pagefile on shutdown [19], this group policy is disabled by default. Such an option is enhancing the security of sensitive data available on the pagefile. The same functionality could be enabled for Windows Home edition users using a registry key “ClearPageFileAtShutdown”.

2. “swapfile.sys”: The swap file first appeared in Windows 8, with the introduction of Windows modern applications [also named as “Universal Windows Platform Apps”, “Windows Store Apps”, “Metro Apps” [13], which are applications that could be downloaded from Microsoft store [20]. The swap file is a part of the virtual memory, so it is used to save physical memory space, like the paging file, while the swap file is currently dedicated to suspending modern applications.

3. “hiberfil.sys”: *“A hibernation file (hiberfil.sys) contains a compressed copy of memory that the system dumps to disk during the hibernation process”* [21]. The hibernation file is a protected file system, copying the file from the root drive would fail. Exporting the file using FTK imager (with run as administrator option) for example, might be, in some cases, a solution to overcome the system protection for the file. The hibernation file is binary. Figure 3 shows the output of the “file” command when running on a hibernation file.



```
root@kali:~/Downloads# file --mime hiberfil.sys

hiberfil.sys: application/octet-stream; charset=binary
```

Figure 3. Hibernation file type.

2.3.1 Entering hibernation state

The following steps are published by Microsoft [15], to explain what happens when a hibernation request is received by the Windows system:

1. Notifications sent to applications and services.
2. Notifications sent to drivers.
3. User and system state are saved to disk (hibernation file) in a compressed format.
4. Notifications sent to the firmware.

This hibernation request is done through a Windows function named SetSuspendState[1].

2.3.2 Resuming system from hibernation

When the system is powered on from hibernation, and the boot manager detects a valid hibernation file, the content of system memory is resumed using the data saved in the hibernation file, and the system is resumed to the exact state it was in before hibernation.

Microsoft briefed in [15] what happens when a system is resumed from hibernation. It might be advantageous for forensics investigators and malware researchers to understand the whole startup process of Windows 10, as this might support their investigations. Below the Windows 10 startup process as identified by [22] and [15].

1. Once the computer is powered on, it performs some checks named power-on self-test (POST), these checks are done by the UEFI in case of modern computers, or by the BIOS in case of older devices. The POST check search for a configured boot device that contains a valid master boot record (MBR), which stores the partition information on the bootable device (hard disk for example). Once these data is retrieved, the computer access the primary active partition (often referred as system partition and it does not have an assigned drive letter), and loads the boot manager file BOOTMGR from the root directory of the active partition [22].
2. BOOTMGR checks the boot configuration data available in the BCD database, which could be found on the same path as the BOOTMGR. The BCD database

[1] https://docs.microsoft.com/en-us/Windows/win32/api/powrprof/nf-powrprof-setsuspendstate

helps the boot manager to determine if one or more operating systems are installed on the device, and also it detects if the system is in the hibernation state [22].

3. If a Hibernation file is found by the boot manager, it passes the control to Winresume.exe to restore the system to its previous state. Otherwise, the control is given to Winload.exe. These activities end by initializing the operating system kernel (Ntoskrnl.exe) [22].

4. Drivers are restored to their previous state in case of hibernation restoration [15].

5. Services are restored to the state they were in before hibernation [15]. At this stage, the Windows session is loaded, and the display is switched to graphical mode [22].

6. The system becomes available for login [15].

2.3.3 File size and types

Windows systems compress the memory content before saving it to the hibernation file, to reduce disk space usage. The default proportion between the hibernation file size and the total physical memory on the system may differ from a Windows version to another. For example, for Windows 7, the hibernation is enabled by default on the system, the actual size of Hiberfil.sys is equal to 75% of the total physical memory on the system (for a computer of 2 GB RAM, the default hibernation file size would be 1.5 GB RAM). On the other hand, the hibernation is disabled by default on Windows server 2008 R2, and once enabled the hibernation file size is equal to 100% of the total physical file size. Windows provide an option of adjustment of the size of the Windows hibernation file using the built-in PowerCfg command-line utility [23].

As for Windows 10, we already mentioned that the system creates a hibernation file in the case of three power states: hybrid sleep, fast startup, and the standard hibernation. The size of the hibernation file has two types: full and reduced size files and each type of file supports power sleep mode as clarified in Table 2 [15].

Table 2. Hibernation file types and default sizes [15].

Hibernation file type	Default size	Supports
Full	40% of physical memory	hibernate, hybrid sleep, fast startup
Reduced	20% of physical memory	fast startup

2.3.4 Usage

The Art of Memory Forensics book [21] proposed factors to consider before acquiring memory. In Figure 4, The book recommends considering the hibernation file in case the computer was found “not running”. Such action matches the modern hibernation file characteristics, as the file should be collected offline to contain a useful amount of data into the file.

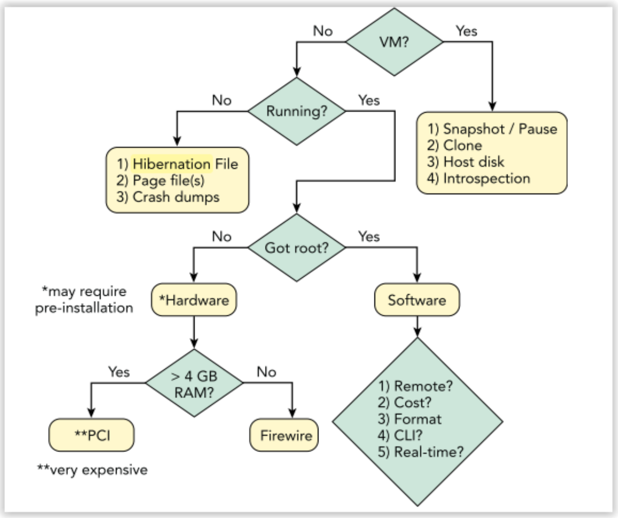


Figure 4. Some of the factors to be considered before acquiring memory [21].

The book highlights that before hibernating, the DHCP configuration would be released and the active connection would be terminated, which leads to incomplete networking data of the hibernation file. Some malware might remove themselves from memory during the hibernation, so the hibernation file would not detect the presence of such malware. Such behavior favors the full memory image to the hibernation file; for this reason, the book recommended using the hibernation file in case of unavailability of a memory image[21].

2.3.5 File structure

2.3.5.1 Windows XP-7 Hibernation file structure

Mattieu Suiche and Nicolas Ruff were the first to publicly document the hibernation file format in PacSec 2007[1] [24]. Later Suiche provided another presentation during the Black Hat Briefings in the USA in 2008 [25]. Suiche presentations were essential to understand the legacy (also named as Windows XP-7 by [11]) hibernation file format.

Figure 5 shows the hibernation file layout of Windows 7, as per [11]. We noticed that this structure has some modifications when compared to the structure described by Suiche in [24] and [26]. The described layout in Suiche's documents seems to be more relevant to Windows XP and Vista. Suiche mentioned that some pages order modifies depending on the Windows versions [27], while we also noticed that some pages mentioned in Suiche documentations are not available in Windows XP-7 hibernation file layout presented by [11] (for example the FreeMap page is not available in the Figure 5 layout). As [11] is our primary reference paper, and the most recent published paper related to the hibernation file layout topic, we decided to use the layout presented in [11], while highlighting the differences between this layout and other documentations layouts and naming conventions.

The first page of a hibernation file is named the file header, and its content is defined by the "PO_MEMORY_IMAGE" structure. The first four bytes segment of the headers are the file signature (also named magic bytes), and it varies depending on the system state. While the system is resuming from hibernation, the signature value changes to rstr/RSTR, once the system is successfully resumed, the header content is lost as the first page of the hibernation file is zeroed [11]. The value of the "Signature" of a hibernated file would be hibr/HIBR (depending on the OS version) in the hibernation state [11]. The header contains interesting information about the system, such as the system time, the page size, and the "FirstTablePage" that contains the address of the first memory table [28].

[1] This presentation was available on Suiche's website in the past, while the website is no longer available, the last archive for the website is dated for the 4th of October 2017, and the domain msuiche.net is currently owned by another person. This is an example of references previously mentioned in academic papers related to hibernation file, that are not valid anymore. In such cases we used the search in the web archive (https://web.archive.org/) to find the required resources.

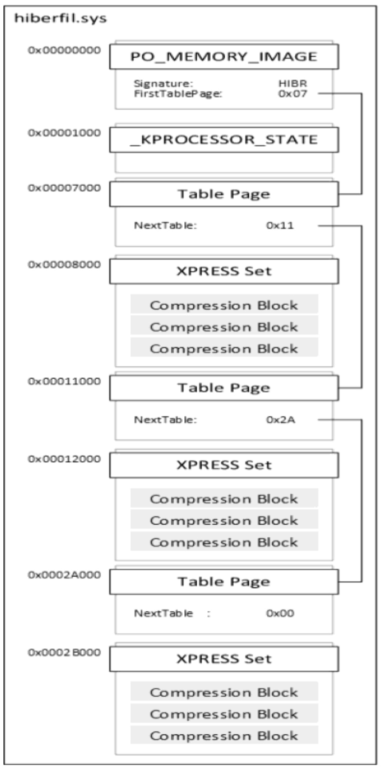
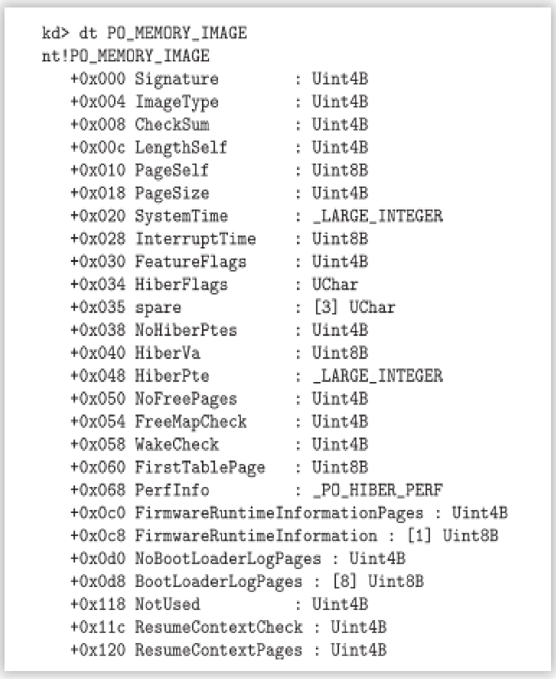


Figure 5. Example of Windows 7 hiberfil.sys layout [11].

The "PO_MEMORY_IMAGE" structure may vary depending on the Windows version. The header structure definition could be found in debugging symbols and could be extracted using WinDbg [29]. Figure 6 shows the definition of "PO_MEMORY_IMAGE" for Windows 7 SP1 x64.



```
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
```

Figure 6. Definition of the "PO_MEMORY_IMAGE" for Windows 7 SP1 x64 [11].

The processor context becomes the second page of the hibernation file starting from Windows Vista. This page is composed of the structure “_KPROCESSOR_STATE”, where Windows store a copy of every processor register value. The “_KPROCESSOR_STATE” structure is defined in the debugging symbols and could be extracted using WinDbg. The offset of this page gets modified depending on the OS versions [the starting offset is 0x1000 for Windows versions starting from Windows Vista SP1] [11].

The processor context page contains information about the registers saved by ntoskrnl.exe [27], similar to:

- Control registers: These registers determine the operating mode of the processor and the characteristics of the currently executing task, they are 32 or 64 bits depending on the processor. Table 3 provide examples of control registers and their function that we created based on [30] and [11].
- Global descriptor table (GDT)[1] offset.
- Interrupt Descriptor Table (IDT)[2] offset.

Table 3. Control Register Functions as defined by [30] and [11].

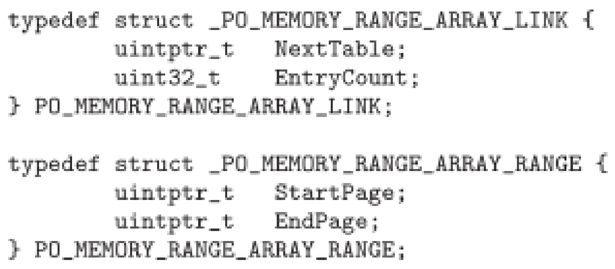
Control Register	Function
CR0	It contains system control flags that control operating mode and states of the processor.
CR2	It contains the page-fault linear address (the linear address that caused a page fault).
CR3	It contains the physical address of the system’s page tables.
CR4	It contains a group of flags, like the page address extensions (PAE).
CR8	Provide read and write access to the Task Priority Register (TPR).

[1] When operating in protected mode, all memory accesses pass through either the GDT or an optional local descriptor table (LDT), these tables contain entries called segment descriptors. Segment descriptors provide the base address of segments well as access rights, type, and usage information [30].

[2] External interrupts, software interrupts and exceptions are handled through the IDT. The IDT stores a collection of gate descriptors that provide access to interrupt and exception handlers [30].

The system needs to reconstruct data extracted from the "table pages" and "XPRESS sets" to restore the content of Windows XP-7 hibernation file. A "table page"[1] contains two structures that start with "_PO_MEMORY_RANGE_ARRAY"[2]. Each "table page" is followed by the "XPRESS set" of compression blocks. The combination of data available in "table pages" and "XPRESS sets" provide all the meta-data and data required for restoring the state of physical memory [11].

Figure 7 shows that the "table page" is composed of two structures. The header consists of the "_PO_MEMORY_RANGE_ARRAY_LINK" structure, which contains the fields "NextTable" and "EntryCount". A "table page" cannot exceed 4KB, so the "NextTable" field points to the offset of the next "table page" in the hibernation file. The value of "NextTable" should be multiplied by 4096 or 0x1000 to find the offset of the next table page. The last "table page" has the "NextTable" entry as zero [11].



```
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;
```

Figure 7. Definition of "table page" structures [11].

As per [11], the header is followed by a "number" of entries of the structure "_PO_MEMORY_RANGE_ARRAY_RANGE" (same number as "EntryCount"[3]), each

---

[1] The "table page" is a terminology used only in [11], while [12] and [27] named these pages "memory range array", on the other hand [35], [36] and [28] named it "memory table".

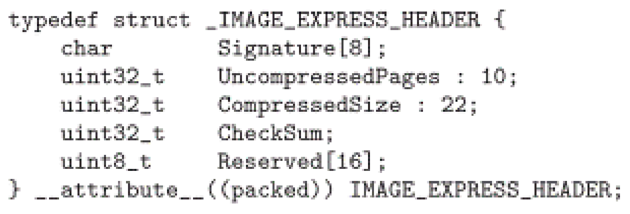
[2] By searching for the structure name "_PO_MEMORY_RANGE_ARRAY" in a site publishing different versions of Windows kernel structure, we found that this structure name was last found in Windows Vista SP1 [31]. As legacy hibernation file as out of the scope of this research, no further analysis was done to double-check this information and understand what replaced the "_PO_MEMORY_RANGE_ARRAY" structure in Windows 7 for example, in case the information published on [31] is correct. We mentioned such information to highlight that the proposed XP-7 hibernation file layout in [11] doesn't seems to be a standard for all Windows versions earlier than Windows 8.

[3] Papers [28] and [36] mentioned that the header cannot contains more than 255 entries in the field that was named "EntryCount" by [11]. They base their calculations on calculating the header length as 16 bytes and they consider each "_PO_MEMORY_RANGE_ARRAY_RANGE" length 16 bytes. Maximum Number of EntryCount in a table page = 4096 bytes – 16 bytes (header) = 4080 /16 = 255 Entries (16 bytes each).

range has a field for “StartPage” and another for “EndPage”. The StartPage” and “EndPage” fields define the range of physical memory pages stored in an “XPRESS set”.

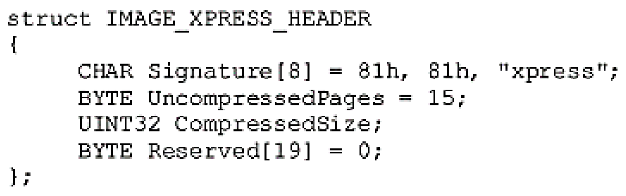
We mentioned earlier that the definition of a structure is modified depending on the Windows version. We notice that the definition of the structures starting with “_PO_MEMORY_RANGE_ARRAY”[1] for Windows Vista SP1 in [31], is not the same as the structure in [11]. To clearly understand these structures, we installed Windows 7 SP1 x64, and we did some quick checks on a test hibernation file extracted from the Windows 7 version[2]. We applied the definition of the “table page” defined in Figure 7 to this test hibernation file, and the definition seems to be valid for Windows 7 SP1 x64 version. Appendix 1 shows parts of the manual analysis that we applied on Windows 7 SP1 x64 hibernation file.

Each “table page” is followed by an “XPRESS set” of compression blocks. The compression block starts with the structure “_IMAGE_EXPRESS_HEADER” [11] – Figure 8 (named “IMAGE_XPRESS_HEADER” in [28] - Figure 9) and has 8 bytes of signature equals to “\x81\x81xpress”.



```
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;
```

Figure 8. “_IMAGE_EXPRESS_HEADER” structure definition [11].



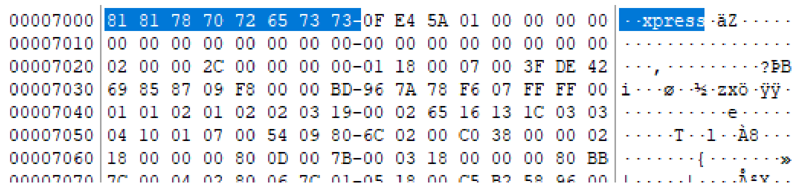
```
struct IMAGE_XPRESS_HEADER
{
     CHAR Signature[8] = 81h, 81h, "xpress";
     BYTE UncompressedPages = 15;
     UINT32 CompressedSize;
     BYTE Reserved[19] = 0;
};
```

Figure 9. “IMAGE_EXPRESS_HEADER” structure definition [28].

[1] Some papers replace these structures starting with “_PO_MEMORY_RANGE_ARRAY” with structures that starts with “MEMORY_TABLE” referencing this structure to P. Kleissner [36]. In Suiche documentations, these structures starts with “MEMORY_RANGE_ARRAY” [27].

[2] Unfortunately, it was not possible to run WinDbg on this version, due to technical limitations. The only way to confirm the structure definition was to perform a manual analysis on the file, as demonstrated in this section.

It is clear from Figures 8 and 9 that structures names are not coherent in the XP-7 hibernation file layout. The definition of the same structure varies, we may refer this variation to the dependency of field names and lengths on the Windows version. The references of Figures 8 and 9 did not clarify the Windows version of each structure. Figure 10 shows the "compression block" signature that we extracted from a Windows 7 SP1 x64 hibernation file.



```
00007000  81 81 78 70 72 65 73 73-0F E4 5A 01 00 00 00 00  ··xpress·äZ·····
00007010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ················
00007020  02 00 00 2C 00 00 00 00-01 18 00 07 00 3F DE 42  ···,·········?ÞB
00007030  69 85 87 09 F8 00 00 BD-96 7A 78 F6 07 FF FF 00  i···ø··½·zxö·ÿÿ·
00007040  01 01 02 01 02 02 03 19-00 02 65 16 13 1C 03 03  ··········e·····
00007050  04 10 01 07 00 54 09 80-6C 02 00 C0 38 00 00 02  ·····T··l··À8···
00007060  18 00 00 00 80 0D 00 7B-00 03 18 00 00 00 80 BB  ·······{·······»
00007070  7C 00 04 02 80 06 7C 01-05 18 00 C5 B2 58 96 00  |·····|····Å²X··
```

Figure 10. XPRESS "compression block" signature of Windows 7 SP1 x64.

The content of a compression block might be uncompressed or compressed using the XPRESS compression algorithm[32].

From the described structure of the legacy XP-7 hibernation file layout, we can see that this structure was vulnerable for recovering its content at any time. The file content was kept after restoring the file from hibernation, only the header (first 4096 bytes) was zeroed. Deleting the header seems to be a blocking point from recovering the legacy hibernation file, due to the missing details about the "FirstTablePage". However, our tests on Windows 7 SP1 x64 hibernation file shows that the missing header is not a problem. The "FirstTablePage" entry could be found by searching for the first XPRESS header signature in the file (\x81\x81xpress). Going backwards for 4096 bytes offset from the beginning of the first XPRESS header would lead to the "FirstTablePage". Once the FirstTablePage is found, it is possible to extract the "NextTable" entries as well as the "XPRESS sets" and restore the memory content saved in the hibernation file, even after its restoration.

2.3.5.2 Modern Hibernation file structure

A new hibernation file structure was used by Microsoft starting from Windows 8. To the Author's knowledge, the new structure was only documented by [11], and they named this structure the "Modern hibernation file structure". This section would describe the modern hibernation file layout as per [11]. Figure 11 shows the modern hibernation file layout.

Similar to the legacy hibernation file, the first page of a modern hibernation file contains the structure “PO_MEMORY_IMAGE”, while the content of the structure was modified, as many fields were added to it [11].

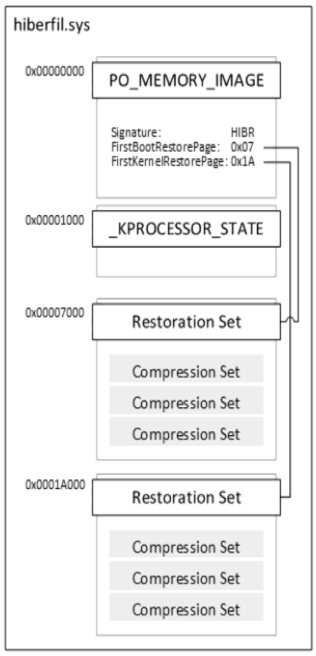


Figure 11. Modern Windows hibernation file layout [11].

The processor context has the same structure “_KPROCESSOR_STATE”, and it is always available at offset 0x1000 in the modern hibernation file layout [11].

The storage of the content of the physical memory in the modern hibernation file is the major change that was done to the new file structure. The “table pages” and “XPRESS sets” were replaced by “restoration sets” that contain many “compression sets” [11].

The modern hibernation file header has four valid signature values [11]. “HORM”[33] is a feature that enables the system to be resumed using the same hibernation file. This feature is supported by IoT Core Editions. “HORM” feature is supported by Windows 10 Enterprise and Education starting from Windows 1709, which was released after publishing [11]. For this reason, the HORM signature was not included in [11] research.

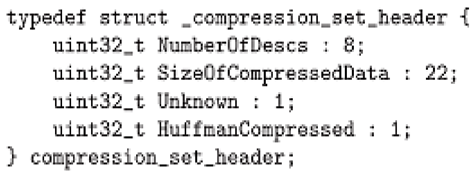
Once the system is resumed from hibernation, the “hiberfil.sys” signature is changed to “WAKE”, the rest of the header content is kept unmodified (First 4096 bytes), while all other data in the file is zeroed. Such behavior makes a restored modern hibernation file almost useless, once restored [11].

Table 4 describes the four modern hibernation file signatures.

Table 4. Modern hibernation file signatures value as defined by [11].

Signature Value	System State
HIBR	The system is in the hibernation state
RSTR	The system is actively being resumed from hibernation
WAKE	The system was resumed successfully from hibernation
HORM	The system has the feature Hibernate Once/Resume Many enabled

The modern hibernation file contains mainly two “restoration sets” related to “FirstBootRestorePage” and “FirstKernelRestorePage” entries. There is a potential third restoration set related to the” FirstSecureRestorePage” field, where the content always consisted of zeroes during [11] research. Each “restoration set” contains one or more “compression sets” (terminology proposed by [11]) with a maximum size of 64KB of data (4K physical memory pages). Since the structure of “restoration sets” is undocumented, the paper [11] proposed naming and terminologies. A “compression set” starts with a 32-bit little-endian “compression_set_header”, as described in Figure 12 and Table 5.



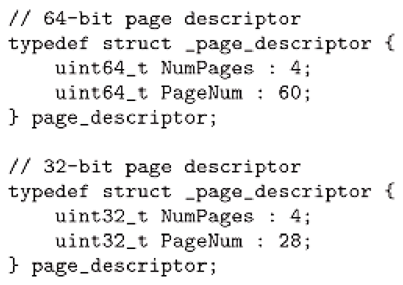
```
typedef struct _compression_set_header {
    uint32_t NumberOfDescs : 8;
    uint32_t SizeOfCompressedData : 22;
    uint32_t Unknown : 1;
    uint32_t HuffmanCompressed : 1;
} compression_set_header;
```

Figure 12. Definition of “compression set” header [11].

Table 5. Explanation of the “compression set” header’s fields as defined by [11].

Field name	Size in bits	Description
NumberOfDescs	8	The number of page descriptors that follow the header (0 < NumberOfDescs <= 16).
SizeOfCompressionData	22	the size (in bytes) of the compressed data that follows the descriptors.
HuffmanCompressed	1	When its value equals 1, the data is compressed using the “LZ77 + Huffman XPRESS” algorithm. When its value equals 0, The data is compressed using the “Plain LZ77 XPRESS” algorithm.

A “NumberOfDescs” of “page descriptors” follow each “compression_set_header”. Figure 13 and Table 6 define the “page descriptor”. A “page descriptor” is composed of 64 bits (8bytes) for x64 Windows versions, while for 32-bit Windows versions, the “page descriptor” length is 4 bytes. Each “page descriptor” defines a set of 4KB “contiguous pages” [11].



```
// 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;
```

Figure 13. Page descriptors definitions for 32 and 64 Windows versions [11].

Table 6. Explanation of the page descriptors fields as defined by [11].

NumPages	Number of contiguous pages in the contiguous page set = “NumPages” + 1
PageNum	physical page address of the first page in the contiguous page = “PageNum” * 0x1000, to obtain the address offset in hex decimal.

The sum of the number of contiguous pages in each contiguous page set determines the total number of pages in the compression set. An example of manual investigations of a compression set would be demonstrated in section 4.3.2.

2.3.6 Hibernation file security

Windows system provides its user with an option of encrypting the hard drive volumes using Bitlocker, while pen testers proved that the Bitlocker encryption key could be compromised during the sleep mode as the machine would be vulnerable to a cold boot attack. For better security for the memory data, pen tests recommend using the hibernation option instead of the sleep power state option, besides the hard disk encryption [34].

More than 10 years ago, Suiche claimed that the XP-7 hibernation file could be modified to bypass Windows authentication using an exploit that he created [12], [25]. Unfortunately, during Suiche live presentation of this attack, it did not work. We were not able to find any demonstration for this hibernation file authentication bypass attack online. While the Sandman framework documents mention that the framework gave users the possibility of writing data in the hibernation file [27], on the other hand, Kleissner provided presentations about the proof of concept of another type of Windows XP-7 hibernation file attack [35], [36], [37]. Kleissner created bootkit (a bootable rootkit) named Stoned, that can inject some code into the memory while booting the Windows system. He discussed the possibility of injecting a code into the hibernation file to resume Windows using a modified hibernation file. As per Kleissner's presentations, the code would be injected into the memory, as far as the checksum is set to zero. In such a case, no checks would be done by Windows to confirm that the file was not modified. Kleissner also clarified that such a way of attack could load any unsigned code into the kernel, and he has reported this issue to Microsoft by the time. As per Kleissner in [37], no real attack was done against the hibernation file, it was just a proof of concept. It is not clear if the file is still vulnerable to such kinds of attacks or not. We did not find any academic researches or online demonstrations discussing similar kinds of attacks, except the presentations done by Suiche and Kleissner during the period from 2007 to 2010. It is important to highlight that Kleissner's presentation about Stoned bootkit [35] stated clearly that his tests were successful only on the legacy BIOS.

2.4 Related work

The white paper [27] published by Suiche in 2008, introduced the forensics value of the hibernation feature. The paper considered the hibernation file as an efficient way to save physical memory content, without the need for external tools that might perform changes on the system. The paper discussed two points related to the hibernation files. The first discussed point was the use of the hibernation file in defensive computing, and read the undocumented structure of the hibernation file. The second discussed point was the offensive usage of this file and the possibility of writing data in a hibernation file. As per the paper, there are two processes related to hibernation. The Windows kernel (ntoskrnl.exe) process is responsible for hibernation file creation, and writing the physical memory dump into it, when the hibernation is required. The second process is OSLoader (osloader.exe), which read the content of the hibernation file and load the data into the physical memory. Suiche provided in the paper Windows XP-7 hibernation file structure and introduced the SandMan framework, which aimed to read and write the hibernation file.

The research [10] published in 2015, covered a wide range of topics related to Windows XP-7 hibernation file. The study aimed to demonstrate the importance of the hibernation file in memory forensics. The research was very informative. However, it did not include new details about the file structure and based all its information on other references. Despite mentioning some analysis tools in the study, no tests were done on those tools. The study could be considered as a useful literature review for the Windows XP-7 hibernation file.

The paper [38] was released In 2016. This paper was one of the first papers claiming discussing Windows 10 hibernation file values. The research stated that the hibernation file is advantageous even when a memory image is available. The study explained such an advantage as the file might contain data from the past, especially in the case of scheduled system backups. Multiple backup versions of a stored hibernation file might be utilized as various memory snapshots for different times in the past. This behavior is advantageous in malware analysis, sessions usernames, and passwords as well as disk encryption keys, while it was not mentioned in the paper that this functionality is not applicable on a Windows 10 hibernation file. By the time the research was published, there was no clear vision about the structure change of the hibernation file. the study described the XP-7 hibernation file structure and functionalities, as the role of hibernation file in memory forensics of Windows 10. The modifications of the hibernation file structure were not even highlighted in this paper, which gives misleading information to the reader. The study context leads to conclude that Windows 10 hibernation file is using the same Windows XP-7 hibernation file structure. The study also discussed some tools that are used in hibernation file analysis and stated their options, while some of these options are not supporting Windows 10 hibernation file, which was not clarified in the paper. An example of misleading information in that study is stating that MoonSols Windows Memory Toolkit can convert hiberfil.sys into a crash dump format, which does not apply for a Windows 10 hibernation file. Also, our tests show that when applying the Volatility profile “ Win10x64_17763”, for example, on a” hiberfil.sys”, the plugin “hibinfo” would state that this profile is an Incompatible profile. Contrarily, [38] stated that the plugin could be used for Windows 10 hibernation file analysis. The paper also stated that Volatility supports Windows 10 hibernation file analysis in its native format, while it is published on the GitHub wiki of volatility foundation that “*Volatility currently does not support hibernation files from Windows 8 and newer* *machines. Support for modern hibernation files will be released sometime in the future"* [39].

The paper [11] was published in 2017. To our knowledge, this paper is the first paper to study the new structure of the hibernation file. The study also clarified the impacts of the file layout modifications on memory forensics. We have referred to paper [11] in many sections of this research, as we find it very informative. The paper first discussed the legacy hibernation file layout, named it as "XP-7 hibernation file format", and provided its file structure in detail, which was mentioned earlier in the file structure section 2.3.5.1. Hence, the study introduced the hibernation file layout used starting from Windows 8, and named the new layout as "the modern hibernation file". As the paper was released by BlackLight, verifications of the newly revealed file format were performed by decompressing different hibernation files using BlackLight 2016R3, then compared BlackLight output to Hibr2Bin output. The results of [11] showed that the modern hibernation files converted by Hibr2Bin were identical to the files converted by BlackLight, except for Windows 10 version 1607. Hibr2Bin converted hibernation files of Windows 10 version 1607 were missing the "KernelRestorePages". The results of the research also showed that the behavior changes introduced starting from Windows 8, has reduced the lifetime of valuable artifacts available in a modern Windows hibernation file. The artifacts' lifetime of a modern hibernation file is limited to the time between hibernation and the first power-on. The study explained that hibernation file content is zeroed once the Windows is resumed, and only the file header is preserved, which makes collecting a hibernation file from a running machine almost useless. The results of [11] mentioned that shutting down the Windows8+ using shutdown /s, or pulling the plug, would not leave a hibernation file. While shutting down the machine using the shutdown GUI button would contain a hibernation file that includes the Kernel session only. As per the research, powering down a machine using the "Hibernate" option preserve the largest amount of data.

The paper [13], released in 2017, stated that memory forensics could reveal many volatile artifacts, such as the list of running processes, network connections, chat messages, and encryption keys. Such valuable evidence could be lost in case of following the traditional "pull the plug" forensics technique. The research discussed many problems impacting memory acquisition and analysis. Also, future directions for each of the discussed problems were proposed. One of the discussed problems was the modification of the hibernation file format and function based on [11] research, which makes the analysis of a hibernation file captured from a running system almost useless. Paper [13] proposed to exclude the hibernation file from live acquisition, to speed up the process due to the large size of the hibernation file. The research also recommended adding warning messages to the memory analysis tools when users tend to analyze hibernation files. One of the interesting problems discussed in [13] was the use of swapfile.sys by Windows 10, which is used for swapping out metro applications, instead of the traditional pagefile.sys. The paper proposed further researches to understand the impact of such a new feature on memory forensics. The study covered many other issues related to memory acquisition as well as memory analysis, while they are considered out of the scope of this research.

2.5 Introduction to analysis Tools

This section introduces the main tools used during the research and provides a briefing about each tool. For each tool, we would mention the used versions, and we would highlight some notes about the tool. For instance, we found many contradictions in volatility and Passware documentations that would be stated in this section. The section also includes challenges faced during the setup of some free tools. For commercial tools, we would clarify how we knew that the tool could analyses Windows 10 hibernation files, how the trial version was requested, what are the limitations of the trial version, and we would also clarify any special configuration options that were applied during the testing period.

2.5.1 WinDbg

The debugging tool for Windows (WinDbg) is a free tool that makes part of Windows of the Software Development Kit (SDK) [29] published by Microsoft. WinDbg supports kernel mode and user mode, it uses Visual Studio debug symbol formats and helps in debugging Windows internal structures. WinDbg was published by Microsoft to help developers in creating Windows applications. The application was used in [11] to extract names and locations of known structures' fields.

The Debugging tool supports Kernel mode debugging using different types of connections between a host computer (a computer that runs the debug) and a target computer (computer being debugged) [40]. Examples of media used for establishing connections required for Windows kernel-debug are serial cables, USB 2.0, and 3.0

cables. Windows debugger also allows local kernel-debug, to enable kernel-debug using a single computer. The local kernel-debug way was used during this study, as all other types of connections are not supported by the analysis laptops. The local kernel-debug was enabled on each Windows version as per instructions provided in [41]. WinDbg version 10.0.18362.1 used for that research.

2.5.2 FTK Imager

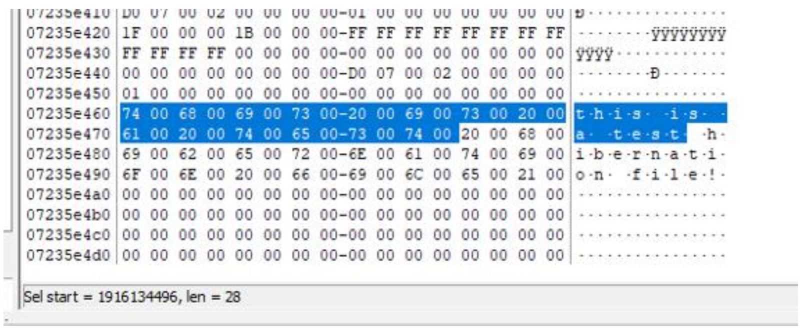
FTK imager is a free tool released by "AccessData". The tool is used for forensics imaging and data preview [42]. This tool is an excellent support for manual analysis, and it was used to check the content of the hibernation files, calculate their hashes, and capture memory and disk images from the test laptop when required.

2.5.3 Hibr2Bin

The credit of decompressing the legacy hibernation file, and highlighting its forensics values goes to Matthieu Suiche, who was the first to present and publish the hibernation file structure back to 2007, as a part of his open-source SandMan project [43]. In 2008, he stopped open-sourcing his projects after the theft of the source code by a company [44]. In 2016, Matthieu announced the return of his toolkit that included DumpIt and Hibr2Bin CLI utilities [45]. Comae toolkit includes other tools like Hibr2Dmp, which could convert a legacy hibernation file to a Microsoft crash dump format, while Hibr2Dmp does not support Windows 10 hibernation files format. The utilities are available to download from Comae website for free, after registration [46].

Following the release of [11] in 2017, Suiche announced opening the source of Hibr2Bin on GitHub [44], the compiled version is still downloadable for free from Comae website. During our research Hibr2Bin role was not only as a conversion tool, but we also took advantage of the availability of the source code on GitHub and used the code to confirm our understanding of the hibernation file structure. the open-source code of Hibr2bin helped us to spot the reason of differences between the resulted output file of Hibr2Bin and Hibernation Recon.

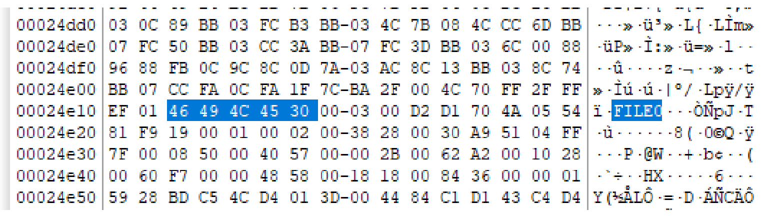
During our first studies of the feasibility of our research, we did a proof of concept test, to confirm that it is possible to extract some data from a hibernation file extracted from recent Windows 10 versions. During that time we did not have any visibility about what kind of tools support processing hibernation files extracted from the latest Windows 10 versions. Tests were done on hibernation files extracted from x64 Windows 10 Pro, version 10.0.17763 built 17763, RAM size 4GB. For the proof of concept, we created a WordPad draft document in which it was written: “This is a test hibernation file!”. Then, the machine was hibernated using the hibernation power option in Windows start menu, after enabling this option. The machine was then booted using another Windows 10, installed on a bootable drive, and the hibernation file of our target machine was extracted using FTK imager that run as administrator. We opened the extracted hibernation file using FTK imager. The file signature (the first 4 bytes of the file) was "HIBR", and when searching for the strings “this is a test” in the extracted hiberfil.sys, the strings were not found. The file was then converted using Hibr2Bin included in Comae-Toolkit-3.0.20191016.4. The output of Hibr2Bin is a single binary file. A per Figure 14, when searching for the strings “this is a test” in the binary file converted by Hibr2Bin, we were able to find our test file’s content. The result of this proof of concept confirmed that Hibr2Bin could convert the hibernation file to a format that we could process later by other programs, as the converted file content is not compressed anymore.



```
07235e410 D0 07 00 02 00 00 00 00-01 00 00 00 00 00 00 00 Đ···············
07235e420 1F 00 00 00 1B 00 00 00-FF FF FF FF FF FF FF FF ········ÿÿÿÿÿÿÿÿ
07235e430 FF FF FF FF 00 00 00 00-00 00 00 00 00 00 00 00 ÿÿÿÿ············
07235e440 00 00 00 00 00 00 00 00-D0 07 00 02 00 00 00 00 ········Đ·······
07235e450 01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
07235e460 74 00 68 00 69 00 73 00-20 00 69 00 73 00 20 00 t·h·i·s· ·i·s· ·
07235e470 61 00 20 00 74 00 65 00-73 00 74 00 20 00 68 00 a· ·t·e·s·t· ·h·
07235e480 69 00 62 00 65 00 72 00-6E 00 61 00 74 00 69 00 i·b·e·r·n·a·t·i·
07235e490 6F 00 6E 00 20 00 66 00-69 00 6C 00 65 00 21 00 o·n· ·f·i·l·e·!·
07235e4a0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
07235e4b0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
07235e4c0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
07235e4d0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················

Sel start = 1916134496, len = 28
```

Figure 14. Search for known strings in a converted binary file using Hibr2Bin.

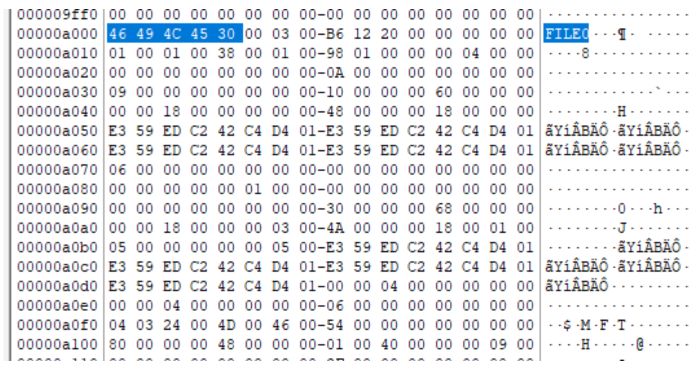
From a quick analysis of this proof of concept hibernation file, we noticed that the header of a converted file using Hibr2Bin was replace by zeros, from this information we understand that a converted hibernation file does not contain the exact information available in the hiberfil.sys. Also, we noticed that the converted file contains an example of known file structures, like $MFT entries (searched for “FILE0” string), and registry fragments (searched for “hbin” string). While the original hiberfil.sys also contains the strings (“FILE0” and “hbin”), the entries layout seems to be compressed and doesn’t have the known layout for these files type. Figure 15 shows the result of searching for an $MFT entry in hiberfil.sys, while Figure 16 shows the result of a search for the same string in a hibernation file that was covered by Hibr2bin.



```
00024dd0 03 0C 89 BB 03 FC B3 BB-03 4C 7B 08 4C CC 6D BB ···»·ü³»·L{·LÌm»
00024de0 07 FC 50 BB 03 CC 3A BB-07 FC 3D BB 03 6C 00 88 ·üP»·Ì:»·ü=»·l··
00024df0 96 88 FB 0C 9C 8C 0D 7A-03 AC 8C 13 BB 03 8C 74 ··û····z·¬··»··t
00024e00 BB 07 CC FA 0C FA 1F 7C-BA 2F 00 4C 70 FF 2F FF »·Ìú·ú·|º/·Lpÿ/ÿ
00024e10 EF 01 46 49 4C 45 30 00-03 00 D2 D1 70 4A 05 54 ï·FILE0···ÒÑpJ·T
00024e20 81 F9 19 00 01 00 02 00-38 28 00 30 A9 51 04 FF ·ù······8(·0©Q·ÿ
00024e30 7F 00 08 50 00 40 57 00-00 2B 00 62 A2 00 10 28 ···P·@W··+·b¢··(
00024e40 00 60 F7 00 00 48 58 00-18 18 00 84 36 00 00 01 ·`÷··HX·····6···
00024e50 59 28 BD C5 4C D4 01 3D-00 44 84 C1 D1 43 C4 D4 Y(½ÅLÔ·=·D·ÁÑCÄÔ
```

Figure 15. Searching for $MFT file entry in the proof of concept hiberfil.sys.

The result shown in Figure 16 is promising, as the data has a valid $MFT entry layout that could be carved manually. While in such case we would carve separate segments of the $MFT file. The segments length would equal to a hibernation file page size. This part of the proof of concept aims to prove that we can find known structure files in a decompressed hibernation file, that could be used to carve some known files types. While this research would not include carving any data manually. The examples of "hbin" entries found in compressed and decompressed hibernation file are similar to the screenshots available in [47].



```
000009ff0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00000a000 46 49 4C 45 30 00 03 00-B6 12 20 00 00 00 00 00 FILE0···¶· ·····
00000a010 01 00 01 00 38 00 01 00-98 01 00 00 00 04 00 00 ····8···········
00000a020 00 00 00 00 00 00 00 00-0A 00 00 00 00 00 00 00 ················
00000a030 09 00 00 00 00 00 00 00-10 00 00 00 60 00 00 00 ············`···
00000a040 00 00 18 00 00 00 00 00-48 00 00 00 18 00 00 00 ········H·······
00000a050 E3 59 ED C2 42 C4 D4 01-E3 59 ED C2 42 C4 D4 01 ãYíÂBÄÔ·ãYíÂBÄÔ·
00000a060 E3 59 ED C2 42 C4 D4 01-E3 59 ED C2 42 C4 D4 01 ãYíÂBÄÔ·ãYíÂBÄÔ·
00000a070 06 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00000a080 00 00 00 00 00 01 00 00-00 00 00 00 00 00 00 00 ················
00000a090 00 00 00 00 00 00 00 00-30 00 00 00 68 00 00 00 ········0···h···
00000a0a0 00 00 18 00 00 00 03 00-4A 00 00 00 18 00 01 00 ········J·······
00000a0b0 05 00 00 00 00 00 05 00-E3 59 ED C2 42 C4 D4 01 ········ãYíÂBÄÔ·
00000a0c0 E3 59 ED C2 42 C4 D4 01-E3 59 ED C2 42 C4 D4 01 ãYíÂBÄÔ·ãYíÂBÄÔ·
00000a0d0 E3 59 ED C2 42 C4 D4 01-00 00 04 00 00 00 00 00 ãYíÂBÄÔ·········
00000a0e0 00 00 04 00 00 00 00 00-06 00 00 00 00 00 00 00 ················
00000a0f0 04 03 24 00 4D 00 46 00-54 00 00 00 00 00 00 00 ··$·M·F·T·······
00000a100 80 00 00 00 48 00 00 00-01 00 40 00 00 00 09 00 ····H·····@·····
```

Figure 16. Search for $MFT entry in decompressed hibernation file.

We proceeded the rest of the tests done during this research, using Hibr2Bin included in Comae-Toolkit-3.0.20200224.1.

2.5.4 Hibernation Recon

Hibernation Recon is a tool produced by Arsenal Recon. The tool has a GUI and CLI interfaces. As per the product FAQ page [48], the software can export a ".bin" file that contains Active memory decompressed & reconstructed from a hibernation file. Hibernation Recon also extracts decompressed and raw Slack data in a ".bin" file. The software can also extract NTFS Metadata in two separates .csv files. Similar to Hibr2Bin, the output binary file of Hibernation Recon requires processing by a memory forensics toolkit to extract artifacts from the decompressed hibernation file. Both Hibernation

Recon and Hibr2Bin, do not provide any analysis options. Hibernation Recon offers two running modes: Free (When Hibernation Recon runs without a license) and Professional modes (Paid service). As per [48], Hibernation Recon does not currently support processing BitLocker and TPM-impacted hibernation files. Arsenal Recon mentioned on [48] that the hibernation files content is zeroed out when the clear page file setting is enabled or when Windows 8+ is resumed, while for our understanding, in win XP-7 case the hibernation file was zeroed out only in case of disabling the hibernation option [49].

Appendix 2 shows the expected output files when processing a hiberfil.sys using Hibernation Recon, as listed on [48]. We eliminated the file names related to the legacy format of the hibernation file, as they are out of the scope of the thesis.

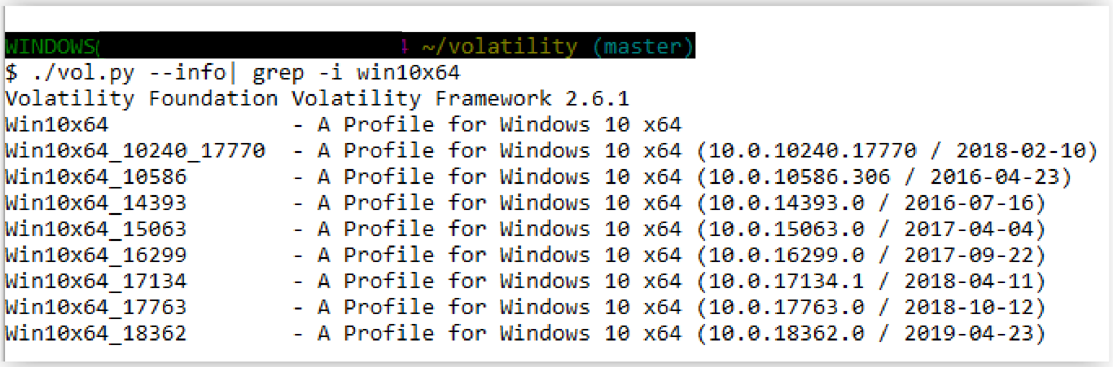
The tests done during this research used the Hibernation-Recon-v1.2.0.70_Beta version, with Free Mode. We received a full version testing license during our testing period, which extracts more data as clarified in Table 10.

2.5.5 Volatility

Volatility was first released in 2007. The software is an open framework for advanced volatile memory analysis. Volatility is the most widely used memory forensics software in the world. It is a free open source CLI platform that supports memory analysis of Windows systems, Mac OS, and Linux, the Volatility foundation is open for the contribution of developers [50]. Volatility does not support the analysis of modern hibernation files in its original format.

When processing a memory image with Volatility, it is required to choose the correct operating system profile, to obtain an accurate output of Volatility analysis. The analysis of a memory image extracted from a Windows version 1809, for example, using the generic Windows 10 x64 profile (win10x64) on Volatility would not show a correct output. Volatility Foundation website [50] shows that the latest official version is Volatility 2.6 (release 2016) improve the support of Windows 10. As per the website, this version supports Windows 10 including built 14393.447 (version 1607). While the GitHub source page[39] mention that Volatility supports Windows 10 profiles up to OS built 10.0.18362 (version 1903). On the other hand, the wiki page of Volatility's GitHub page mentions that volatility only supports analyzing memories of three versions for Windows 10 x64 (1511, 1607, 1703). Such non-coherence in the published information about the list of Windows 10 profiles that are currently supported by volatility, might cause massive confusion for users with limited experience with Volatility.

Tests done during that research shows that available Volatility 2.6 profiles vary depending on the way the program was installed. The user should be aware of such profiles availably incoherence to achieve maximum usability of Volatility. For example, downloading the Windows executable of Volatility 2.6 from [51] has outdated Windows 10 x64 profiles (supports Windows versions 1511 and 1607, in plus of the generic Windows 10 profile). On the other hand, Volatility 2.6 version that is automatically included in Kali Linux release 2019.4(released 20th of November 2019), supports the generic Windows 10 x64 profile plus six other versions of Windows 10 (up to version 17134). Figure 17 shows the available profile on Volatility GitHub source code [39], as on the 19th of January 2020. The GitHub code includes Windows 10 x64 profiles up to version 1903 (released in the first half of 2019). Volatility currently does not support only a single version of Windows 10 x64 (version 1909 that was released November 2019).



```
WINDOWS([redacted] ~/volatility (master)
$ ./vol.py --info| grep -i win10x64
Volatility Foundation Volatility Framework 2.6.1
Win10x64               - A Profile for Windows 10 x64
Win10x64_10240_17770   - A Profile for Windows 10 x64 (10.0.10240.17770 / 2018-02-10)
Win10x64_10586         - A Profile for Windows 10 x64 (10.0.10586.306 / 2016-04-23)
Win10x64_14393         - A Profile for Windows 10 x64 (10.0.14393.0 / 2016-07-16)
Win10x64_15063         - A Profile for Windows 10 x64 (10.0.15063.0 / 2017-04-04)
Win10x64_16299         - A Profile for Windows 10 x64 (10.0.16299.0 / 2017-09-22)
Win10x64_17134         - A Profile for Windows 10 x64 (10.0.17134.1 / 2018-04-11)
Win10x64_17763         - A Profile for Windows 10 x64 (10.0.17763.0 / 2018-10-12)
Win10x64_18362         - A Profile for Windows 10 x64 (10.0.18362.0 / 2019-04-23)
```

Figure 17. Volatility - Windows 10 x64 profiles using the GitHub source code.

For this research, Volatility GitHub source code was installed on a Windows 10 [52], to run the last updates of Volatility v2.6 version on the analysis laptop. Volatility released a Beta version of Volatility v3.0 in October 2019, and they plan to officially release this version in August 2020 [53]. The new release would not be required to mention an operating system profile for each command.

2.5.6 Rekall

The Rekall Forensic and Incident Response Framework is an open collection of tools to extract and analyze digital artifacts computer systems [54]. Rekall source code was implemented to be a part of volatility, and then a decision was made to treat the project as a stand-alone project and call it Rekall. The Framework has the capability of memory acquisition and live forensics analysis. The last release is 1.7.2 RC1, released in December 2017. We installed the Windows installer of Rekall using the last version available on GitHub. Rekall's GitHub page mention that it can support the analysis of a 64bit Windows 10 memory image. No clear information found on Rekall official pages [54], [55], [56] about the support for hibernation file analysis in its original form. We found in the GitHub code a python file named "hibernate.py" in the path "rekall/rekall-core/rekall/plugins/addrspaces/hibernate.py", while we were not able to find any plugin with the name of "hibernate" or includes "hiber" when searching in the list of Rekall plugins [56]. Searching for the keywords "hibernate", "hiberfil.sys", "hiber" in [55] would only return a single result pointing to [57], where it is mentioned under "Address Spaces" section, *"Rekall uses an address space to abstract the handling of different images and formats and therefore allow plugins to support multiple kinds of input images (or indeed live memory) easily. An address space is an object which can satisfy a read request for data at a certain offset." "There are a number of simple address spaces which simply provide access to a specific data source:*

FileAddressSpace - Simply opens a file and satisfies read requests from it.

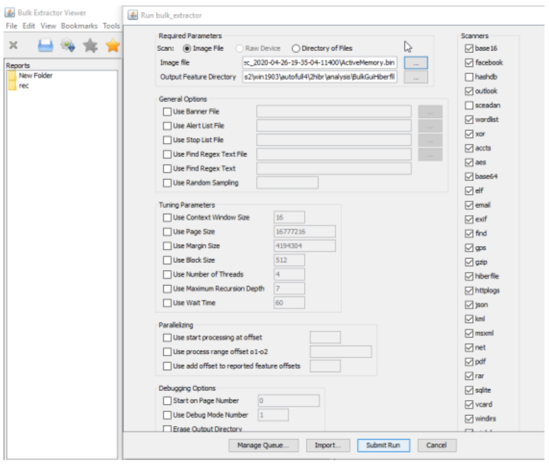
WindowsHiberFileSpace - Supports Windows hibernation files" [57].

From this description, we understand that recall could proceed with a hibernation file directly, while the definitions mentioned in "hibernate.py" seem to be related to old versions of the Windows profile.

During our searches about Rekall, we found that in July 2019, FireEye published an update for volatility and Rekall to support memory compression up to Windows 10 version 1809 x64 [58], [59]. FireEye mentioned that starting from Windows 8.1, Microsoft introduced memory compression to its operating systems, which turned into a limitation for memory forensics tools. They demonstrated in [58] that Rekall and volatility were not able to read compressed memory pages, which makes the memory analysis incomplete. It is not clear if FireEye update might help our research or not, as Rekall profiles seem to be old profiles, for this reason, we decided to test the FireEye updates. We did our tests using two versions of Rekall. The first tested Rekall version was Release 1.7.2 RC1 (released on December 2017) using the Windows installer available on GitHub. We also installed FireEye GitHub version [59] on Kali Linux release 2020.1 virtual machines. Following the installation instructions indicated on [59] [54] to install Rekall on Kali Linux produced errors and Rekall did not function till we downgraded pyaff4 to version 0.26.post6 and future to version 0.16.0 as per [60]. The tests done on both tested versions shows that Rekall produce errors even when processing a memory image of Windows 10 version 1809, for this reason, we decided to exclude Rekall from the next sections, and document our tests results in Appendix 3.

2.5.7 Bulk_extractor

Bulk_extractor is a *"high-performance carving and feature extraction tool that uses bulk data analysis to allow the triage and rapid exploitation of digital media"*[61]. Bulk_extractor is a free tool available on GitHub [62], the tool scan the entire content of the analyzed file to extract different types of artifacts that are categorized per type, which might help investigators in quick triage. Bulk_extractor program can extract different types of evidence, including email addresses, URL, AES keys, SQLite databases, telephone numbers, search terms. The tool has a CLI interface and a GUI interface named Bulk Extractor Viewer. A report could be generated using the GUI interface and contains different types of information. Figure 18 shows the applied configuration that was used during the tests.



Required Parameters
Scan: [✓ Image File] Raw Device Directory of Files
Image file: ec_2020-04-26-19-35-04-11400\ActiveMemory.bin
Output Feature Directory: s2\win1903\autofull4\2hibr\analysis\BulkGuiHiberfil

General Options
☐ Use Banner File
☐ Use Alert List File
☐ Use Stop List File
☐ Use Find Regex Text File
☐ Use Find Regex Text
☐ Use Random Sampling

Tuning Parameters
☐ Use Context Window Size 16
☐ Use Page Size 16777216
☐ Use Margin Size 4194304
☐ Use Block Size 512
☐ Use Number of Threads 4
☐ Use Maximum Recursion Depth 7
☐ Use Wait Time 60

Parallelizing
☐ Use start processing at offset
☐ Use process range offset o1-o2
☐ Use add offset to reported feature offsets

Debugging Options
☐ Start on Page Number 0
☐ Use Debug Mode Number 1
☐ Erase Output Directory

Scanners
✓ base16
✓ facebook
☐ hashdb
✓ outlook
☐ sceadan
✓ wordlist
✓ xor
✓ accts
✓ aes
✓ base64
✓ elf
✓ email
✓ exif
✓ find
✓ gps
✓ gzip
✓ hiberfile
✓ httplogs
✓ json
✓ kml
✓ msxml
✓ net
✓ pdf
✓ rar
✓ sqlite
✓ vcard
✓ windirs

Figure 18. Bulk Extractor Viewer options.

Bulk_extractor was not included in our primary test plan, and we decided to include it after reading [63], while we did not have enough time to deeply analyze the tool outputs using the test cases. Our tests were done on Bulk_extractor version 1.6.0-dev.

2.5.8 Passware Kit Forensic

Passware Kit Forensic is a commercial GUI software that describes itself as a solution that reports all password-protected items on a computer and decrypts them. Passware list under the "Live memory analysis" section on their official page, that they "*Analyze live memory images, hibernation files and extracts encryption keys for FileVault2, TrueCrypt, VeraCrypt, BitLocker, logins for Windows & Mac accounts from memory images and hibernation files*" [64]. While it is not mentioned in Passware documentations which Windows versions are currently supporting hibernation file analysis. Passware is installed on Windows systems only. We downloaded the free demo version 2020.1.3, 64-bit Release 20200226.1351 from Passware official website. The demo version allows each type of attack to work for up to 1 minute, and it is showing the first 3 characters of the recovered password [65].

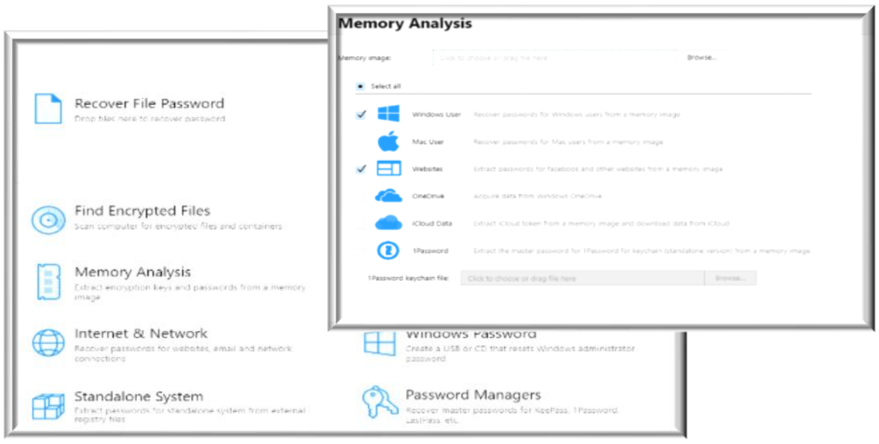
We knew by Passware, by searching online for tools that claim to process the hibernation file. As we found a topic under the name of "How to extract Windows login passwords from hibernation file or memory image instantly" on Passware official website [66].While this topic seems to be outdated[1]. Another topic with the title "How to extract website passwords from hibernation file or memory image" [67] for the same author, was also found on Passware support page. This second topic did not mention the supported Windows version for such a feature and claimed that the "*Passware Kit recovers passwords for Facebook, Gmail, and other websites by analyzing a memory image or a system hibernation file"* [67]. [2]

The help page of version 2020.1.3, in "Live Memory Analysis" section, mentions that Passware can extract the following types of passwords from memory images or a hibernation files: Passwords for Windows and Mac users; Passwords for websites; Authentication data for Apple iCloud and OneDrive; Master passwords for 1Password

[1] the topic has many pictures that are not shown anymore, while the same topic has two archives since 2017. Archives could be found at https://web.archive.org/web/2017*/https://support.passware.com/hc/en-us/articles/221742428-How-to-extract-Windows-login-passwords-from-hibernation-file-or-memory-image-instantly

[2] The instructions included in [67] requires selecting the option "Analyze Memory and Decrypt Hard Disk", while this option is not available anymore in the current software version. On the other hand, the print-screens included in the topic seems to be relevant to Passware latest versions.

manager and encryption keys for BitLocker, TrueCrypt, PGP, FileVault2, and other FDE. From Passware start page, when clicking on the memory analysis option, user can add a file, the tool display by default raw image files with the extensions (.DD, .IMG, and .BIN), while it gives the option to add any file extension.



Recover File Password
Drop files here to recover password

Find Encrypted Files
Scan computer for encrypted files and containers

Memory Analysis
Extract encryption keys and passwords from a memory image

Internet & Network
Recover passwords for websites, email and network connections

Standalone System
Extract passwords for standalone system from external registry files

Windows Password
Create a USB or CD that resets Windows administrator password

Password Managers
Recover master passwords for KeePass, 1Password, LastPass, etc.

Memory Analysis

Memory image: Click to choose or drag file here Browse...

● Select all

✓ Windows User — Recover passwords for Windows users from a memory image
Mac User — Recover passwords for Mac users from a memory image
✓ Websites — Extract passwords for facebook and other websites from a memory image
OneDrive — Acquire data from Windows OneDrive
iCloud Data — Extract iCloud token from a memory image and download data from iCloud
1Password — Extract the master password for 1Password for keychain (standalone version) from a memory image

1Password keychain file: Click to choose or drag file here Browse...

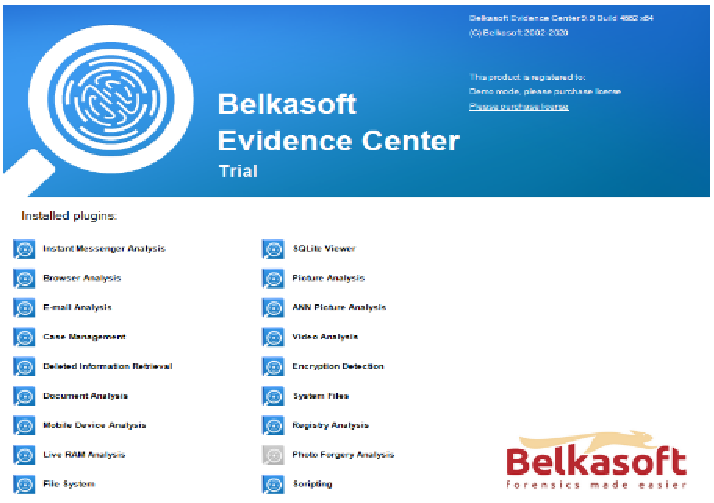
Figure 19. Passware Memory Analysis options.

Figure 19 shows the chosen options that were used during our tests. The Chosen options where: **Windows User** (Recover passwords for Windows users from a memory image), and **Websites** (Extract passwords for Facebook and other websites from a memory image). We added the option **OneDrive** (Acquire data from Windows OneDrive) for a single hibernation file extracted from Windows 10 version 1903, to test such functionality. The tool supports adding additional dictionaries using the **Dictionary Manager** option, while this option was not tested as it is out of the thesis scope.

2.5.9 Belkasoft Evidence center

Belkasoft Evidence Center [68] is a commercial forensics solution that helps investigators to acquire and analyze digital evidence. BEC is a GUI software that can directly analyze hibernation files. Analysis of "pagefile.sys" and live RAM dumps of devices running Windows, Linux, and Android, are also supported by the BEC. A trial request for BEC software was submitted from Belkasoft website. Our tests were done on the Belkasoft Evidence Center 9.9 Build 4662 x64. The trial version has full functionality of artifacts recovery and extraction, while a created report contains 50% of randomly chosen artifacts. The trial version does not support decryption functionality, and photo forgery detection, it does not also support other functionalities that will not impact our tests [69].

Once we open the tool, we can see the currently installed plugins. From plugins names, it is shown that the Evidence Center can detect encryption, analyze E-mails, browser information, documents, video, instant messenger, registry and it has many other capabilities as shown in Figure 20.



Belkasoft Evidence Center 5.3 Build 4882 x64
(C) Belkasoft 2002-2020

This product is registered to:
Demo mode, please purchase license
Please purchase license

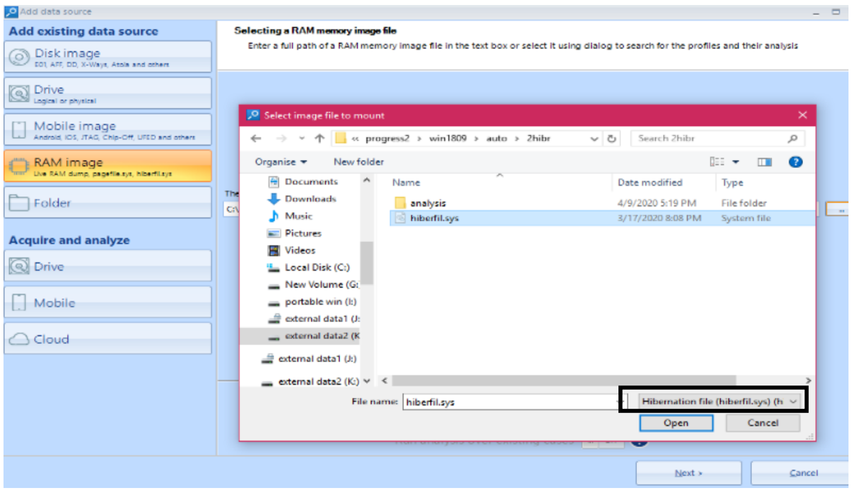
Belkasoft
Evidence Center
Trial

Installed plugins:

Instant Messenger Analysis	SQLite Viewer
Browser Analysis	Picture Analysis
E-mail Analysis	ANN Picture Analysis
Case Management	Video Analysis
Deleted Information Retrieval	Encryption Detection
Document Analysis	System Files
Mobile Device Analysis	Registry Analysis
Live RAM Analysis	Photo Forgery Analysis
File System	Scripting

Figure 20. Belkasoft Evidence Center - list of installed plugins.

A hibernation file could be analyzed by BEC using the “RAM image” option, as per Figure 21. The tool supports the analysis of pagefile.sys, and there is a possibility to add more than one data sources to the same case. BEC does not require a selection of Windows version profiles.



Add data source

Selecting a RAM memory image file
Enter a full path of a RAM memory image file in the text box or select it using dialog to search for the profiles and their analysis

Add existing data source
- Disk image — E01, AFF, DD, X-Ways, Atola and others
- Drive — Logical or physical
- Mobile image — Android, iOS, JTAG, Chip-Off, UFED and others
- RAM image — Live RAM dump, pagefile.sys, hiberfil.sys
- Folder

Acquire and analyze
- Drive
- Mobile
- Cloud

Select image file to mount

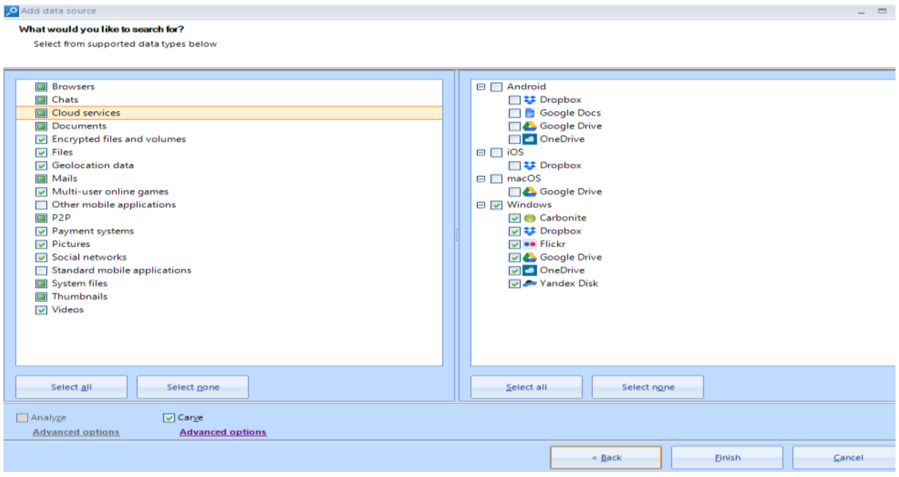
« progress2 › win1809 › auto › 2hibr

Name	Date modified	Type
analysis	4/9/2020 5:19 PM	File folder
hiberfil.sys	3/17/2020 8:08 PM	System file

File name: hiberfil.sys
Hibernation file (hiberfil.sys) (h

Figure 21. Belkasoft Evidence Center - loading a Hibernation file.

Once the file is selected, BEC opens a menu include the list of supported artifacts options. User can choose the target artifacts type, that BEC should search for in the hibernation file, as shown in Figure 22.


Add data source

What would you like to search for?

Select from supported data types below

- ▨ Browsers
- ▨ Chats
- ▨ Cloud services
- ▨ Documents
- ✓ Encrypted files and volumes
- ✓ Files
- ✓ Geolocation data
- ▨ Mails
- ✓ Multi-user online games
- ☐ Other mobile applications
- ▨ P2P
- ✓ Payment systems
- ✓ Pictures
- ✓ Social networks
- ☐ Standard mobile applications
- ▨ System files
- ▨ Thumbnails
- ✓ Videos

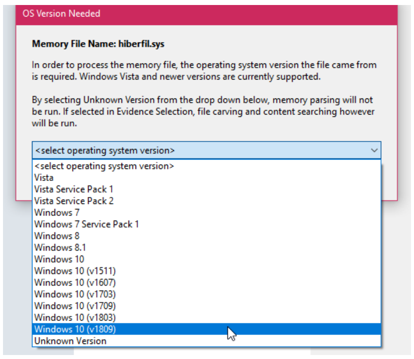
- ☐ Android
  - ☐ Dropbox
  - ☐ Google Docs
  - ☐ Google Drive
  - ☐ OneDrive
- ☐ iOS
  - ☐ Dropbox
- ☐ macOS
  - ☐ Google Drive
- ✓ Windows
  - ✓ Carbonite
  - ✓ Dropbox
  - ✓ Flickr
  - ✓ Google Drive
  - ✓ OneDrive
  - ✓ Yandex Disk

☐ Analyze ✓ Carve

Figure 22. BEC - artifacts search options.

2.5.10 BlackLight

BlackLight is commercial software that supports the analysis of modern hibernation files since version 2016 R3 [11]. We requested a trial of BlackLight software from its official website [70]. The test results included in this thesis were done on BlackLight trial version 2019 R3. The trial version includes full processing functionality with limitations on data exporting and reporting. Our first tests were done without installing any add-on for the software, then we installed the offline maps, operating system hash sets, and memory symbols installers as per BlackLight release 2019 R3 notes [71], to confirm maximum usage of all the software features.


OS Version Needed

Memory File Name: hiberfil.sys

In order to process the memory file, the operating system version the file came from is required. Windows Vista and newer versions are currently supported.

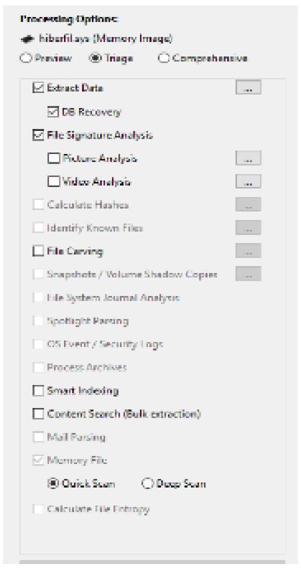
By selecting Unknown Version from the drop down below, memory parsing will not be run. If selected in Evidence Selection, file carving and content searching however will be run.

<select operating system version>

- <select operating system version>
- Vista
- Vista Service Pack 1
- Vista Service Pack 2
- Windows 7
- Windows 7 Service Pack 1
- Windows 8
- Windows 8.1
- Windows 10
- Windows 10 (v1511)
- Windows 10 (v1607)
- Windows 10 (v1703)
- Windows 10 (v1709)
- Windows 10 (v1803)
- [✓ Windows 10 (v1809)]
- Unknown Version

Figure 23. BlackLight - list of supported Windows 10 versions.

Figure 23 shows that the program detects the hibernation file as a memory image, while it requires identification of the Windows version, the latest supported Windows 10 version is 1809. BlackLight provides a list of choices for data processing, as shown in Figure 24. During the tests, we chose the comprehensive analysis mode, while enabling all its available options, to extract the maximum amount of data that BlackLight could extract from a hibernation file.



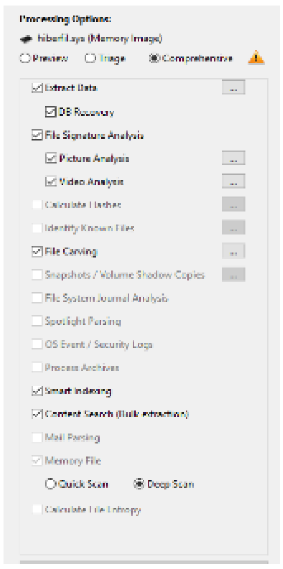
Processing Options:

hiberfil.sys (Memory Image)

Preview [✓ Triage] Comprehensive

- [✓] Extract Data
  - [✓] DB Recovery
- [✓] File Signature Analysis
  - [ ] Picture Analysis
  - [ ] Video Analysis
- [ ] Calculate Hashes
- [ ] Identify Known Files
- [ ] File Carving
- [ ] Snapshots / Volume Shadow Copies
- [ ] File System Journal Analysis
- [ ] Spotlight Parsing
- [ ] OS Event / Security Logs
- [ ] Process Archives
- [ ] Smart Indexing
- [ ] Content Search (Bulk extraction)
- [ ] Mail Parsing
- [✓] Memory File
  - [✓ Quick Scan] Deep Scan
- [ ] Calculate File Entropy

(a)



Processing Options:

hiberfil.sys (Memory Image)

Preview Triage [✓ Comprehensive]

- [✓] Extract Data
  - [✓] DB Recovery
- [✓] File Signature Analysis
  - [✓] Picture Analysis
  - [✓] Video Analysis
- [ ] Calculate Hashes
- [ ] Identify Known Files
- [✓] File Carving
- [ ] Snapshots / Volume Shadow Copies
- [ ] File System Journal Analysis
- [ ] Spotlight Parsing
- [ ] OS Event / Security Logs
- [ ] Process Archives
- [✓] Smart Indexing
- [✓] Content Search (Bulk extraction)
- [ ] Mail Parsing
- [✓] Memory File
  - Quick Scan [✓ Deep Scan]
- [ ] Calculate File Entropy

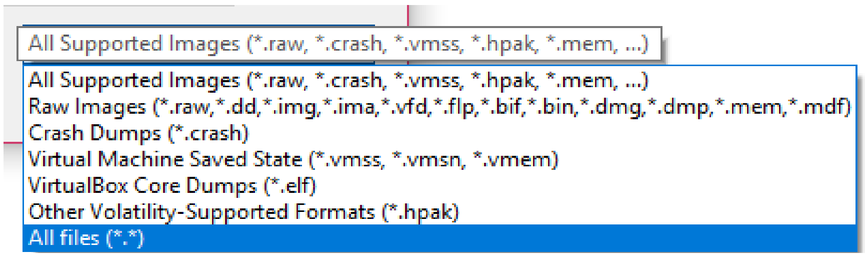
(b)

Figure 24. BlackLight - default processing options (a) versus chosen options (b).

2.5.11 Magnet Axiom

Magnet Forensics introduces Magnet Axiom as a complete digital investigation platform that can acquire evidence from different types of sources, recover and extract most evidence from each source of data, then analyze the artifacts and generate reports [72]. They also mention that the software can generate automatic insights using some features like Timeline, Connections, and Magnet.AI, which are a great support in investigations. Magnet Axiom is a GUI commercial software that has full integration of Volatility, such integration allows the use of different Volatility plugins through a GUI interface, instead of the Volatility command line. The software installs two programs, Magnet "Process" for case creation, and Magnet "Examine" where user can see case analysis details and recovered evidence.

We found the information about Magnet Axiom capability to analysis hibernation files in Windows Forensics Cookbook [73]. While we were not sure if the tool can analysis modern hibernation files or not, especially that such information was not found in Magnet Forensics official website or Magnet axiom user guide [74]. The user guide includes information about examples of supported memory dumps format like ".raw" and ".bin" files formats, and how to use Volatility integration feature to find the profile of memory dump, but no information about hibernation files analysis. When creating a new case, and choosing evidence source as memory, the hiberfil.sys is not shown in the supported images types, while we can still show all file types and choose a hibernation file. Figure 25 shows the list of supported memory images extensions.



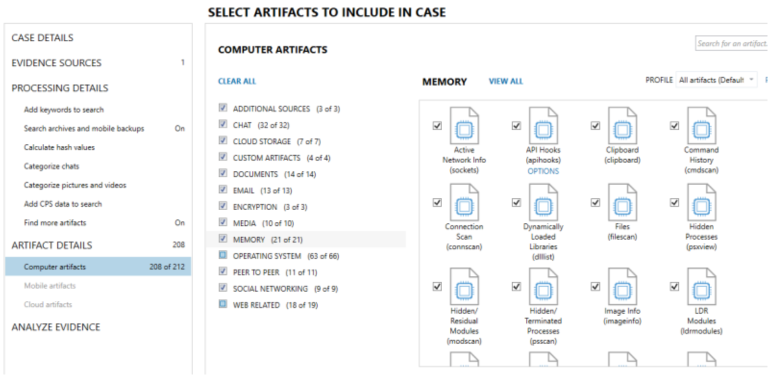
All Supported Images (*.raw, *.crash, *.vmss, *.hpak, *.mem, ...)

All Supported Images (*.raw, *.crash, *.vmss, *.hpak, *.mem, ...)
Raw Images (*.raw,*.dd,*.img,*.ima,*.vfd,*.flp,*.bif,*.bin,*.dmg,*.dmp,*.mem,*.mdf)
Crash Dumps (*.crash)
Virtual Machine Saved State (*.vmss, *.vmsn, *.vmem)
VirtualBox Core Dumps (*.elf)
Other Volatility-Supported Formats (*.hpak)
All files (*.*)

Figure 25. Magnet Axiom - supported memory images extensions.

It was surprising to find that Magnet Axiom can analyze Windows 10 hibernation files in its original format, despite the mentioned indicators that show that the program does not support this functionality. As automatic lookup of memory profile is powered by Volatility, such a feature would not function when we analyze a hibernation file, the user must identify the image profile (Windows version) in such a case. The latest supported Windows 10 memory dump profile for x64 machines is version is Win10x64_18362 (version 1903), same as Volatility.

As shown is Figure 26, Magnet Axiom provides many options in processing details. Options include but not limited to: adding keywords and regular expression to search, enable chat, video, and pictures categorization based on selected category (example of available categories for pictures: Child abuse, Documents (card/ID), Drones, Drugs, Weapons). It also includes a list of computer artifacts that users might choose to search for in the analyzed file, and this includes Volatility plugins under the memory artifacts. Once we choose the options and start the evidence analysis, Magnet Axiom “Examine” would open automatically in a new window to show analysis progress.



SELECT ARTIFACTS TO INCLUDE IN CASE

CASE DETAILS	
EVIDENCE SOURCES	1
PROCESSING DETAILS	
Add keywords to search	
Search archives and mobile backups	On
Calculate hash values	
Categorize chats	
Categorize pictures and videos	
Add CPS data to search	
Find more artifacts	On
ARTIFACT DETAILS	208
Computer artifacts	208 of 212
Mobile artifacts	
Cloud artifacts	
ANALYZE EVIDENCE	

COMPUTER ARTIFACTS

CLEAR ALL

- ✓ ADDITIONAL SOURCES (3 of 3)
- ✓ CHAT (32 of 32)
- ✓ CLOUD STORAGE (7 of 7)
- ✓ CUSTOM ARTIFACTS (4 of 4)
- ✓ DOCUMENTS (14 of 14)
- ✓ EMAIL (13 of 13)
- ✓ ENCRYPTION (3 of 3)
- ✓ MEDIA (10 of 10)
- ✓ MEMORY (21 of 21)
- ▪ OPERATING SYSTEM (63 of 66)
- ✓ PEER TO PEER (11 of 11)
- ✓ SOCIAL NETWORKING (9 of 9)
- ▪ WEB RELATED (18 of 19)

MEMORY VIEW ALL PROFILE All artifacts (Default

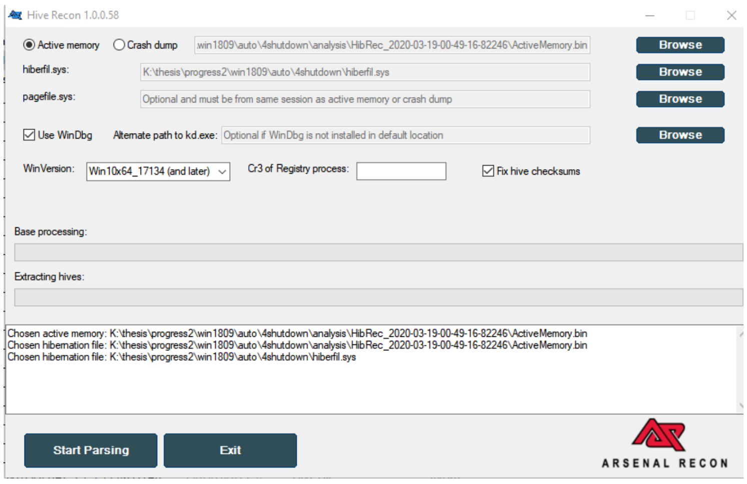
- ✓ Active Network Info (sockets)
- ✓ API Hooks (apihooks) OPTIONS
- ✓ Clipboard (clipboard)
- ✓ Command History (cmdscan)
- ✓ Connection Scan (connscan)
- ✓ Dynamically Loaded Libraries (dlllist)
- ✓ Files (filescan)
- ✓ Hidden Processes (psxview)
- ✓ Hidden/ Residual Modules (modscan)
- ✓ Hidden/ Terminated Processes (psscan)
- ✓ Image Info (imageinfo)
- ✓ LDR Modules (ldrmodules)

Figure 26. Magnet Axiom Process - Creating new case options.

We requested the software trial using Magnet Forensics website. Our tests were done on trial version 3.11.0.19007, which is a full version valid for 30 days.

2.5.12 Hive Recon

Hive Recon is another commercial Arsenal Recon software [75] that can extract registry-related data from a hibernation file. Hive Recon extracts registry hives from the Windows hibernation file by processing both “hiberfil.sys” and Hibernation Recon output “ActiveMemory.bin” or crash dump, user can add “pagefile.sys” to the input files, as shown in Figure 27. We processed our tests with Hive Recon 1.0.0.58. Arsenal Recon trial license does not have any restrictions on programs functionality. The program output includes data related to volatile registry hives. Hive Recon was not included in the thesis main plan and was the last tool to be added to the testing tools. Therefore, the results would not include much details about the software, due to time limitations.



(● Active memory) ○ Crash dump: win1809\auto\4shutdown\analysis\HibRec_2020-03-19-00-49-16-82246\ActiveMemory.bin

hiberfil.sys: K:\thesis\progress2\win1809\auto\4shutdown\hiberfil.sys

pagefile.sys: Optional and must be from same session as active memory or crash dump

[✓ Use WinDbg] Alternate path to kd.exe: Optional if WinDbg is not installed in default location

WinVersion: Win10x64_17134 (and later) Cr3 of Registry process: [✓ Fix hive checksums]

```
Chosen active memory: K:\thesis\progress2\win1809\auto\4shutdown\analysis\HibRec_2020-03-19-00-49-16-82246\ActiveMemory.bin
Chosen hibernation file: K:\thesis\progress2\win1809\auto\4shutdown\analysis\HibRec_2020-03-19-00-49-16-82246\ActiveMemory.bin
Chosen hibernation file: K:\thesis\progress2\win1809\auto\4shutdown\hiberfil.sys
```

Figure 27. Hive Recon 1.0.0.58 GUI.

Arsenal Recon has two other commercial software that can extract information related to Windows registry, which are Registry Recon and HBin Recon. We had the chance to test these two tools using the same Arsenal Recon trial license, as we were trying to differentiate between these two tools functionality and Hive Recon. We found that Registry Recon is totally out of the scope of our research, as it targets extracting registry information from disk images and volume. While understanding the difference between HBin and Hive Recon was challenging. Hive Recon target is extracting registry hives from a hibernation file or a crash dump and the tool can handle the special nature of volatile memory. On the other hand, HBin Recon can handle a decompressed hibernation file to extract “hbin” hives. Digital forensics practitioners can use Hive Recon and HBin

Recon to extract the maximum number of artifacts related to registry entries from a hibernation file[1]. We decided to include Hive Recon only to our tests, as this program was designed to deals with a hibernation file in the first place. More details about HBin and Hive Recon and their functionalities could be found at [76] and [77].

At the end of this introduction to the tools that will be used in our study, it is important to highlight that the initial plan of this thesis included benchmarking the different analysis tools. Due to the wide scope of the thesis, besides time and technical limitations, we decided to update the plan, and just concentrate on what types of artifacts could be extracted by each tool. As we already made some studies related to features comparison of some commercial tools, like the minimum system requirements – Appendix 4 and the supported types of artifacts – Appendix 5. We decided to add these details to the appendices, as this information might be useful for some readers.

[1] Arsenal Recon President was very supportive to explain for us in detail the difference between these two tools.

3 Research Methods

A controlled experiment method was used to perform the hibernation file analysis based on three variables: Windows version, hibernation file type, and power State, and two types of analysis as described in Figure 28. The files were analyzed using two types of analysis: Manual analysis and analysis with tools.

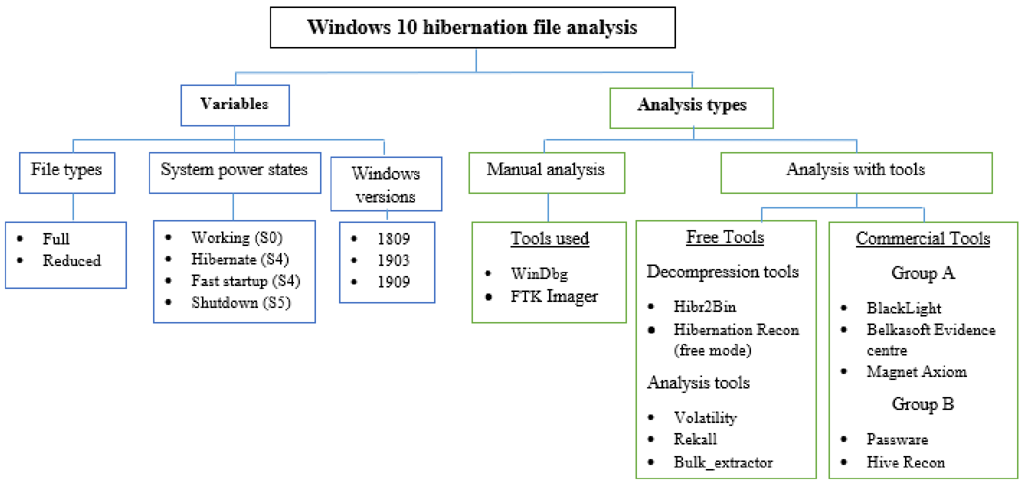


Figure 28. Components of the controlled experiment.

3.1 Variables

The content of the hibernation file and the file structure depends mainly on the three factors identified in this section.

3.1.1 Windows Version

The hibernation file structure and the names of its fields depend on the used Windows version, as described in section 2.3.5. As at the time of this research, Microsoft has released nine versions of Windows 10 [8]. For each of these versions there exist different editions, each edition contains various features depending on the required usage for the operating system. For example, the BitLocker Drive Encryption feature is available on the Pro and Enterprise editions of Windows 10, but not available on the Home edition [78]. Windows 10 has special releases for embedded OS, under the name of Windows 10 IoT, and this operating system has its editions [79]. Windows 10 IoT is out of the scope of our research. Paper [11] studied two versions of Windows 10 (v1511 and v1607)[1]. As per Table 9, Windows 10 versions earlier than v1809 are currently not supported by Microsoft, as they are end of service. The scope of this research is the Home edition of Windows 10 x64 versions 1809, 1903, 1909, which are the latest released windows versions.

Table 7. Windows 10 versions [8], [80].

Windows 10 version history	**Date of availability**	**OS Build**	**End of service for Home, Pro, Pro Education, and Pro for Workstations editions**	**End of service for Enterprise and Education editions**
1909	November 12, 2019	18363	May 11, 2021	May 10, 2022
1903	May 21, 2019	18362	December 8, 2020	December 8, 2020
1809	November 13, 2018	17763	May 12, 2020	May 11, 2021
1803	April 30, 2018	17134	November 12, 2019	November 10, 2020
1709	October 17, 2017	16299	April 9, 2019	April 14, 2020
1703	April 5, 2017	15063	October 9, 2018	October 8, 2019
1607	August 2, 2016	14393	April 10, 2018	April 9, 2019
1511	November 10, 2015	10586	October 10, 2017	October 10, 2017
1507	July 29, 2015	10240	May 9, 2017	May 9, 2017

The selected three versions were installed on three different formatted partitions on the test laptop hard drive. Appendix 6 provides specifications of hardware and OS used in this research. As the version variable aims to compare the structure used by the latest Windows versions, to the structure described in section 2.3.5.2. We decided to install Windows 10 v1607 on a virtual machine, to document the modifications that occurred on Windows hibernation file structure since the last version studied in [11].

[1] During the research we might refer to a special Windows 10 version by v*XXXX*. Example, Windows 10 version 1809 would be also written as v1809.

3.1.2 Hibernation file type

This study tested the two hibernation file types previously described in section 2.3.3. A Full hibernation file is created by the system when these three sleep options are enabled: hibernation, hybrid sleep, and fast startup. On the other hand, a reduced hibernation file is created by the system when the fast startup power state is the only enabled sleep option.

We noticed that Microsoft added the title “Fast startup (reduced hibernation file)” in [15] to describe the fast startup power state. For this reason, at some point of this research, we called any hibernation file extracted from the fast startup state as a reduced hibernation file. While during the hibernation file analysis phase, we noticed that the size of a hibernation file with “HIBR” signature was always the same, regardless of the configured hibernation type or power state. For this reason, we decided to use the expression “fast startup hibernation file”, when talking about a hibernation file extracted from a fast startup power state. Details would be discussed in section 4.4.

3.1.3 System power state

When talking about the system power state variable, we mean that the hibernation file was extracted at this power state. Our test cases covered hibernation files extracted from three different system states:

- **Working power State (S0)**: A working system might be waking from states the (S1-S5) as explained in section 2.2. We used FTK imager, launched with the “run as administrator” option, to extract hibernation files from the S0 power state. In this research, we used the expression extract “online” to refer to hibernation files extracted from a working system.

- **Hibernation power state (S4):** The use of a hibernation file is connected to three power states: Hibernate, fast startup and hybrid sleep. Our tests included hibernating the system following a hibernation request or a fast startup request. The hybrid sleep power state is out of the scope of this research. Collecting the hibernation file in the S4 state could be done using different ways. During this research, we extracted the hibernation files by rebooting the test laptop from an external HD, after modifying the boot order using the BIOS setup utility. We used FTK imager installed on the external HD to extract hibernation files. In this research, we used the expression extract "offline" to refer to hibernation files extracted from a non-working state (S1-S5).

- **Soft Off power state (S5)**: we tested different power states configurations to understand the cases in which a system enters the shutdown state (S5). File at the S5 state was collected offline.

3.2 Analysis Types

The collected hibernation files were analyzed using two types of analysis as described in this section.

3.2.1 Manual analysis

When talking about a Manually analysis of the hibernation file, we mean that the analysis does not depend on the tools' capabilities. The function of tools in the manual analysis is displaying the file content. The analysis results depend on the researcher knowledge and observations. The manual analysis was challenging and might be done in an independent study as it is time-consuming. The manual analysis is fundamental to understand the hibernation file characteristics. Using manual analysis, we explored the content of hibernation file pages and documented our findings in Chapter 4. The manual analysis was done on seven stages.

Stage 1: We started our analysis by the file header. As discussed in section 2.3.5, the first page of the hibernation file is defined by "PO_MEMORY_IMAGE" structure. Using WinDbg outputs, we documented the definition of the header structure. This analysis would be discussed in section 4.2.1. The results of this analysis include a datagram of the hibernation file header. This Stage should provide an answer to the first research question (section 1.1)

Stage 2: We then proceeded to document the content of the second page of the hibernation file, which could also be named as the processor context. This page is defined by the structure "_KPROCESSOR_STATE". Using WinDbg outputs, we documented the definition of the processor context page. This part of the manual analysis would be discussed in section 4.2.2. The results of this analysis include a datagram of the processor context. This Stage should provide an answer to the first research question (section 1.1)

Stage 3: It was not possible to extract the pages structures using WinDbg starting from the third page of the hibernation file, as they are not defined publicly. This phase required the generation of hibernation files to analyze the file content using FTK imager. The aim of this stage was identifying any undocumented structures/pages when comparing the analyzed files to the file layout defined by [11] and explained in section 2.3.5.2. This analysis would be discussed in section 4.3.1. The results of this analysis include observations related to undocumented structures.

Stage 4: The aim of this stage was understanding how the physical memory pages are stored in a hibernation file. We applied the restoration set concepts described in section 2.3.5.2, on test hibernation files. All data related to a restoration set could be extracted manually from the hibernation file header, as the header is not compressed. We used the header datagram created at Stage 1 to extract the information required for manually analyze the content of a restoration set using FTK imager. This analysis would be discussed in section 4.3.2. The results of this analysis would clarify if the hibernation file still uses the same restoration sets characteristics that were defined by [11].

Stage 5-a: This stage aimed to study how restoration set is decompressed and investigate its content. For this reason, we decided to take advantage of Hibr2Bin open-source code. We would explain how this analysis was done in section 4.3.3. This analysis helped to confirm our understanding of the hibernation file structure and clarified some of the unknown structures found in stage 3.

Stage 5-b: Comparing the decompressed files using Hibr2Bin and Hibernation Recon. This Stage should provide an answer to the research question number 4 (section 1.1)

Stage 6: Starting from stage 6 we aimed to understand the impact of various windows configurations on the hibernation file content. The analysis would be discussed in section 4.4. This Stage should provide an answer to the research question number 5 (section 1.1)

Stage 7: The last stage of the manual analysis was exploring the characteristics of hibernation files with “HORM” signatures. This signature was out of the scope of [11], and we wanted to clarify the impact of enabling the HORM feature on the hibernation file. This analysis did not include extracting hibernation files offline due to technical limitations. The analysis results would be discussed in section 4.5. This Stage should provide an answer to the research question number 6 (section 1.1)

3.2.2 Analysis of hibernation file using tools

The aim of performing a hibernation file analysis using tools was extracting artifacts from the hibernation file. This part is a simulation for what a forensics practitioner does in real life. The type of analysis depends on the capability of each tool to analyze the hibernation file and the features of the tool. By analyzing hibernation files using different tools, we wanted to demonstrate that the hibernation file contains considerable types of artifacts. The number of extracted artifacts from a hibernation file does not depend only on the file content, while the used tool to extract the artifacts is a crucial factor in the results.

We first created artifacts and footprints on the test laptop based on a predefined scenario, and then we hibernated the test laptop. By this way, we were able to search for the predefined list of evidence in the outputs of the tested tools. We analyzed the extracted files using two types of tools:

Free Tools: A free tool does not mean that the tool is open source. A tool like Hibernation Recon, for example, could run in free mode while the source code is not published. All the tested free tools were not designed to analyze a modern hibernation file. To overcome such problem, we added a decompression stage for the hibernation file. The decompression of a hibernation file could be done using Hibr2Bin and Hibernation Recon as discussed, respectively, in sections 2.5.3 and 2.5.4. By decompressing the hibernation file, we have a higher possibility to find free tools that could extract data from the file. We tested the analysis of a decompressed file using Volatility and Rekall are examples of widely used memory forensics tools. We added Bulk_extractor to the test plan after reading [63], as clarified earlier.

Commercial Tools: Initially, four commercial tools were chosen for testing. We chose tools claiming extracting data from Windows 10 hibernation files, while the test also included tools chosen based on online topics or books recommendations. A limited testing period was guaranteed by each vendor. We expanded the list of tested commercial tool, by adding Hive Recon, after receiving a testing license for all Arsenal Recon products. We could categorize the tested commercial tools into two categories. A category of tools with a wide scope that can extract various types of artifacts. BlackLight, Belkasoft Evidence center, and Magnet Axiom are typical examples. Each of the three tools covers a wide number of artifacts types. In this research, we would name this category of tools as Category A. The other category of tools has limited scope and can extract some specific types of artifacts that might not be covered by Category A tools. Hive Recon and Passware are typical examples. Hive Recon extract data related to registry entries, and Passware can find passwords and different encryption keys. We named this category of tools as Category B. some of Category A tools might integrate with tools from Category B in their features to support more artifacts. For example, the integration of Magnet Axiom with Passware through Passware plugin available in Magnet Axiom Process.

The analysis of the output of Hibr2Bin and Hibernation Recon would be discussed in chapter 4. The decompression tools do not extract any artifacts from the hibernation file. They are used in the file analysis done using tools as a workaround for tools that doesn't support the direct analysis of hibernation file.

Chapter 5 would discuss the findings of the analysis of the hibernation file using tools. Rekall results were omitted from Chapter 5 results, as explained in section 2.5.6.

3.3 Test Cases

We created four test case to respond with the analysis requirements states in section 3.2. Each test case helped us to answer to one or more of the research questions. We created relevant users activities for each test case, to be tracked during the analysis.

3.3.1 Test Case A

The test case A helped to answer the research questions : "1. Is the Modern hibernation file layout [11] still applicable for the latest versions of Windows 10 hibernation file?" and "2. Do the free tools decompressing Windows 10 hibernation file have the same output file when processing the same input hibernation file?"

This test case was applied on Windows 10 versions v1809, v1903 and v1909. For each of the three Windows versions, we extracted four hibernation files:

- File_1: We collected the hiberfil.sys file from the working system state.
- File_2: We collected the hiberfil.sys file from hibernate system state. The system entered the hibernate state in response to a hibernation request. In other words, we hibernated the system using the hibernate power option in Windows start menu.

- File_3: We collected the hiberfil.sys file from the system after waking from hibernation. This file is collected online.

- File_4: We collected the hiberfil.sys file offline after system shutdown using the shutdown option in Windows start menu. This file is a fast startup hibernation file.

As for user activities, during the working state, we created a draft notepad file. The file contained some keywords that we might use in the search.

3.3.2 Test case B

The test case B helped to answer the research question: "5. What are the impacts of the modifications of power configurations, as well as power states on a hibernation file content?"(section 1.1)

The test plan included many steps and thirteen hibernation files extraction. Due to the test length, we decided to perform the test on Windows 10 v1903 only. This version is very similar to the latest Windows version. In plus, we have noticed from the primary investigations that a hibernation file extracted from the working state contains more pages than the other two versions, in case of windows v1903. Test case results would be discussed in section 3.3.2.

As for user activities, there were no specific user activities requirement in that test case. We opened some URLs on the working system and documented our activities before the system state transition.

3.3.3 Test case C

Hibernate Once Resume Many (HORM) feature could be found on Windows 10 Education and Enterprise editions starting from v1709. This feature was only available on IoT Windows by the time of writing [11], which made it out of the scope of the paper. We were not able to test such functionality on our test laptop as the tests were done on Windows Home edition. As we believe that studying the characteristics of a hibernation file with "HORM" signature is beneficial, as it would give information about this file structure and how the feature works. We upgraded our analysis laptop to Windows 10 v1909 Build 18363.720. Using the HORM feature, Windows could resume many times using the same hibernation file. Enabling this feature requires Configuring the Windows system using the instructions indicated in [33]. Due to technical problems, we were not able to extract an offline version of the file, so our analysis was limited to the online collected hibernation files. Test case C answers the research question: What is the effect of enabling the “HORM” feature on the hibernation file characteristics? This Stage should provide an answer to the research question number 6 (section 1.1)

3.3.4 Test case D

Test case D have the same plan of test case A, described in section 3.3.1, except for what concerns the activities. Four hibernation files are the output of test case D (D_File1, D_File2, D_File3 and D_File4), and the three Windows 10 versions are included in the test.

Test case D answers the below research questions: “7. Could we extract artifacts from Windows 10 hibernation files using free tools?”, “8. Are there any differences found between commercial tools outputs?”, “9. What kind of artifacts could be collected from a Windows 10 hibernation file?”, “10. In which Windows 10 power state, a hibernation file would contain the maximum number of artifacts?”

It was required to generate different types of artifacts on the system in the working state, to answer the research questions. Before creating what we named “the user activities scenario”. We first created a list of artifacts types and identified the list of applications that we target by this test case. Then we created a user activities scenario based on the predefined list of artifacts. The user activities scenario was applied to the three Windows versions. While due to technical issues, we repeated the tests on Windows v1903 more than four times. We took advantage of this matter, to create an extended user activities scenario for v1903. Subsequently, we created the list of evidence shown in Tables 10 and 11. We marked an evidence number (E*x*) for each action of the evidence list, to help in tracking the evidence on each of the tested tools. The output hibernation files were then processed by each of the tools mentioned in section 3.2. finally, we analyzed the outputs of the tool. Results of the analysis are discussed in chapter 5.

A hibernation file contains system footprints and files, besides the user activities. A hibernation file extracted from the hibernate power state would always contain data related to the system, such as system files and system footprint, while it might not contain user activities. The fast startup hibernation file is a case in point. Having a list of evidence helped us to differentiate between a fast startup hibernation file (File_4) and a hibernation file that contains user activities (File_2).

Table 8. List of evidence of test case D.

Application / Evidence number	User activities scenario	Target artifact type
Firefox		
E1	Google search for “coronavirus report”	URL
E2	Open URL: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/situation-reports	URL
E3	Open online PDF	URL, PDF
Chrome incognito		
E4	Google search for “Memory forensics hack”	URL, Google search
E5	Open URL: https://www.hackingarticles.in/memory-forensics-investigation-using-volatility-part-1/	URL
E6	Open Google book online.	URL, picture
E7	Open Google maps: https://www.Google.com/maps/@59.3945433,24.6454586,13z	URL, Location, picture of the map
Chrome		
E8	Stream video: https://www.youtube.com/watch?v=9lwYrfE6FA0	URL, Parts of video, Social media site
E9	Google search for flowers.	URL, flowers pictures
E10	Open a flower picture and download it.	URL, download activity
E11	Enter the webmail password and do not save it in the browser. URL: https://outlook.live.com/mail/0/inbox	URL, account name, password
E12	create a draft email in webmail. Message body includes keywords: “test”, “draft”.	Draft email content
Skype / E13	Send chat messages using a private message	Chat content
Libre Office / E14	Create a draft text file to include keywords: “test”, “hibernation file”, Windows version, date, and time. The file will not be saved.	Draft document content
Windows drives / E15	Unlock at least 2 Windows encrypted drives with BitLocker	BitLocker ID, password

Table 9. List of evidence created only on v1903.

Application / Evidence number	User activities scenario	Target artifact type
Chrome / E16	Access Gmail https://mail.Google.com/mail/u/0/#inbo?compose=new and created a draft email.	URL, username, password, draft email
Chrome / E17	Do Hangout chat.	Chat
Edge/ E18	Access URL: https://www.msn.com/en-gb/news/coronav patients-test-positive-again-in-blow-to-immunity-hopes/ BB12rSb0?ocid=spartandhp	URL
Edge - InPrivate / E19	Search for “secret” using bing.com.	URL, Bing search
Outlook / E20	Create a draft email using outlook.	Draft email
Notepad / E21	Create a draft text file.	Draft text content
Windows Cmd / E22	Using cmd.exe, run the commands: “ipconfig /all” and “netstat -a”	Cmd session
BitTorrent / E23	Use BitTorrent to download a file.We hibernated the system during the file download.	URL, remote connection IP

4 Manual analysis of Windows 10 hibernation file

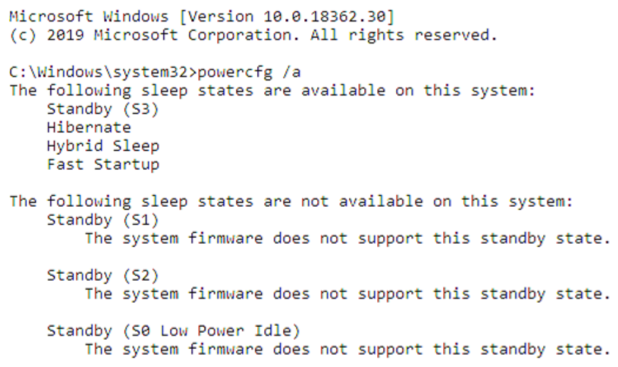
In this section, we would perform a manual analysis of Windows 10 hibernation file, to study the file layout. By the end of this section, we should have a clear understanding of the Windows 10 hibernation file layout and structure definitions.

4.1 Default Windows 10 settings

This section aims to understand the default Windows 10 power configurations. By the end of this section, we should answer the question: "Is a hibernation file created by default by Windows 10?"

As discussed earlier, the hibernation file is a protected operating system file. The file is hidden by default, to view it, we must modify the folder options[1]. Once the protected files are visible, we found that for all the tested Windows versions, the files hiberfil.sys, pagefile.sys, swapfile.sys are available by default. The hibernation file size in the working power state is equal to 1.52 GB. Taking into consideration that the "installed physical memory (RAM)" for the test laptop is 4 GB, and the "total physical memory" is 3.82 GB, we found that the hibernation file size equals 40% of the total physical memory size.

The default available sleep states were checked using the command prompt. For all tested Windows versions, four sleep states available by default: Standby (S3), Hibernate, Hybrid Sleep, and Fast Startup, as shown in Figure 29.



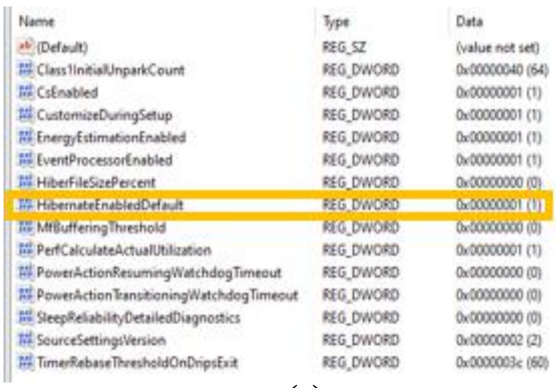
```
Microsoft Windows [Version 10.0.18362.30]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>powercfg /a
The following sleep states are available on this system:
    Standby (S3)
    Hibernate
    Hybrid Sleep
    Fast Startup

The following sleep states are not available on this system:
    Standby (S1)
        The system firmware does not support this standby state.

    Standby (S2)
        The system firmware does not support this standby state.

    Standby (S0 Low Power Idle)
        The system firmware does not support this standby state.
```

Figure 29. The Default available sleep states in a Windows 10 v1903.

[1] https://kb.blackbaud.com/articles/Article/41890

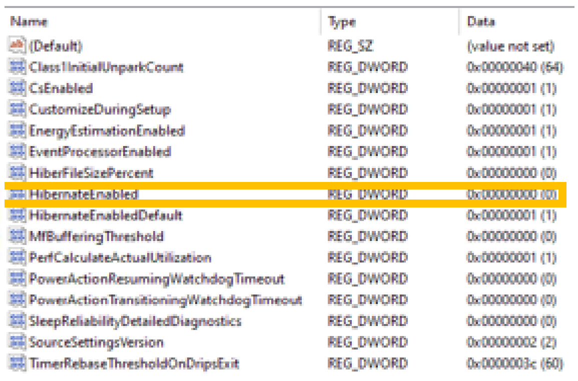
Newly installed Windows 10 Home edition has a hibernation registry enabled by default, this was checked using the registry editor. This registry entry could be found in the path Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power. For Windows 10 version 1809, the registry named “HibernateEnabled” is enabled by default, as shown in Figure 30. Disabling the hibernation option using the command “powercfg -h off” using command prompt (run the Cmd as administrator) [5], the “HibernateEnabled” registry value turns to zero and the available sleep states would be just Standby (S3).

For Windows 10 Home edition versions 1903 and 1909, it was found that a registry named is "HibernateEnabledDefault” is enabled by default. Another Registry named “HibernateEnabled” appears after disabling the hibernation option using the command “powercfg -h off “. Figure 30 shows the default power registry entries for Windows 10 v1909, versus the entries list after disabling the hibernation. When the hibernation option is enabled again using the command prompt, the “HibernateEnabled” registry value turns to 1.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Class1InitialUnparkCount	REG_DWORD	0x00000040 (64)
CsEnabled	REG_DWORD	0x00000001 (1)
CustomizeDuringSetup	REG_DWORD	0x00000001 (1)
EnergyEstimationEnabled	REG_DWORD	0x00000001 (1)
EventProcessorEnabled	REG_DWORD	0x00000001 (1)
HiberFileSizePercent	REG_DWORD	0x00000000 (0)
HibernateEnabledDefault	REG_DWORD	0x00000001 (1)
MfBufferingThreshold	REG_DWORD	0x00000000 (0)
PerfCalculateActualUtilization	REG_DWORD	0x00000001 (1)
PowerActionResumingWatchdogTimeout	REG_DWORD	0x00000000 (0)
PowerActionTransitioningWatchdogTimeout	REG_DWORD	0x00000000 (0)
SleepReliabilityDetailedDiagnostics	REG_DWORD	0x00000000 (0)
SourceSettingsVersion	REG_DWORD	0x00000002 (2)
TimerRebaseThresholdOnDripsExit	REG_DWORD	0x0000003c (60)

(a)



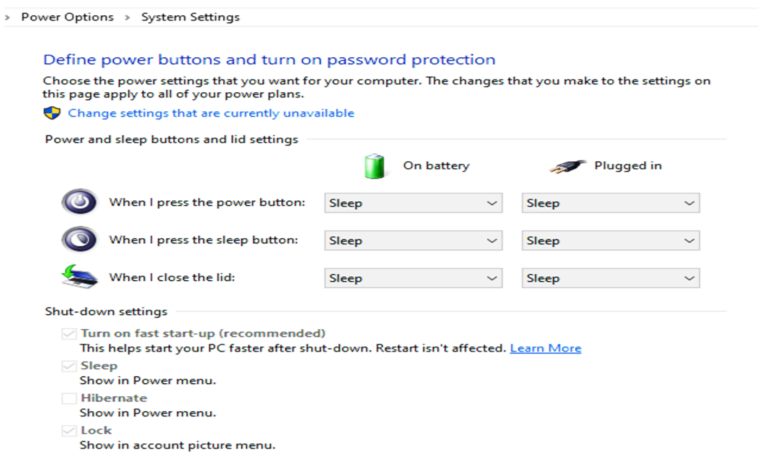
Name	Type	Data
(Default)	REG_SZ	(value not set)
Class1InitialUnparkCount	REG_DWORD	0x00000040 (64)
CsEnabled	REG_DWORD	0x00000001 (1)
CustomizeDuringSetup	REG_DWORD	0x00000001 (1)
EnergyEstimationEnabled	REG_DWORD	0x00000001 (1)
EventProcessorEnabled	REG_DWORD	0x00000001 (1)
HiberFileSizePercent	REG_DWORD	0x00000000 (0)
HibernateEnabled	REG_DWORD	0x00000000 (0)
HibernateEnabledDefault	REG_DWORD	0x00000001 (1)
MfBufferingThreshold	REG_DWORD	0x00000000 (0)
PerfCalculateActualUtilization	REG_DWORD	0x00000001 (1)
PowerActionResumingWatchdogTimeout	REG_DWORD	0x00000000 (0)
PowerActionTransitioningWatchdogTimeout	REG_DWORD	0x00000000 (0)
SleepReliabilityDetailedDiagnostics	REG_DWORD	0x00000000 (0)
SourceSettingsVersion	REG_DWORD	0x00000002 (2)
TimerRebaseThresholdOnDripsExit	REG_DWORD	0x0000003c (60)

(b)

Figure 30. Default Windows power registries(a), disabled the hibernation option(b).

The command shutdown -h hibernate the system, using the default system configuration. While the “Hibernate” option is not shown by default in the power options of the Windows start menu. Showing the hibernation option is a feature related to the power options setting in the control panel, while it does not mean that the hibernation is disabled on the system. Figure 31 shows that the fast startup, sleep and lock are the available default options in the power options of the Windows start menu.

This section concludes that the hibernation file is created by default by the Windows 10 versions 1809, 1903, 1909.


› Power Options › System Settings

Define power buttons and turn on password protection

Choose the power settings that you want for your computer. The changes that you make to the settings on this page apply to all of your power plans.

Change settings that are currently unavailable

Power and sleep buttons and lid settings

	On battery	Plugged in
When I press the power button:	Sleep	Sleep
When I press the sleep button:	Sleep	Sleep
When I close the lid:	Sleep	Sleep

Shut-down settings

- ✓ Turn on fast start-up (recommended)
  This helps start your PC faster after shut-down. Restart isn't affected. Learn More
- ✓ Sleep
  Show in Power menu.
- ☐ Hibernate
  Show in Power menu.
- ✓ Lock
  Show in account picture menu.

Figure 31. Default Shut-down setting for Windows 10 x64.

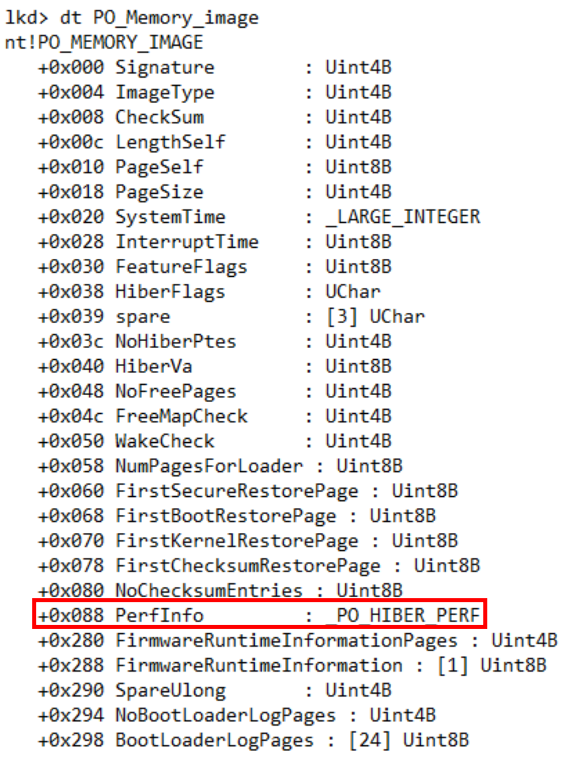
4.2 Definitions of the known structures

In this section, we would document the known file structures content. This includes the first two pages of the hibernation files, as we already know their structure names and we can search for its content using WinDbg. By the end of this section, we should answer the question: “Could we document the known hibernation file structures in a clear mapping/datagram?”

4.2.1 File header

As discussed in section 2.3.5, the first page of the modern Windows hibernation file is composed of the “PO_MEMORY_IMAGE” structure. We used WinDbg to extract the header structure for the Windows versions question of this research. The command “dt PO_MEMORY_IMAGE” could be used to show the content of the header without recursion, as shown in Figure 32. The figure shows a length for some fields, while some fields “PerfInfo”, has a length defined by a structure (actually, a sub-structure) “_PO_HIBER_PERF”. In that case, it is required to add a recursion[1] option to the “dt” [81] command to extract all the fields names of the header.

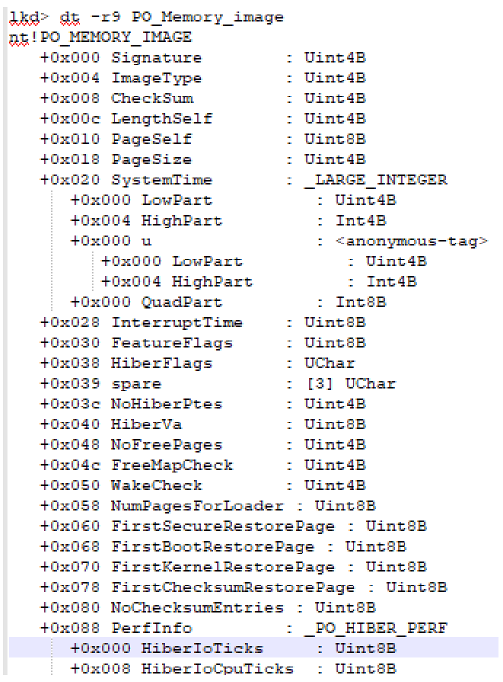
[1] The “-b” option could be added to the commend to display the content of any sub-structures. Using the option “-r[*depth*]” is another way to show the fields of a structure, in plus of sub-structures. The“-r” option requires a depth level from 1 to 9, as the “-r” would “Recursively dumps the subtype fields. If depth is given, this recursion will stop after depth levels” [81]. We recommend using that the “-r” option, as it shows slightly more options (in case the a field name is repeated more than time).



```
lkd> 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     : Uint8B
   +0x038 HiberFlags       : UChar
   +0x039 spare            : [3] UChar
   +0x03c NoHiberPtes      : Uint4B
   +0x040 HiberVa          : Uint8B
   +0x048 NoFreePages      : Uint4B
   +0x04c FreeMapCheck     : Uint4B
   +0x050 WakeCheck        : Uint4B
   +0x058 NumPagesForLoader : Uint8B
   +0x060 FirstSecureRestorePage : Uint8B
   +0x068 FirstBootRestorePage : Uint8B
   +0x070 FirstKernelRestorePage : Uint8B
   +0x078 FirstChecksumRestorePage : Uint8B
   +0x080 NoChecksumEntries : Uint8B
   +0x088 PerfInfo         : _PO_HIBER_PERF
   +0x280 FirmwareRuntimeInformationPages : Uint4B
   +0x288 FirmwareRuntimeInformation : [1] Uint8B
   +0x290 SpareUlong       : Uint4B
   +0x294 NoBootLoaderLogPages : Uint4B
   +0x298 BootLoaderLogPages : [24] Uint8B
```

Figure 32. Windows 10 v1903 x64 - “PO_MEMORY_IMAGE” structure definition.

The output of command “dt -r9 PO_Memory_image” using WinDbg is shown in Figure 33. Windows debug shows the sub-structure offsets referring to the first offset of the sub-structure. While the offset from the beginning of the file or to the beginning of the main structure “PO_MEMORY_IMAGE”, which is offset 0x088, is not shown in the output.



```
lkd> dt -r9 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
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
      +0x000 u                : <anonymous-tag>
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
      +0x000 QuadPart         : Int8B
   +0x028 InterruptTime    : Uint8B
   +0x030 FeatureFlags     : Uint8B
   +0x038 HiberFlags       : UChar
   +0x039 spare            : [3] UChar
   +0x03c NoHiberPtes      : Uint4B
   +0x040 HiberVa          : Uint8B
   +0x048 NoFreePages      : Uint4B
   +0x04c FreeMapCheck     : Uint4B
   +0x050 WakeCheck        : Uint4B
   +0x058 NumPagesForLoader : Uint8B
   +0x060 FirstSecureRestorePage : Uint8B
   +0x068 FirstBootRestorePage : Uint8B
   +0x070 FirstKernelRestorePage : Uint8B
   +0x078 FirstChecksumRestorePage : Uint8B
   +0x080 NoChecksumEntries : Uint8B
   +0x088 PerfInfo         : _PO_HIBER_PERF
      +0x000 HiberIoTicks     : Uint8B
      +0x008 HiberIoCpuTicks  : Uint8B
```

Figure 33. Win10 version 1903 x64 - “PO_MEMORY_IMAGE” with recursion.

To avoid confusion, we created a datagram (Table 10), where we illustrated the full content of “PO_MEMORY_IMAGE” and included the content of the sub-structure “_PO_HIBER_PREF” while referring the offsets to the beginning of hibernation file.

We also extracted the full content of the header structure for Windows 10 versions 1809, 1903 and 1909 (documented in Appendix 7). After extracting the content of the header, we noticed that the header structure is the same for Windows 10 version 1903 and 1909. The WinDbg is showing the same Windows built for both Windows version (18362.1.amd64fre.19h1_release.190318-1202), which is related to version 1903 build, in spite the fact that the debug for Windows 10 version 1909 was done on build 18363.628.

As explained in section 3.1.1, we used a virtual machine with Windows 10 v1607, to extract the header structure of the last version analyzed in paper [11]. We created a table to compare the structure of version 1607 to the three versions in the scope of our research. A table comparing the header's entries of Windows versions 1607, 1809, 1903, and 1909 could be found in Appendix 8.

From the analysis of Appendix 8, we found that some of the names of the fields are the same across different Windows versions, while their location varies from a version to another. In such a case, we mentioned the relevant offset for the required variable in the Windows version column. For example, “FirmwareRuntimeInformationPages” could be found at offset 0x280 for Windows 10 versions 1809,1903, 1909, while it is found at offset 0x270 for Windows version 1607. We also noticed that “FirstBootRestorePage”, “FirstKernelRestorePage”, “NumPagesForLoader” and “_PO_HIBER_PERF” structure are located at the same offsets for the four tested versions. We believe that this last information is important, as we understand from section 2.3.5.2 that these fields are key points in finding the file’s restoration sets. In case one of these last fields modifies its location, it might not be feasible for some tools to decompress the file content.

We documented the structure of the hibernation file in the below header. The only difference found between versions 1903,1909 and version 1809 is that the field “HvPageTableRoot” starting at offset 0x368 replaced “HvCr3” that was available in version 1809. We referred to the content of the “_PO_HIBER_PREF” structure as “PrefInfo.*xxx*”.

Table 10. Hibernation file header datagram for Win10 v1903 and 1909 x64.

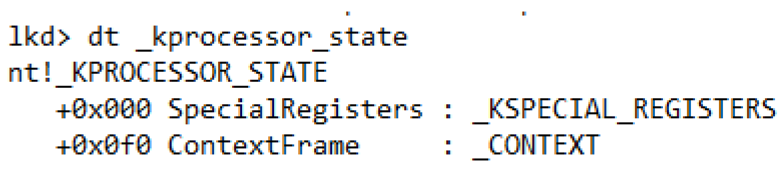
	00	01	02	03	04	05	06	07	08	09	A	B	C	D	E	F
0x000	Signature				ImageType				CheckSum				LengthSelf			
0x010	PageSelf								PageSize							
0x020	SystemTime								InterruptTime							
0x030	FeatureFlags								HiberFlags	spare	spare	spare	NoHiberPtes			
0x040	HiberVa								NoFreePages				FreeMapCheck			
0x050	WakeCheck								NumPagesForLoader							
0x060	FirstSecureRestorePage								FirstBootRestorePage							
0x070	FirstKernelRestorePage								FirstChecksumRestorePage							
0x080	NoChecksumEntries								PerfInfo.HiberIoTicks							
0x090	PerfInfo.HiberIoCpuTicks								PerfInfo.HiberInitTicks							
0x0A0	PerfInfo.HiberHiberFileTicks								PerfInfo.HiberCompressTicks							
0x0B0	PerfInfo.HiberSharedBufferTicks								PerfInfo.HiberChecksumTicks							
0x0C0	PerfInfo.HiberChecksumIoTicks								PerfInfo.TotalHibernateTime							
0x0D0	PerfInfo.HibernateCompleteTimestamp								PerfInfo. POSTTime				PerfInfo. ResumeBootMgrTime			
0x0E0	PerfInfo. BootmgrUserInputTime								PerfInfo.ResumeAppTicks							
0x0F0	PerfInfo.ResumeAppStartTimestamp								PerfInfo.ResumeLibraryInitTicks							
0x100	PerfInfo.ResumeInitTicks								PerfInfo.ResumeRestoreImageStartTimestamp							
0x110	PerfInfo.ResumeHiberFileTicks								PerfInfo.ResumeIoTicks							
0x120	PerfInfo.ResumeDecompressTicks								PerfInfo.ResumeAllocateTicks							
0x130	PerfInfo.ResumeUserInOutTicks								PerfInfo.ResumeMapTicks							
0x140	PerfInfo.ResumeUnmapTicks								PerfInfo.ResumeChecksumTicks							
0x150	PerfInfo.ResumeChecksumIoTicks								PerfInfo.ResumeKernelSwitchTimestamp							
0x160	PerfInfo.CyclesPerMs								PerfInfo.WriteLogDataTimestamp							
0x170	PerfInfo.KernelReturnFromHandler								PerfInfo.TimeStampCounterAtSwitchTime							
0x180	PerfInfo.HalTscOffset								PerfInfo.HvlTscOffset							
0x190	PerfInfo.SleeperThreadEnd								PerfInfo.PostCmosUpdateTimestamp							
0x1A0	PerfInfo. KernelReturnSystemPowerStateTimestamp								PerfInfo.IoBoundedness							
0x1B0	PerfInfo.KernelDecompressTicks								PerfInfo.KernelIoTicks							
0x1C0	PerfInfo.KernelCopyTicks								PerfInfo.ReadCheckCount							
0x1D0	PerfInfo.KernelInitTicks								PerfInfo.KernelResumeHiberFileTicks							
0x1E0	PerfInfo.KernelIoCpuTicks								PerfInfo.KernelSharedBufferTicks							
0x1F0	PerfInfo.KernelAnimationTicks								PerfInfo.KernelChecksumTicks							
0x200	PerfInfo.KernelChecksumIoTicks								PerfInfo.AnimationStart							
0x210	PerfInfo.AnimationStop								PerfInfo.DeviceResumeTime							
0x220	PerfInfo.SecurePagesProcessed								PerfInfo.BootPagesProcessed							
0x230	PerfInfo.KernelPagesProcessed								PerfInfo.BootBytesWritten							
0x240	PerfInfo.KernelBytesWritten								PerfInfo.BootPagesWritten							
0x250	PerfInfo.KernelPagesWritten								PerfInfo.BytesWritten							
0x260	PerfInfo.PagesWritten				PerfInfo.FileRuns				PerfInfo. NoMultiStageResumeReason				PerfInfo. MaxHuffRatio			
0x270	PerfInfo.AdjustedTotalResumeTime								PerfInfo.ResumeCompleteTimestamp							
0x280	FirmwareRuntimeInformationPages								FirmwareRuntimeInformation							
0x290	SpareUlong				NoBootLoaderLogPages				BootLoaderLogPages							
0x2A0	BootLoaderLogPages								BootLoaderLogPages							
0x2B0	BootLoaderLogPages								BootLoaderLogPages							
0x2C0	BootLoaderLogPages								BootLoaderLogPages							
0x2D0	BootLoaderLogPages								BootLoaderLogPages							
0x2E0	BootLoaderLogPages								BootLoaderLogPages							
0x2F0	BootLoaderLogPages								BootLoaderLogPages							
0x300	BootLoaderLogPages								BootLoaderLogPages							
0x310	BootLoaderLogPages								BootLoaderLogPages							
0x320	BootLoaderLogPages								BootLoaderLogPages							
0x330	BootLoaderLogPages								BootLoaderLogPages							
0x340	BootLoaderLogPages								BootLoaderLogPages							
0x350	BootLoaderLogPages								NotUsed				ResumeContextCheck			
0x360	ResumeContextPages				Hiberboot	SecureLaunched	SecureBoot		HvPageTableRoot							
0x370	HvEntryPoint								HvReservedTransitionAddress							
0x380	HvReservedTransitionAddressSize								BootFlags							
0x390	RestoreProcessorStateRoutine								HighestPhysicalPage							

	00	01	02	03	04	05	06	07	08	09	A	B	C	D	E	F
0x3A0	BitlockerKeyPfns								BitlockerKeyPfns							
0x3B0	BitlockerKeyPfns								BitlockerKeyPfns							
0x3C0	HardwareSignature								SMBiosTablePhysicalAddress							
0x3D0	SMBiosTableLength				SMBiosMajorVersio	SMBiosMinorVersion	HiberResumeXhciHa	InitializeUSBCore	ValidUSBCoreId	USBCoreId	SkipMemoryMapVali					

From this section, We conclude that decompressing the memory pages would not be impacted by the minor variations in the header content between different versions of hibernation files. The locations of fields used to restore the memory pages (“FirstBootRestorePage”, “FirstKernelRestorePage”, “NumPagesForLoader” and the structure “_PO_HIBER_PERF”) were not modified since v1607.

4.2.2 Processor context

As per [11], the header’s structure is followed by the “_KPROCESSOR_STATE” structure, which we could find at the offset 0x1000. The “_KPROCESSOR_STATE” structure is composed of two sub-structures: “_KSPECIAL_REGISTERS” and “_CONTEXT”, as shown in Figure 34.



```
lkd> dt _kprocessor_state
nt!_KPROCESSOR_STATE
   +0x000 SpecialRegisters : _KSPECIAL_REGISTERS
   +0x0f0 ContextFrame     : _CONTEXT
```

Figure 34. “_KPROCESSOR_STATE” structure - Win10 v1903 x64.

The “_CONTEXT” structure starts at offset +0x0e0 from the beginning “_KPROCESSOR_STATE” structure for Windows version 1607 while for the latest Windows versions it starts at offset +0x0f0 from the beginning of “_KPROCESSOR_STATE” structure. Detailed output of the “_KPROCESSOR_STATE” members could be found in Appendix 8. Comparing the fields and offsets of “_KPROCESSOR_STATE” across the different versions could be found in Appendix 9. we found that the “_KPROCESSOR_STATE” fields names and offsets are the same for the three latest Windows versions and that the only difference between them and version 1607, is the two fields “MsrFsBase” and “SpecialPadding0” added to the new versions at the end of “_KSPECIAL_REGISTERS” structure.

Table 11 illustrates a part of the content of the “_KPROCESSOR_STATE” structure. The mentioned offset is considered from the beginning of the hibernation file.

Table 11. Part of "_KPROCESSOR_STATE" datagram v1809, 1903, 1909 x64.

	00	01	02	03	04	05	06	07	08	09	A	B	C	D	E	F
0x1000	SpecialRegisters.Cr0								SpecialRegisters.Cr1							
0x1010	SpecialRegisters.Cr3								SpecialRegisters.Cr4							
0x1020	SpecialRegisters.KernelDr0								SpecialRegisters.KernelDr1							
0x1030	SpecialRegisters.KernelDr2								SpecialRegisters.KernelDr3							
0x1040	SpecialRegisters.KernelDr6								SpecialRegisters.KernelDr7							
0x1050	SpecialRegisters.Gdtr.Pad		SpecialRegisters.Gdtr.Pad		SpecialRegisters.Gdtr.Pad		SpecialRegisters.Gdtr.Limit		SpecialRegisters.Gdtr.Base							
0x1060	SpecialRegisters.Idtr.Pad		SpecialRegisters.Idtr.Pad		SpecialRegisters.Idtr.Pad		SpecialRegisters.Idtr.Limit		SpecialRegisters.Idtr.Base							
0x1070	SpecialRegisters.Tr		SpecialRegisters.Ldtr		SpecialRegisters. MxCsr				SpecialRegisters.DebugControl							
0x1080	SpecialRegisters.LastBranchToRip								SpecialRegisters.LastBranchFromRip							
0x1090	SpecialRegisters.LastExceptionToRip								SpecialRegisters.LastExceptionFromRip							
0x10a0	SpecialRegisters.Cr8								SpecialRegisters.MsrGsBase							
0x10b0	SpecialRegisters.MsrGsSwap								SpecialRegisters.MsrStar							
0x10c0	SpecialRegisters.MsrLStar								SpecialRegisters.MsrCStar							
0x10d0	SpecialRegisters.MsrSyscallMask								SpecialRegisters.Xcr0							
0x10e0	SpecialRegisters.MsrFsBase								SpecialRegisters.SpecialPadding0							
0x10f0	ContextFrame.P1Home								ContextFrame.P2Home							
0x1100	ContextFrame.P3Home								ContextFrame.P4Home							
0x1110	ContextFrame.P5Home								ContextFrame.P6Home							
0x1120	ContextFrame. ContextFlags				ContextFrame.MxCsr				ContextFrame.SegCs		ContextFrame.SegDs		ContextFrame.SegEs		ContextFrame.SegFs	
0x1130	ContextFrame.SegGs		ContextFrame.SegSs		ContextFrame. EFlags				ContextFrame.Dr0							
0x1140	ContextFrame.Dr1								ContextFrame.Dr2							
0x1150	ContextFrame.Dr3								ContextFrame.Dr6							
0x1160	ContextFrame.Dr7								ContextFrame.Rax							
0x1170	ContextFrame.Rcx								ContextFrame.Rdx							
0x1180	ContextFrame.Rbx								ContextFrame.Rsp							
0x1190	ContextFrame.Rbp								ContextFrame.Rsi							
0x11a0	ContextFrame.Rdi								ContextFrame.R8							
0x11b0	ContextFrame.R9								ContextFrame.R10							
0x11c0	ContextFrame.R11								ContextFrame.R12							
0x11d0	ContextFrame.R13								ContextFrame.R14							
0x11e0	ContextFrame.R15								ContextFrame.Rip							

From this section, we conclude that the second hibernation file page content was not modified through the latest versions of Windows 10 (v1809, v1903 and v1909). Minor differences were found between the three versions and version 1607. As per we are not sure if any of the decompression tools or the analysis tools extract data from this page, we could not asses the risk of the minor found variance.

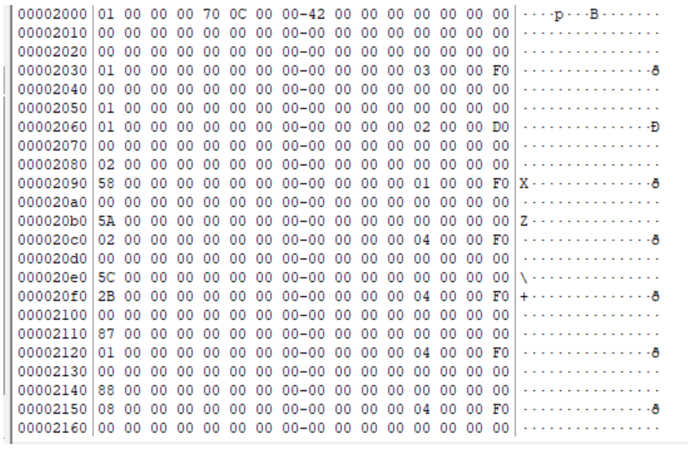
4.3 Test case A analysis

By the end of this section, we would answer the research questions mentioned in section 3.3.1.

4.3.1 Undocumented structures

The structure of the first two pages of the hibernation file was extracted using WinDbg as documented in the previous section. Starting from the third-page file (offset 0x2000), we do not have a known Windows structure name to search for in WinDbg.

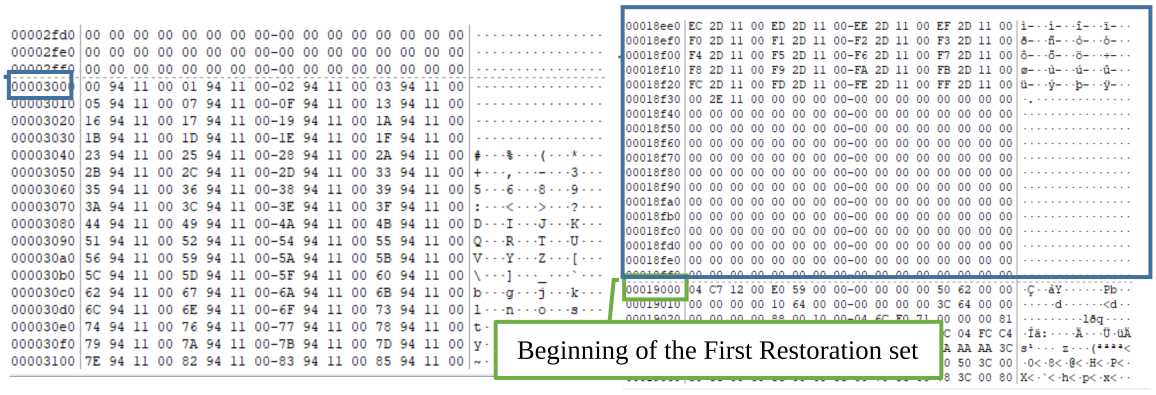
During the analysis of test case A file, we found that following the "_KPROCESSOR_STATE" page, there exist some pages including data, that was not documented in [11], we are not sure what type of data is stored in these pages. Figure 35 shows the first undocumented structure.



```
00002000 01 00 00 00 70 0C 00 00-42 00 00 00 00 00 00 00 ····p···B·······
00002010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002030 01 00 00 00 00 00 00 00-00 00 00 00 03 00 00 F0 ···············ð
00002040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002050 01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002060 01 00 00 00 00 00 00 00-00 00 00 00 02 00 00 D0 ···············Ð
00002070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002080 02 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002090 58 00 00 00 00 00 00 00-00 00 00 00 01 00 00 F0 X··············ð
000020a0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
000020b0 5A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 Z···············
000020c0 02 00 00 00 00 00 00 00-00 00 00 00 04 00 00 F0 ···············ð
000020d0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
000020e0 5C 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 \···············
000020f0 2B 00 00 00 00 00 00 00-00 00 00 00 04 00 00 F0 +··············ð
00002100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002110 87 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002120 01 00 00 00 00 00 00 00-00 00 00 00 04 00 00 F0 ···············ð
00002130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002140 88 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002150 08 00 00 00 00 00 00 00-00 00 00 00 04 00 00 F0 ···············ð
00002160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
```

Figure 35. Part of the third page of the hibernation file, extracted from v1903 x64.

We also found that there exist some data that ends at the page precedes FirstBootRestorePage page. For example, in case the FirstBootRestorePage offset is 0x19000, the first page of this undocumented structure would start at the fourth page (offset 0x3000) and ends at page starting at offset 0x18000. Figure 36 shows the second undocumented structure.



```
00002fd0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002fe0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00002ff0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00003000 00 94 11 00 01 94 11 00-02 94 11 00 03 94 11 00 ················
00003010 05 94 11 00 07 94 11 00-0F 94 11 00 13 94 11 00 ················
00003020 16 94 11 00 17 94 11 00-19 94 11 00 1A 94 11 00 ················
00003030 1B 94 11 00 1D 94 11 00-1E 94 11 00 1F 94 11 00 ················
00003040 23 94 11 00 25 94 11 00-28 94 11 00 2A 94 11 00 #···%···(···*···
00003050 2B 94 11 00 2C 94 11 00-2D 94 11 00 33 94 11 00 +···,···-···3···
00003060 35 94 11 00 36 94 11 00-38 94 11 00 39 94 11 00 5···6···8···9···
00003070 3A 94 11 00 3C 94 11 00-3E 94 11 00 3F 94 11 00 :···<···>···?···
00003080 44 94 11 00 49 94 11 00-4A 94 11 00 4B 94 11 00 D···I···J···K···
00003090 51 94 11 00 52 94 11 00-54 94 11 00 55 94 11 00 Q···R···T···U···
000030a0 56 94 11 00 59 94 11 00-5A 94 11 00 5B 94 11 00 V···Y···Z···[···
000030b0 5C 94 11 00 5D 94 11 00-5F 94 11 00 60 94 11 00 \···]···_···`···
000030c0 62 94 11 00 67 94 11 00-6A 94 11 00 6B 94 11 00 b···g···j···k···
000030d0 6C 94 11 00 6E 94 11 00-6F 94 11 00 73 94 11 00 l···n···o···s···
000030e0 74 94 11 00 76 94 11 00-77 94 11 00 78 94 11 00 t·
000030f0 79 94 11 00 7A 94 11 00-7B 94 11 00 7D 94 11 00 y·
00003100 7E 94 11 00 82 94 11 00-83 94 11 00 85 94 11 00 ~·
```

```
00018ee0 EC 2D 11 00 ED 2D 11 00-EE 2D 11 00 EF 2D 11 00 ì-··í-··î-··ï-··
00018ef0 F0 2D 11 00 F1 2D 11 00-F2 2D 11 00 F3 2D 11 00 ð-··ñ-··ò-··ó-··
00018f00 F4 2D 11 00 F5 2D 11 00-F6 2D 11 00 F7 2D 11 00 ô-··õ-··ö-··÷-··
00018f10 F8 2D 11 00 F9 2D 11 00-FA 2D 11 00 FB 2D 11 00 ø-··ù-··ú-··û-··
00018f20 FC 2D 11 00 FD 2D 11 00-FE 2D 11 00 FF 2D 11 00 ü-··ý-··þ-··ÿ-··
00018f30 00 2E 11 00 00 00 00 00-00 00 00 00 00 00 00 00 ·.··············
00018f40 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018f50 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018f60 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018f70 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018f80 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018f90 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018fa0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018fb0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018fc0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018fd0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018fe0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00018ff0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
00019000 04 C7 12 00 E0 59 00 00-00 00 00 00 50 62 00 00 ·Ç··àY······Pb··
00019010 00 00 00 00 10 64 00 00-00 00 00 00 3C 64 00 00 ·····d······<d··
00019020 00 00 00 00 88 00 10 00-04 6C F0 71 00 00 00 81 ·········lðq····
                                          C 04 FC C4 ·Ìä:····Ä···Ü·üÄ
                                          A AA AA 3C s¹····z···(ªªªª<
                                          0 50 3C 00 ·0<·8<·@<·H<·P<·
                                          8 3C 00 80 X<·`<·h<·p<·x<··
```

Figure 36. Second undocumented structure, extracted from Win10 v1903 x64.

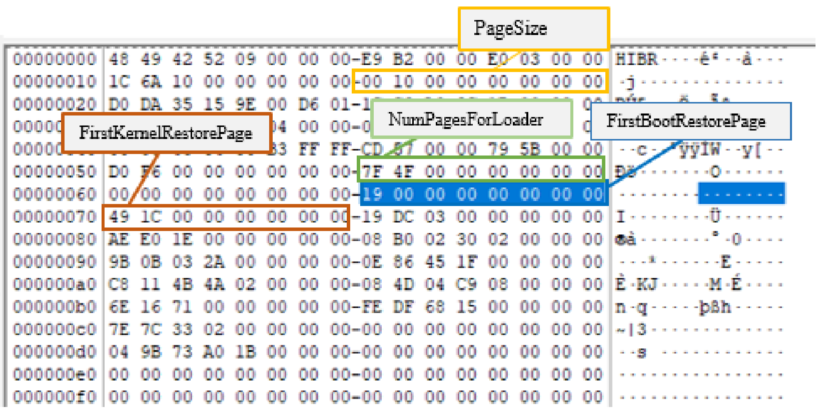
These two undocumented structures are very similar to the pages starting at the same offset (0x2000) in the hibernation file extracted from Windows 7 service pack 1 hibernation file[1]. The second undocumented structure (starting from offset 0x3000) seems to be the same pages described as “FreeMap Pages” [27] in Suiche description for legacy hibernation file. “FreeMap Pages” we described in Suiche documentations as “*ULONG array which looks to be a table of free memory pages*” [27]. In [28], Kleissner mentioned that the third page (offset 0x2000) content as *“The free/reserved memory map starts. It is used to initialize and reload spaces in memory”*. These short sentences are the only available details about these legacy hibernation file pages. The papers did not provide more any screenshots or more details, to confirm that they are talking about the two undocumented structures that we found in Windows 10 hibernation file. We believe that the descriptions of the free memory pages, match the second undocumented structure (starting from the fourth page in Windows 7 SP1 and Windows 10 hibernation files).

4.3.2 Restoration Sets

Referenced to [11], we could extract details about the first restoration set using the hibernation file header. For all the analyzed hibernation files with variable Windows 10 versions and states, we found that the FirstBootRestorePage was always the first used restoration set, as the “FirstSecureRestorePage” value was zero for all the files. Enabling the “Secure Boot” [82] BIOS option did not enable the “FirstSecureRestorePage”. Furthermore, due to the Windows 10 Home edition limitations, we were not able to test the Secure Kernel Mode(SKM) configuration to confirm if this would enable the third restoration set or not.

This section aims to check if the same restoration set structure is still applicable to our target Windows versions and confirm our understanding of the restoration set structures. In the following steps, we used the File_2 previously described in section 3.3.1 of Windows version1903 as an example, to demonstrate in detail the restoration set structure, described earlier in the File Structure section, same steps were applied on other Windows versions. Figure 37 shows the header of the used test hibernation file.

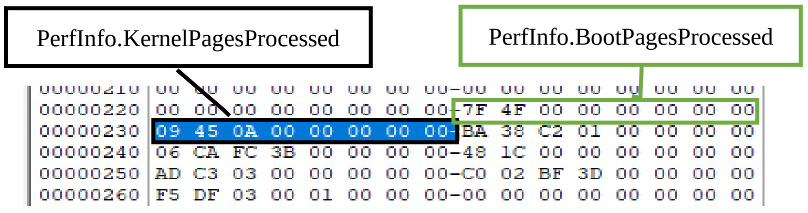
[1] this file we previously used to explain section 2.3.5.1. and was not added to the method section and it was not planned to use it. No other actions required the use of this file, except that step.



```
00000000 48 49 42 52 09 00 00 00-E9 B2 00 00 E0 03 00 00 HIBR····é²··à···
00000010 1C 6A 10 00 00 00 00 00-00 10 00 00 00 00 00 00 ·j··············
00000020 D0 DA 35 15 9E 00 D6 01-1[illegible]
0000[illegible] 4 00 00-0[illegible]
0000[illegible] 3 FF FF-CD 57 00 00 79 5B 00 00 ··c··ÿÿÍW··y[··
00000050 D0 F6 00 00 00 00 00 00-7F 4F 00 00 00 00 00 00 Ðö·······O······
00000060 00 00 00 00 00 00 00 00-19 00 00 00 00 00 00 00 ················
00000070 49 1C 00 00 00 00 00 00-19 DC 03 00 00 00 00 00 I········Ü······
00000080 AE E0 1E 00 00 00 00 00-08 B0 02 30 02 00 00 00 ®à·······°·0····
00000090 9B 0B 03 2A 00 00 00 00-0E 86 45 1F 00 00 00 00 ···*······E·····
000000a0 C8 11 4B 4A 02 00 00 00-08 4D 04 C9 08 00 00 00 È·KJ·····M·É····
000000b0 6E 16 71 00 00 00 00 00-FE DF 68 15 00 00 00 00 n·q·····þßh·····
000000c0 7E 7C 33 02 00 00 00 00-00 00 00 00 00 00 00 00 ~|3·············
000000d0 04 9B 73 A0 1B 00 00 00-00 00 00 00 00 00 00 00 ··s ············
000000e0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
000000f0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
```

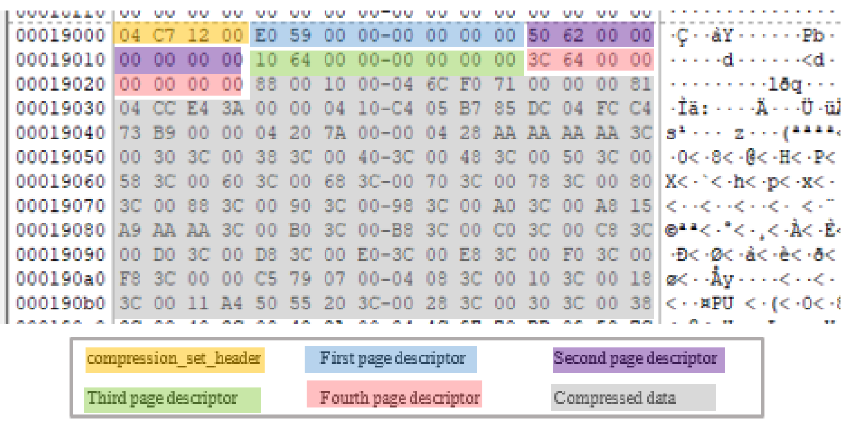
Figure 37. Hibernation file header of a Windows 10 version 1903 x64.

This file contains only two restoration sets. The “FirstBootRestorePage” value is stored in little-endian (value is 0x19). We can calculate the offset of the first restoration set as 0x19 * 0x1000 = 0x19000, where 0x1000 is the page size, which is equivalent to 4096 bytes (“PageSize” entry of the header, is 8 bytes starting from offset 0x18). The total number of pages in this restoration set are stored in the fields “NumPagesForLoader” (8 bytes length, starting from offset 0x058) and “prefinfo.BootPagesProcessed” (8 bytes length, starting from offset 0x228), as shown in figure 40, which equal to 0x4F7F (20351 pages). The first compression set starts at the beginning of the first restoration set (offset 0x19000 in our case), with the details shown in Figure 38.



```
00000210 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00000220 00 00 00 00 00 00 00 00-7F 4F 00 00 00 00 00 00
00000230 09 45 0A 00 00 00 00 00-BA 38 C2 01 00 00 00 00
00000240 06 CA FC 3B 00 00 00 00-48 1C 00 00 00 00 00 00
00000250 AD C3 03 00 00 00 00 00-C0 02 BF 3D 00 00 00 00
00000260 F5 DF 03 00 01 00 00 00-00 00 00 00 00 00 00 00
```

Figure 38. Fields total number of pages in each restoration set.



```
00019000 04 C7 12 00 E0 59 00 00-00 00 00 00 50 62 00 00 ·Ç··àY······Pb··
00019010 00 00 00 00 10 64 00 00-00 00 00 00 3C 64 00 00 ·····d······<d··
00019020 00 00 00 00 88 00 10 00-04 6C F0 71 00 00 00 81 ··········lðq···
00019030 04 CC E4 3A 00 00 04 10-C4 05 B7 85 DC 04 FC C4 ·Ìä:····Ä···Ü·üÄ
00019040 73 B9 00 00 04 20 7A 00-00 04 28 AA AA AA AA 3C s¹··· z···(ªªªª<
00019050 00 30 3C 00 38 3C 00 40-3C 00 48 3C 00 50 3C 00 ·0<·8<·@<·H<·P<·
00019060 58 3C 00 60 3C 00 68 3C-00 70 3C 00 78 3C 00 80 X<·`<·h<·p<·x<··
00019070 3C 00 88 3C 00 90 3C 00-98 3C 00 A0 3C 00 A8 15 <··<··<··<· <·¨·
00019080 A9 AA AA 3C 00 B0 3C 00-B8 3C 00 C0 3C 00 C8 3C ©ªª<·°<·¸<·À<·È<
00019090 00 D0 3C 00 D8 3C 00 E0-3C 00 E8 3C 00 F0 3C 00 ·Ð<·Ø<·à<·è<·ð<·
000190a0 F8 3C 00 00 C5 79 07 00-04 08 3C 00 10 3C 00 18 ø<··Åy····<··<··
000190b0 3C 00 11 A4 50 55 20 3C-00 28 3C 00 30 3C 00 38 <··¤PU <·(<·0<·8
```

Figure 39.The first page of the first Restoration set.

The First "compression_set_header" value is 0x12C704 (binary value is 0 **0** 00 0000 0001 0010 1100 0111 0000 0100), where the data fields are shown in Table 12.

Table 12. Test hibernation file - first compression_set_header analysis.

Value	HuffmanCompressed	Unknown	SizeOfCompressionData	NumberOfDescs
Hex	0	**0**	00 0000 0001 0010 1100 0111	0000 0100
Integer			4807	4

We have four "page descriptors" following the "compression_set_header" (each descriptor is 8 bytes as this system is x64). A compressed data size 4807 bytes (SizeOfCompressionData) follow the last page descriptor.

From Table 13, we can see that the first compression set contains 16 pages, which is the maximum number of pages in each set; each page is 4KB of uncompressed data.

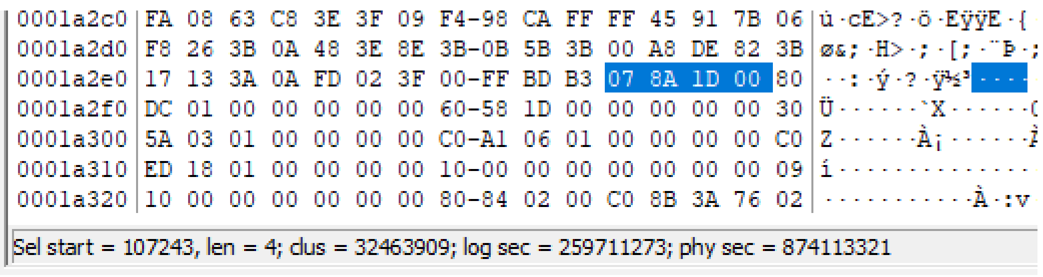
Table 13. The number of pages in each page descriptor contiguous set.

	The binary value of the page descriptor	Physical page address of the first page (PageNum * 0x1000)	Number of contiguous pages in the set (NumPages + 1)
First-page descriptor	010110011110 0000	0x59e000	1
Second-page descriptor	11000100101 0000	0x625000	1
Third-page descriptor	011001000001 0000	0x641000	1
Fourth-page descriptor	011001000011 1100	0x643000	13
Total number of pages in the compression set			**16**

These pages are compressed into 4807 bytes. The compression method used in that compression set is the Plain LZ277 Xpress algorithm (as the HuffmanCompressed flag is set to zero).

Taking an offset from the beginning of the compressed data (offset 0x19024) with the length of 4807 bytes would lead us to the second compression set header.

Second compression set header is shown in Figure 32, and it has the same structure previously described.



```
0001a2c0 FA 08 63 C8 3E 3F 09 F4-98 CA FF FF 45 91 7B 06 ù·cÈ>?·ô·ÊÿÿE·{
0001a2d0 F8 26 3B 0A 48 3E 8E 3B-0B 5B 3B 00 A8 DE 82 3B ø&;·H>·;·[;·¨Þ·;
0001a2e0 17 13 3A 0A FD 02 3F 00-FF BD B3 07 8A 1D 00 80 ··:·ý·?·ÿ½³····
0001a2f0 DC 01 00 00 00 00 00 60-58 1D 00 00 00 00 00 30 Ü······`X······0
0001a300 5A 03 01 00 00 00 00 C0-A1 06 01 00 00 00 00 C0 Z······À¡······À
0001a310 ED 18 01 00 00 00 00 10-00 00 00 00 00 00 00 09 í···············
0001a320 10 00 00 00 00 00 00 80-84 02 00 C0 8B 3A 76 02 ············À·:v

Sel start = 107243, len = 4; clus = 32463909; log sec = 259711273; phy sec = 874113321
```

Figure 40. Second compression set header of Windows 10 version 1903 x64.

The second restoration set starting at the offset 0x1C49000 (FirstKernelRestorePage * 0x1000) from the beginning of the hibernation file, and it contains PerfInfo.KernelPagesProcessed pages, which value is 673033 pages (0x0A4509) in our case. The second restoration set has the same structure as the first restoration set.

4.3.3 Understanding the file structure with the support of Hibr2Bin

We took the chance of availability of Hibr2Bin source code and tried to understand the code, and mapping the definitions naming used in the code to the description provided in [11]. We used Visual Studio to debug the code and added many "wprintf" commands to print the output of some variable to confirm that we understand the structure of the file correctly and check if Hibr2Bin is using the same structure proposed by [11] in modern hibernation file analysis, or it is using its methodology instead.

We found that hiber2bin is basing its analysis for modern hibernation file on uncompressing two restoration sets. The code searches for the first restoration set by searching for a pattern of zeros using the function GetInitialOffset(). This function does not use the content of FirstBootRestorePage offset to restore the first restoration set. Instead, it searches for 32 bytes of zeroes starting from the 6th memory page (offset 0x5000) from the beginning of modern hibernation file. This methodology seems to be functioning correctly, as our manual analysis for many hibernation files, shows that the FirstBootRestorePage address was always coming after the 6th memory page and there exist several "lines" of zero bytes proceeding the first restoration set. Hibr2Bin restores the second restoration set using the function GetFirstKernelRestorePage(), which is based on the FirstKernelRestorePage entry in the file header. With this configuration, we wonder if, in case of activation of the third restoration set in a hibernation file, Hibr2Bin would be able to convert this restoration set or not.

As explained earlier, each restoration set contains one or more compression sets. To confirm our understanding of the code, we used the same hibernation file used in the manual analysis section. We used the same test file used in section 4.3.2, we processed the same file with the modified build version of Hibr2Bin, as our modifications provide more details about the functioning of the tool. We confirmed that the hash of the decompressed file is the same through the modified and unmodified version of Hibr2Bin, to confirm that we didn’t modify any functional code by mistake. As we found that Hibr2Bin code treats all Windows 10 versions the same way, it was not required to repeat the same test on each Windows version, especially that the target of this test is confirming our understanding of the code functioning. Our test file contains more than 42 000 compression sets, and We found that the Hibr2Bin results match our manual result for the first ten compression sets of each restoration set. This result confirms our understanding of these structures and also confirms that the same restoration sets structures are still used in the latest versions of the hibernation file. We found that the structure “_compression_set_header” is named as “_PO_MEMORY_RANGE_TABLE64_NT62” in Hibr2bin code. This structure is composed of the three variables documented in [11], with different naming conventions, so “NumberOfDescs” is named as “RangeCount” in Suiche’s code, “SizeOfCompressedData” is named as “CompressedSize” and “HuffmanCompressed” variable is “CompressMethod”. The structure size match for both [11] and Hibr2Bin code, while there is a size mismatch for the Compression method variable, as it was mentioned as one byte proceeded by an unknown byte in [11], while Hibr2Bin code defines it as two bytes with four values. Hiber2Bin define the values of compression methods as “XpressFast = 0, XpressMax = 1, XpressHuffFast = 2 and XpressHuffMax = 3”. These values might be confusing, as Microsoft has defined only three variants for the Xpress compression [83], that are: Plain LZ77 (the fastest variant), LZ77+Huffman, and LZNT1.

We also noticed that the structure “_page_descriptor” is named as “_PO_MEMORY_RANGE64_NT62” in Hibr2Bin code, it contains two variables: PageCount (named as “NumPages” in [11]) and StartPage (named as “PageNum”). while there exists a size mismatch for the “StartPage” between Hiber2Bin code and the definition mentioned in [11]. The value of “StartPage” is defined as 28 bits in Hibr2Bin code instead of 60 bits, this size matches a 32-bit system, not a 64-bit system. While this might be a typo error and will not affect the conversion of the file unless the hibernation file is larger than 1 TB.

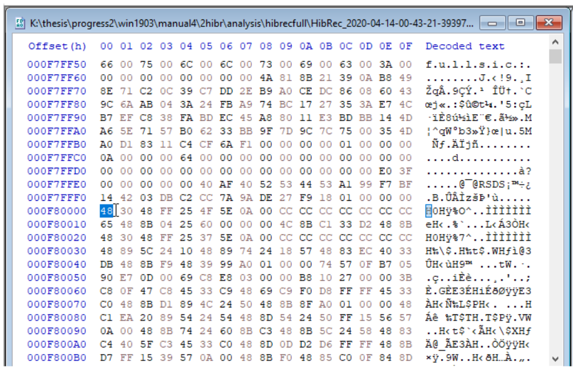
4.3.3.1 Comparing Conversion tools

As a part of test case A, we proceeded with the decompression of the extracted hibernation files using both Hibr2Bin and Hibernation Recon to spot if there are any differences between the output of these two tools.

For our knowledge at this stage, there is no current free tool having the capability to analyze the hibernation file in its original layout. The difference between the outputs of both conversion tools might be related to malfunctioning in one or both tools. If any differences spotted between the output files, this might impact the results of the analysis using free tools like volatility, as in such case, we would expect different outputs between the decompressed files of the two tools. While if both tools have the same output decompressed binary file, this would at least confirm that both tools have the same definition of the modern hibernation file structure.

Comparing the MD5 and SHA256 hash values of output files of both tools reveal that the two files contents are not the same. Moreover, when doing a manual comparison between the content of both files using HxD, we can find that there are too many differences between the content of both output files. To find the reason for such differences, we used the same v1903 file used in section 4.3.2 (File_2). We compared the two decompressed versions of File_2 using HxD. We tracked the compression set data using the logs of the modified version of Hibr2bin. Before starting our tests, we first confirmed that the output decompressed file using Hibr2Bin modified version has the same Hash values as the output file of Hibr2bin version downloaded from Comae site, and this point was confirmed.

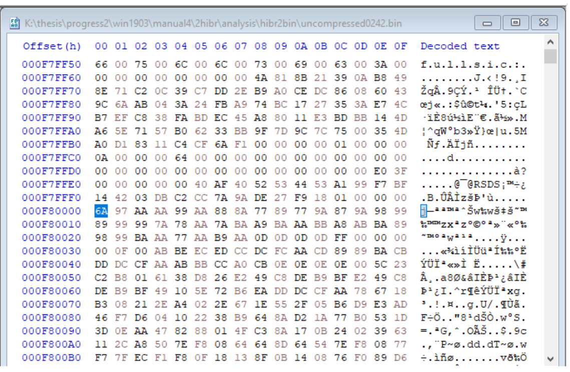
Using HxD comparison, the first difference found between the decompressed output files of Hibr2Bin and Hibernation recon was found at offset 0xF800000. All pages content before v1903 were identical. Figure 41 shows a comparison between the two output files.



K:\thesis\progress2\win1903\manual4\2hibr\analysis\hibrecfull\HibRec_2020-04-14-00-43-21-39397...

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000F7FF50  66 00 75 00 6C 00 6C 00 73 00 69 00 63 00 3A 00  f.u.l.l.s.i.c.:.
000F7FF60  00 00 00 00 00 00 00 00 4A 81 8B 21 39 0A B8 49  ........J.‹!9.¸I
000F7FF70  8E 71 C2 0C 39 C7 DD 2E B9 A0 CE DC 86 08 60 43  ŽqÂ.9ÇÝ.¹ ÎÜ†.`C
000F7FF80  9C 6A AB 04 3A 24 FB A9 74 BC 17 27 35 3A E7 4C  œj«.:$û©t¼.'5:çL
000F7FF90  B7 EF C8 38 FA BD EC 45 A8 80 11 E3 BD BB 14 4D  ·ïÈ8ú½ìE¨€.ã½».M
000F7FFA0  A6 5E 71 57 B0 62 33 BB 9F 7D 9C 7C 75 00 35 4D  ¦^qW°b3»Ÿ}œ|u.5M
000F7FFB0  A0 D1 83 11 C4 CF 6A F1 00 00 00 00 01 00 00 00   Ñƒ.ÄÏjñ........
000F7FFC0  0A 00 00 00 64 00 00 00 00 00 00 00 00 00 00 00  ....d...........
000F7FFD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 3F  ..............à?
000F7FFE0  00 00 00 00 00 40 AF 40 52 53 44 53 A1 99 F7 BF  .....@¯@RSDS¡™÷¿
000F7FFF0  14 42 03 DB C2 CC 7A 9A DE 27 F9 18 01 00 00 00  .B.ÛÂÌzšÞ'ù.....
000F80000  48 30 48 FF 25 4F 5E 0A 00 CC CC CC CC CC CC CC  H0Hÿ%O^..ÌÌÌÌÌÌÌ
000F80010  65 48 8B 04 25 60 00 00 00 4C 8B C1 33 D2 48 8B  eH‹.%`...L‹Á3ÒH‹
000F80020  48 30 48 FF 25 37 5E 0A 00 CC CC CC CC CC CC CC  H0Hÿ%7^..ÌÌÌÌÌÌÌ
000F80030  48 89 5C 24 10 48 89 74 24 18 57 48 83 EC 40 33  H‰\$.H‰t$.WHƒì@3
000F80040  DB 48 8B F9 48 39 99 A0 01 00 00 74 57 0F B7 05  ÛH‹ùH9™ ...tW.·.
000F80050  90 E7 0D 00 69 C8 E8 03 00 00 B8 10 27 00 00 3B  .ç..iÈè...¸.'..;
000F80060  C8 0F 47 C8 45 33 C9 48 69 C9 F0 D8 FF FF 45 33  È.GÈE3ÉHiÉðØÿÿE3
000F80070  C0 48 8B D1 89 4C 24 50 48 8B 8F A0 01 00 00 48  ÀH‹Ñ‰L$PH‹. ...H
000F80080  C1 EA 20 89 54 24 54 48 8D 54 24 50 FF 15 56 57  Áê ‰T$TH.T$Pÿ.VW
000F80090  0A 00 48 8B 74 24 60 8B C3 48 8B 5C 24 58 48 83  ..H‹t$`‹ÃH‹\$XHƒ
000F800A0  C4 40 5F C3 45 33 C0 48 8D 0D D2 D6 FF FF 48 8B  Ä@_ÃE3ÀH..ÒÖÿÿH‹
000F800B0  D7 FF 15 39 57 0A 00 48 8B F0 48 85 C0 0F 84 8D  ×ÿ.9W..H‹ðH…À.„.
```

(a)



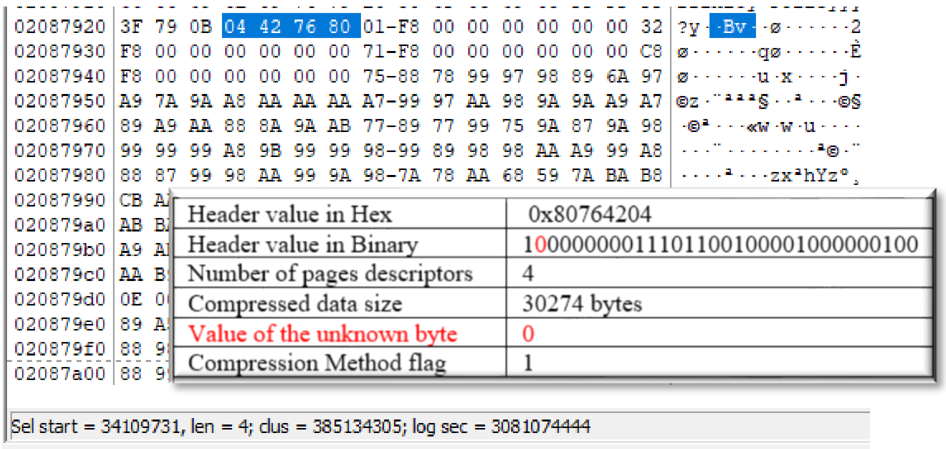
K:\thesis\progress2\win1903\manual4\2hibr\analysis\hibr2bin\uncompressed0242.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000F7FF50  66 00 75 00 6C 00 6C 00 73 00 69 00 63 00 3A 00  f.u.l.l.s.i.c.:.
000F7FF60  00 00 00 00 00 00 00 00 4A 81 8B 21 39 0A B8 49  ........J.‹!9.¸I
000F7FF70  8E 71 C2 0C 39 C7 DD 2E B9 A0 CE DC 86 08 60 43  ŽqÂ.9ÇÝ.¹ ÎÜ†.`C
000F7FF80  9C 6A AB 04 3A 24 FB A9 74 BC 17 27 35 3A E7 4C  œj«.:$û©t¼.'5:çL
000F7FF90  B7 EF C8 38 FA BD EC 45 A8 80 11 E3 BD BB 14 4D  ·ïÈ8ú½ìE¨€.ã½».M
000F7FFA0  A6 5E 71 57 B0 62 33 BB 9F 7D 9C 7C 75 00 35 4D  ¦^qW°b3»Ÿ}œ|u.5M
000F7FFB0  A0 D1 83 11 C4 CF 6A F1 00 00 00 00 01 00 00 00   Ñƒ.ÄÏjñ........
000F7FFC0  0A 00 00 00 64 00 00 00 00 00 00 00 00 00 00 00  ....d...........
000F7FFD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 3F  ..............à?
000F7FFE0  00 00 00 00 00 40 AF 40 52 53 44 53 A1 99 F7 BF  .....@¯@RSDS¡™÷¿
000F7FFF0  14 42 03 DB C2 CC 7A 9A DE 27 F9 18 01 00 00 00  .B.ÛÂÌzšÞ'ù.....
000F80000  6A 97 AA AA 99 AA 88 8A 77 89 77 9A 87 9A 98 99  j—ªª™ªˆŠw‰wš‡š˜™
000F80010  89 99 99 7A 78 AA 7A BA A9 BA AA BB A8 AB BA 89  ‰™™zxªzº©ºª»¨«º‰
000F80020  98 99 BA AA 77 AA B9 AA 0D 0D 0D 0D FF 00 00 00  ˜™ºªwª¹ª....ÿ...
000F80030  00 0F 00 AB BE EC ED CC DC FC AA CD 89 89 BA CB  ...«¾ìíÌÜüªÍ‰‰ºË
000F80040  DD DC CF AA AB BB CC A0 CB 0E 0E 0E 0E 00 5C 23  ÝÜÏª«»Ì Ë.....\#
000F80050  C2 B8 01 61 38 D8 26 E2 49 C8 DE B9 BF E2 49 C8  Â¸.a8Ø&âIÈÞ¹¿âIÈ
000F80060  DE B9 BF 49 10 5E 72 B6 EA DD DC CF AA 78 67 18  Þ¹¿I.^r¶êÝÜÏªxg.
000F80070  B3 08 21 2E A4 02 2E 67 1E 55 2F 05 B6 D9 E3 AD  ³.!.¤..g.U/.¶Ùã.
000F80080  46 F7 D6 04 10 22 38 B9 64 8A D2 1A 77 B0 53 1D  F÷Ö.."8¹dŠÒ.w°S.
000F80090  3D 0E AA 47 82 88 01 4F C3 8A 17 0B 24 02 39 63  =.ªG‚ˆ.OÃŠ..$.9c
000F800A0  11 2C A8 50 7E F8 08 64 64 8D 64 54 7E F8 08 77  .,¨P~ø.dd.dT~ø.w
000F800B0  F7 7F EC F1 F8 0F 18 13 8F 0B 14 08 76 F0 89 D6  ÷.ìñø.......vð‰Ö
```

(b)

Figure 41. Output files of Hibernation Recon (a) and Hibr2Bin (b).

We searched for the page starting at offset 0xF800000 in the compression set logs printed by the modified Hibr2Bin code, to find out what might be different in that compression set. It was found that these data are related to the compression set starting at offset 0x2087923 from File_2 hibernation file. We performed a quick manual analysis of the compression set header as shown in Figure 42.



```
02087920  3F 79 0B 04 42 76 80 01-F8 00 00 00 00 00 00 32  ?y·Bv·ø······2
02087930  F8 00 00 00 00 00 00 71-F8 00 00 00 00 00 00 C8  ø······qø·····È
02087940  F8 00 00 00 00 00 00 75-88 78 99 97 98 89 6A 97  ø······u·x···j·
02087950  A9 7A 9A A8 AA AA AA A7-99 97 AA 98 9A 9A A9 A7  ©z·¨ªªª§··ª···©§
02087960  89 A9 AA 88 8A 9A AB 77-89 77 99 75 9A 87 9A 98  ·©ª···«w·w·u····
02087970  99 99 99 A8 9B 99 99 98-99 89 98 98 AA A9 99 A8  ···¨········ª©·¨
02087980  88 87 99 98 AA 99 9A 98-7A 78 AA 68 59 7A BA B8  ····ª···zxªhYzº¸
02087990  CB A
020879a0  AB B
020879b0  A9 A
020879c0  AA B
020879d0  0E 0
020879e0  89 A
020879f0  88 9
02087a00  88 9
```

Header value in Hex	0x80764204
Header value in Binary	10000000011101100100001000000100
Number of pages descriptors	4
Compressed data size	30274 bytes
Value of the unknown byte	0
Compression Method flag	1

Sel start = 34109731, len = 4; clus = 385134305; log sec = 3081074444

Figure 42. Compression set entries extracted from hiberfil.sys.

We found that the reason for the value mismatch of decompressed pages by both programs might be caused by the compression method flag length definition. When referring this value to our reference paper [11], the compression flag is only a single bit, and in this case, it is set to "1", which indicates that the data is compressed by LZ77+Huffman XPRESS algorithm variant as per [11]. While Hibr2Bin code considers two bytes for the same flag, so as per their code definition this set is compressed by what they named in the code as "XpressHuffFast". When checking Hibr2Bin code, and compare it to the official documentation of XPRESS algorithm [83], we understand that the Plain LZ77 variant is decompressed by the function

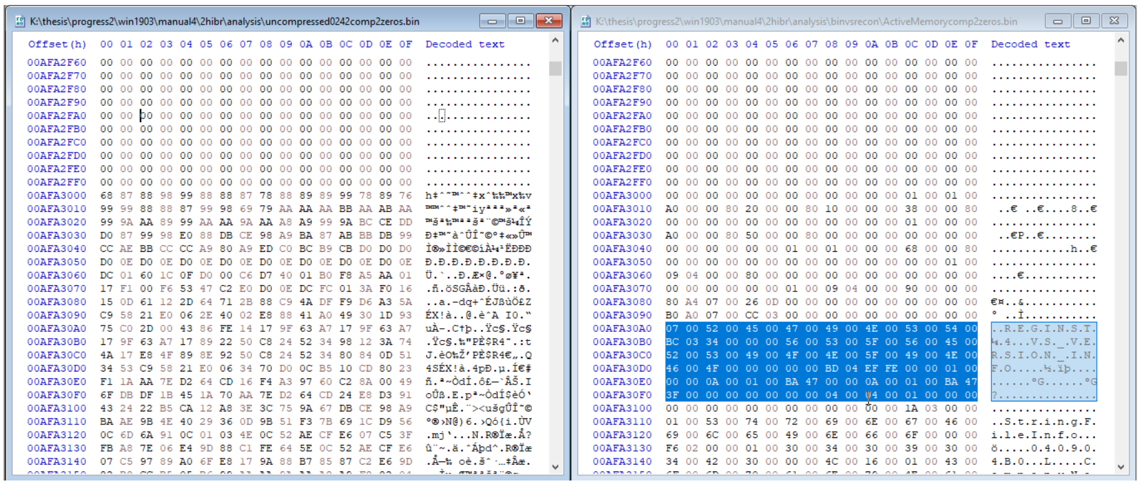
“CompressedMemoryBlock::Xpress_Decompress”, while we are not sure how the code decompresses the LZ77+Huffman variant, as we cannot find the same decompressor code mentioned by Microsoft [83] in Hibr2Bin code.

We created a modified version of each decompressed file and zeroed the full content of compression sets that we manually analyzed to eliminate it from the HxD comparison. We compared the decompressed data of 20 compression sets having a HuffmanCompressed flag set to one, which is related to data compression type LZ77+Huffman as per our reference page [11] and the unknown byte is either “0” or “1” (this would match compression definitions “XpressHuffFast” and “XpressHuffMax” in Hibr2Bin code). We found a data mismatch between decompressed binary files by Hibernation Recon and Hibr2Bin for all the chosen compression sets. The interesting was that the first mismatch found in the files was related to the first compression set with the HuffmanCompressed flag set to 1.

We compared 20 other compression sets data with HuffmanCompressed flag 0, which is related to data compressed with Plain LZ77 variant as per our reference paper [11], and the unknown byte is either “0” or “1” (this would match the compression definitions XpressFast and XpressMax in Hibr2Bin code). All the chosen sets were identically decompressed by both programs, while the compression types found in the logs of decompressed hibernation files using Hibr2bin were only related to what they name “XpressFast” compression type. No compression using “XpressMax” was found in any of the analyzed files using this compiled version of Hibr2Bin.

We proceeded with similar quick tests for other File_4 extracted from the other Windows versions, and the results of our tests were always the same. The decompression of Plain LZ77 is identical for both decompressed files, while LZ77+Huffman decompressed sets data mismatch. The Plain LZ77 compression was the most used compression variant in the files we tested. When calculating the ratio of usage of HuffmanCompressed, we found in our Windows version 1903 test file the ratio was 57% for Plain LZ77 and 43% for the LZ77+Huffman variant, but these numbers vary from a file to another (in other files the ration of usage of Plain LZ77 was 66%).

When looking in details into the content of decompressed mismatch data, we were able to read information from the Hibernation Recon output file for some decompressed pages that had HuffmanCompressed flag enabled. Figure 43 is an example of a decompressed page that initially had HuffmanCompressed flag enabled and the value of “unknown” byte is “1”.



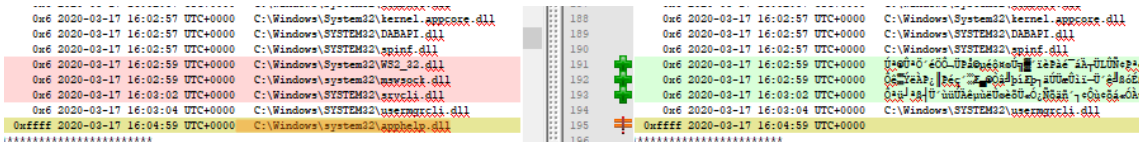
```
K:\thesis\progress2\win1903\manual4\2hibr\analysis\uncompressed0242comp2zeros.bin

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00AFA2F60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2F70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2F80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2F90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FA0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FB0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FC0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA3000  68 87 88 98 99 88 88 87 78 88 89 89 99 78 89 76  h‡ˆ˜™ˆˆ‡xˆ‰‰™x‰v
00AFA3010  99 99 88 88 87 99 98 69 79 AA AA AA BB AA AB AA  ™™ˆˆ‡™˜iyªªª»ª«ª
00AFA3020  99 9A AA 89 99 AA AA 9A AA A8 A9 99 9A BC CE DD  ™šª‰™ªªšª¨©™š¼ÎÝ
00AFA3030  D0 87 99 98 E0 88 DB CE 98 A9 BA 87 AB BB DB 99  Ð‡™˜àˆÛÎ˜©º‡«»Û™
00AFA3040  CC AE BB CC CC A9 80 A9 ED C0 BC B9 CB D0 D0 D0  Ì®»ÌÌ©€©íÀ¼¹ËÐÐÐ
00AFA3050  D0 0E D0 0E D0 0E D0 0E D0 0E D0 0E D0 0E D0 0E  Ð.Ð.Ð.Ð.Ð.Ð.Ð.Ð.
00AFA3060  DC 01 60 1C 0F D0 00 C6 D7 40 01 B0 F8 A5 AA 01  Ü.`..Ð.Æ×@.°ø¥ª.
00AFA3070  17 F1 00 F6 53 47 C2 E0 D0 0E DC FC 01 3A F0 16  .ñ.öSGÂàÐ.Üü.:ð.
00AFA3080  15 0D 61 12 2D 64 71 2B 88 C9 4A DF F9 D6 A3 5A  ..a.-dq+ˆÉJßùÖ£Z
00AFA3090  C9 58 21 E0 06 2E 40 02 E8 88 41 A0 49 30 1D 93  ÉX!à..@.èˆA I0.“
00AFA30A0  75 C0 2D 00 43 86 FE 14 17 9F 63 A7 17 9F 63 A7  uÀ-.C†þ..Ÿc§.Ÿc§
00AFA30B0  17 9F 63 A7 17 89 22 50 C8 24 52 34 98 12 3A 74  .Ÿc§.‰"PÈ$R4˜.:t
00AFA30C0  4A 17 E8 4F 89 8E 92 50 C8 24 52 34 80 84 0D 51  J.èO‰Ž’PÈ$R4€„.Q
00AFA30D0  34 53 C9 58 21 E0 06 34 70 D0 0C B5 10 CD 80 23  4SÉX!à.4pÐ.µ.Í€#
00AFA30E0  F1 1A AA 7E D2 64 CD 16 F4 A3 97 60 C2 8A 00 49  ñ.ª~ÒdÍ.ô£—`ÂŠ.I
00AFA30F0  6F DB DF 1B 45 1A 70 AA 7E D2 64 CD 24 E8 D3 91  oÛß.E.pª~ÒdÍ$èÓ‘
00AFA3100  43 24 22 B5 CA 12 A8 3E 3C 75 9A 67 DB CE 98 A9  C$"µÊ.¨><ušgÛÎ˜©
00AFA3110  BA AE 9B 4E 40 29 36 0D 9B 51 F3 7B 69 1C D9 56  º®›N@)6.›Qó{i.ÙV
00AFA3120  0C 6D 6A 91 0C 01 03 4E 0C 52 AE CF E6 07 C5 3F  .mj‘...N.R®Ïæ.Å?
00AFA3130  FB A8 7E 06 E4 9D 88 C1 FE 64 5E 0C 52 AE CF E6  û¨~.ä.ˆÁþd^.R®Ïæ
00AFA3140  07 C5 97 89 A0 6F E8 17 9A 88 B7 85 87 C2 E6 9D  .Å—‰ oè.šˆ·…‡Âæ.
```

```
K:\thesis\progress2\win1903\manual4\2hibr\analysis\binvsrecon\ActiveMemorycomp2zeros.bin

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00AFA2F60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2F70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2F80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2F90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FA0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FB0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FC0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA2FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00AFA3000  00 00 00 00 00 00 00 00 00 00 00 00 01 00 01 00  ................
00AFA3010  A0 00 00 80 20 00 00 80 10 00 00 00 38 00 00 80   ..€ ..€....8..€
00AFA3020  00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
00AFA3030  A0 00 00 80 50 00 00 80 00 00 00 00 00 00 00 00   ..€P..€........
00AFA3040  00 00 00 00 00 00 01 00 01 00 00 00 68 00 00 80  ............h..€
00AFA3050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00  ................
00AFA3060  09 04 00 00 80 00 00 00 00 00 00 00 00 00 00 00  ....€...........
00AFA3070  00 00 00 00 00 00 01 00 09 04 00 00 90 00 00 00  ................
00AFA3080  80 A4 07 00 26 0D 00 00 00 00 00 00 00 00 00 00  €¤..&...........
00AFA3090  B0 A0 07 00 CC 03 00 00 00 00 00 00 00 00 00 00  ° ..Ì...........
00AFA30A0  07 00 52 00 45 00 47 00 49 00 4E 00 53 00 54 00  ..R.E.G.I.N.S.T.
00AFA30B0  BC 03 34 00 00 00 56 00 53 00 5F 00 56 00 45 00  ¼.4...V.S._.V.E.
00AFA30C0  52 00 53 00 49 00 4F 00 4E 00 5F 00 49 00 4E 00  R.S.I.O.N._.I.N.
00AFA30D0  46 00 4F 00 00 00 00 00 BD 04 EF FE 00 00 01 00  F.O.....½.ïþ....
00AFA30E0  00 00 0A 00 01 00 BA 47 00 00 0A 00 01 00 BA 47  ......ºG......ºG
00AFA30F0  3F 00 00 00 00 00 00 00 04 00 04 00 01 00 00 00  ?...............
00AFA3100  00 00 00 00 00 00 00 00 00 00 00 00 1A 03 00 00  ................
00AFA3110  01 00 53 00 74 00 72 00 69 00 6E 00 67 00 46 00  ..S.t.r.i.n.g.F.
00AFA3120  69 00 6C 00 65 00 49 00 6E 00 66 00 6F 00 00 00  i.l.e.I.n.f.o...
00AFA3130  F6 02 00 00 01 00 30 00 34 00 30 00 39 00 30 00  ö.....0.4.0.9.0.
00AFA3140  34 00 42 00 30 00 00 00 4C 00 16 00 01 00 43 00  4.B.0...L.....C.
```

(a) (b)

Figure 43. Decompressed data using Hibr2Bin (a), Hibernation Recon (b).

As a quick test, we analyzed both decompressed files using Volatility. We found that The results are not the same for many commands output. For example, the output of the Volatility command “dlllist” shows missing and corrupted information in the output file of Hibr2Bin as shown in Figure 44.



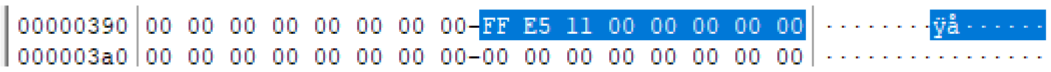
```
   0x6 2020-03-17 16:02:57 UTC+0000   C:\Windows\System32\kernel.appcore.dll
   0x6 2020-03-17 16:02:57 UTC+0000   C:\Windows\SYSTEM32\DABAPI.dll
   0x6 2020-03-17 16:02:57 UTC+0000   C:\Windows\SYSTEM32\spinf.dll
   0x6 2020-03-17 16:02:59 UTC+0000   C:\Windows\System32\WS2_32.dll
   0x6 2020-03-17 16:02:59 UTC+0000   C:\Windows\system32\mswsock.dll
   0x6 2020-03-17 16:03:02 UTC+0000   C:\Windows\SYSTEM32\srvcli.dll
   0x6 2020-03-17 16:03:04 UTC+0000   C:\Windows\SYSTEM32\usermgrcli.dll
0xffff 2020-03-17 16:04:59 UTC+0000   C:\Windows\system32\apphelp.dll
```

```
   0x6 2020-03-17 16:02:57 UTC+0000   C:\Windows\System32\kernel.appcore.dll
   0x6 2020-03-17 16:02:57 UTC+0000   C:\Windows\SYSTEM32\DABAPI.dll
   0x6 2020-03-17 16:02:57 UTC+0000   C:\Windows\SYSTEM32\spinf.dll
   0x6 2020-03-17 16:02:59 UTC+0000   [illegible]
   0x6 2020-03-17 16:02:59 UTC+0000   [illegible]
   0x6 2020-03-17 16:03:02 UTC+0000   [illegible]
   0x6 2020-03-17 16:03:04 UTC+0000   C:\Windows\SYSTEM32\usermgrcli.dll
0xffff 2020-03-17 16:04:59 UTC+0000
```

(a) (b)

Figure 44. The output file of Hibernation Recon (a), output file of Hibr2Bin (b)

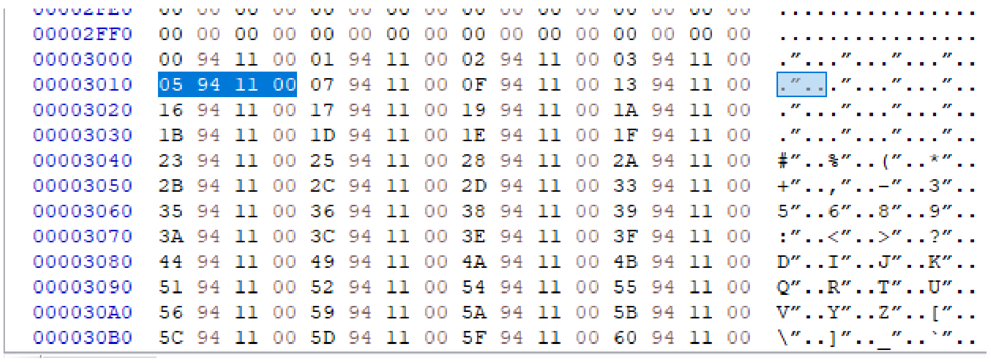
Another note about the differences found between output files of both tools, we noticed that the file length does not match. Comparing the length of both files to the hibernation file header’s entries, we see that Hibernation Recon output file length depends on the value of offset 0x398 (“HighestPhysicalPage”) multiplied by 0x1000 (length of a single page). For example, the last page file of the hibernation file, where “HighestPhysicalPage” is shown in Figure 45, starts at 0x11E5FF000, and the files end at offset 0x11E5FFFFF. We expended this check to many decompressed files and included files extracted from test case B, and C as well, to check a considerable number of the files. We found that the physical address of the last page of the decompressed File_4 using Hibr2Bin was not consistent for 15 verified decompressed files, as the page offset varies from file to another.



```
00000390 00 00 00 00 00 00 00 00-FF E5 11 00 00 00 00 00 ········ÿå······
000003a0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
```

Figure 45. Content of "HighestPhysicalPage" extracted from a test hibernation file.

Interestingly, this analysis helped us to understand the second undocumented structure mentioned in sections 4.3.1. The structure starting at the 4th page (offset 0x3000) is related to free physical memory pages. We verified this by analyzing the content of test case A, File_4 extracted from v1903. We took the first ten pages addresses (value of the content of 4 bytes multiplied by page size, which is 0x1000). We then checked the contents of these memory pages in the decompressed files using Hibr2Bin and Hibernation recon. We found that the pages with the listed addresses in that second undocumented structure are free. Figure 46 shows that page with physical address 0x119405000 is followed by page starting at offset 0x119407000, by checking the relevant decompressed hibernation files we found that the content of memory page starting at offset 0x119406000 contains data. The same procedures were applied on files from other versions, and our understanding seems to be valid.



```
00002FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00003000  00 94 11 00 01 94 11 00 02 94 11 00 03 94 11 00  .”...”...”...”..
00003010  05 94 11 00 07 94 11 00 0F 94 11 00 13 94 11 00  .”...”...”...”..
00003020  16 94 11 00 17 94 11 00 19 94 11 00 1A 94 11 00  .”...”...”...”..
00003030  1B 94 11 00 1D 94 11 00 1E 94 11 00 1F 94 11 00  .”...”...”...”..
00003040  23 94 11 00 25 94 11 00 28 94 11 00 2A 94 11 00  #”..%”..(”..*”..
00003050  2B 94 11 00 2C 94 11 00 2D 94 11 00 33 94 11 00  +”..,”..-”..3”..
00003060  35 94 11 00 36 94 11 00 38 94 11 00 39 94 11 00  5”..6”..8”..9”..
00003070  3A 94 11 00 3C 94 11 00 3E 94 11 00 3F 94 11 00  :”..<”..>”..?”..
00003080  44 94 11 00 49 94 11 00 4A 94 11 00 4B 94 11 00  D”..I”..J”..K”..
00003090  51 94 11 00 52 94 11 00 54 94 11 00 55 94 11 00  Q”..R”..T”..U”..
000030A0  56 94 11 00 59 94 11 00 5A 94 11 00 5B 94 11 00  V”..Y”..Z”..[”..
000030B0  5C 94 11 00 5D 94 11 00 5F 94 11 00 60 94 11 00  \”..]”.._”..`”..
```

Figure 46. Second undocumented structure contains free memory pages addresses.

This free memory pages structure is repeated many times in the decompressed files, so the list of free pages available in the original hibernation file starting from offset 0x3000 is not the full list of free memory pages, it contains just a part of it.

We also shared our notes about the differences found in our user experience between Hibr2Bin and Hibernation Recon. We noticed the Hibernation Recon provide more details and options than Hibr2Bin and wanted to share such details as it might be useful when deciding to use which tool in real life. Details could be found in Appendix 11.

4.4 Test case B analysis

This test case aims to track the characteristics of the hibernation file in different power states and clarifies the changes that might occur to file, based on the modification in power settings. To do such analysis, we collected more than 13 different hibernation files from Windows 10 v1903. The coming paragraphs would clarify the test scenarios and the impact of configuration modification and power state change on the hibernation file content. At each step, we have performed some user activities using Google chrome and libre office to track it in the extracted hibernation file. This test case should answer to the questions mentioned in section 3.3.2.

B_File_1: We collected the hibernation file from a working machine, regardless of the last status before resuming Windows, it was expected that this file would be similar to the B_File_3 and B_File_5. The file size extracted from this step is 1.52 GB (40% of the RAM size). This file header has a signature "WAKE", the first five pages of the file contain data. The content of the file is zeroed starting from offset 0x5000. This file does not contain any data in the restoration sets offsets, which makes the file forensics value almost useless.

B_File_2: To collect this, we first hibernated the system using the hibernation option available in the GUI Windows start menu. It is to be noticed that this GUI option is not enabled by default and that we enabled it once the Windows system was installed. The file collected at this stage was collected offline, using an external bootable hard disk. This file size is 3.82 GB (same size as the total physical memory of the laptop), the file has a signature "HIBR". This file has full content of a hibernation file, it has the same structure discussed in the results of test case A. Quick analysis of the content of this file using tools shows that we could extract data related to user activities from this file.

B_File_3: This file was collected from the system once it resumed from hibernation. When comparing B_File_2 to this file, we found that the only modification on the first page of B_File_3 is a signature "WAKE" instead of the "HIBR" signature used in B_File_2. The content of the file starting from offset 0x1000 to offset 0x4FFF is the same as B_File_2. Starting from offset 0x5000 the content of the file is zeroed, and the file structure is the same as B_File_1. This file does not contain any data in the restoration sets offsets, which makes the file forensics value almost useless.

B_File_4: We kept the same applications that were running once the laptop was resumed from the hibernation state. Then we shutdown the laptop using the GUI option available in the Windows start menu. B_File_4 was collected offline, and the file structure was found the same as B_File_2. To understand what might be the difference between B_File_2 and B_File_4, we processed both files using Magnet Axiom, while this was not a part of the test plan. We found 1234 artifacts were extracted from the file, and this number is less than the number of artifacts extracted by the same program from B_File_2 (3081 artifact). We found some user activities in B_File_4, and it was interesting to find a URL that was visited using Chrome Incognito in B_File_2 and B_File_4, while it was expected to find it only in B_File_2, as B_File_4 should not include any user activities as per Microsoft [15]. This test indicates that the hibernation file collected after a GUI shutdown should be considered in the investigation, although B_File_4 does not contain the maximum number of artifacts, it might contain some of the user activities. We tried to find any characteristics of the file that could differentiate it from B_File_2 without analysis. The only observation is related to the decompressed file size using Hibernation Recon. The size of the decompressed files B_File_2 and B_File_2 was 4.47 GB. On the other hand, the size on disk of decompressed file B_File_2 size was 3.73 GB and B_File_4 it was 3.42 GB. This observation marks a slight size difference between the two decompressed files, while we cannot generalize this observation.

File_5: This file was collected after starting the system; it was collected online. The file content is zeroed starting from offset 0x5000, and it has the same file characteristics of B_File_1 and B_File_3. From this test result, we can conclude that a file with a “WAKE” signature has the same characteristics regardless it resuming from a fast startup or full hibernation power state.

File_6: To collect this file, we first used the command “shutdown /s /hybrid”. The used command is a shutdown command with fast startup option. Windows showed a “sign out” alarm message before shutting down the laptop. B_File_6 was extracted offline. When analyzing the file using Magnet Axiom and Volatility, we did not find any process related to user activity, as Chrome.exe process, for example, while we found some traces about some of the visited URLs.

B_File_6-B: This file was collected online from the system after resuming it from B_File_6 collection step. B_File_6-B has a “WAKE” signature. We would compare the content of B_File_6-B to the content of B_File_7 as we expect that it would be the same content.

B_File_7: To collect B_File_7, we first used the command “shutdown /s”. The hibernation file was collected offline, and FTK imager showed that it has a “WAKE” signature. B_File_7 has the same hash value as B_File_6-B. This proves that a real shutdown power state does not create a hibernation file, while the shutdown using the GUI option or adding “/hybrid” to the CLI command, would turn the machine into the Fast Startup state, not a shutdown. These tests proved those same file behaviors discussed in [11] are still applicable for the latest Windows versions.

B_File_8: Before collecting this hibernation file, we disabled fast startup while keeping the hibernation option from “Control Panel\Hardware and Sound\Power Options\System Settings” -> Define power buttons and turn on password protection. Modifying this option would also turn the value of the registry key “Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Power\HiberbootEnabled” from one to zero. We shutdown the laptop using the GUI option, and then the file was collected offline. The collected file has the signature “WAKE”, and it has the same hash value of B_File_7, this means that this option enables/disable the fast startup capability of Windows 10 OS.

B_File_9: Before collecting this hibernation file, we re-enabled fast startup, and disabled hibernation options from “Control Panel\Hardware and Sound\Power Options\System Settings” -> Define power buttons and turn on password protection. This configuration is the default Windows configuration. By this action, the hibernation option would not be available in the start menu. We ran the command “shutdown /h” then extracted the hibernation file offline. The B_File_9 size is 3.82 GB. When processing the file using Magnet Axiom, we found information related to the user activities that were performed on the laptop before hibernation.

B_File_10: We disabled the hibernation power configuration using the command “powercfg.exe /hibernate off” [5], the hibernation file was not deleted once the command was performed, while the only available sleep mode is shown when running the command “powercfg /a” was Standy (S3). The options “Hibernate”, “Turn on fast start-up” disappeared from “Control Panel\Hardware and Sound\Power Options\System Settings-

> Define power buttons and turn on password protection”. The registry key “Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\HibernateEnabled” turned to zero. We checked the hibernation file offline, after performing a GUI shutdown, to find that the hibernation file was deleted. We resumed the system and re-enabled hibernation; the hibernation file was re-created automatically, and the re-created file has the same content as the “WAKE” file that was available before disabling hibernation.

B_File_11: All the previous steps did not generate any “reduced” hibernation file type, disabling hibernation using the CLI would also disable the fast start-up capability. To create a “reduced” file type we used the command “powercfg /h /type reduced” [15]. Once we ran the command the hibernation file size turned to 20% of the RAM size. We then hibernated the laptop and copied the hibernation file offline. The offline hibernation file size was not impacted by the new configuration and was found as 3.82 GB. The B_File_11 structure is the exact structure of a full hibernation file structure. When analyzing the file using Volatility and Magnet Axiom, we found information related to session 0 (Kernel session), we also found interesting information as Google map URL that contains coordinates.

B_File_12: After resuming Windows after the hibernation done in the last step. We found that the behavior of the reduced “WAKE” hibernation file is the same as described earlier, while its size is 20% of RAM size instead of 40% in the previous cases.

We re-imaged Windows 10 v1903 partition during these tests, the new image of Windows keeps more pages in the “WAKE” hibernation files, while it always stops few pages before the first restoration set data. As an example, in all the files extracted in that section, the first restoration set could be found at offset 0x19000, we found some “WAKE” files that contain data up to offset 0xefff, while none of these “WAKE” files contains restoration sets data.

4.5 Test case C analysis

We enabled HORM using the instructions [33], which requires disable the fast startup. Then we opened few tabs in chrome and created a draft “.txt” document in notepad. The system was then hibernated using the command “shutdown /h”. When we resumed the system from the hibernation state, we found that the hibernation file signature was “HORM”. A “HORM” hibernation file has the same size as the available physical memory, and the file has the same structure as a hibernation file with the “HIBR” signature. When shutdown the laptop, it resumes using the content of the “HORM” hibernation file, and the file hash value is kept the same. The file is overwritten only when a new hibernation occurs. This indicates that all user activities during a session could not be tracked by the hibernation file in case the “HORM” feature is enabled unless the user requests a new hibernation the machine. Enabling this feature is similar to taking a snapshot from a machine then resume the machine every time using the same snapshot. The hibernation file in wake state contains user and kernel session. When we analyze a file with the signature “HORM”, it extracts many types of data, similar to a hibernation file extracted from hibernate state. The collected artifacts are misleading, as they are relevant to the state at which the machine was hibernated.

4.6 Results of the manual analysis

We would discuss in this section the results of the manual tests. The first research question was answered in section 4.1. The hibernation file is created by default by the Windows 10 versions 1809, 1903, 1909.

In section 4.2, we provided an answer to the second research question. Datagrams were created for the first two pages of the file.

In section 4.3, the test case A was used to find out if any undocumented file structures were found manually in the hibernation file, and also to confirm that the same restoration sets layouts defined by [11] as still used for the latest versions of Windows 10. We concluded that some hibernation file structures were not documented in the file layout discussed in [11]. The undocumented structures seem to include two structures. We understand that the undocumented pages and the minor changes in fields names and locations do not impact the conversion of the file, at least to Hibr2Bin while we are not sure how the other tools (Hibernation Recon, Group A tools and Hive Recon) proceed Windows 10 hibernation file. We also concluded that the same restoration set structure is still used to compress hibernation files. In section 4.3.3, using the open-source code of Hibr2Bin, we found a difference in the number of compression methods used in the file. While paper [11] mentioned two compression methods used in modern hibernation files, Hibr2Bin source code defined four different compressions types while its definition was not clear enough for us to understand them. As far as we are not building a decompression code, which is out of the scope of this research, we would not be able to confirm that point using the manual analysis.

As a conclusion of the result of test case A, we could say that the file layout of the modern hibernation file defined in the reference paper [11] is still valid for the latest Windows 10 versions. Some minor changes happened in the known structures, fields names and locations. This answer to the research question three. Section 4.3.3.1 answers research question four, as the decompression tools outputs are different. Reasons seem to be related to Hibr2Bin decompression code.

We can summarize the main results of test case B, discussed in section 4.4, in three main points, as an answer to research question five.

Test case B shows that a hibernation file with the "WAKE" signature keeps the content of a various number of pages not only on the first page, as mentioned in [11]. The only modifications that happen to the kept pages are replacing the "HIBR" signature with the "WAKE" signature. The file content is zeroed starting from one of the "free memory pages" as we named previously. A "WAKE" hibernation file does not contain any restoration set data, and when analyzing it using tools, it would not produce any output. This file is almost useless.

Test case B also shows that a file extracted offline with the "HIBR" signature has the same file structure regardless this file is related to a Hibernation power state or Fast start-up power state. We could differentiate between the two files by analysis, as a file extracted from a hibernation power state would contain user-related activities and processes, while the fast start-up hibernation file would contain information related to the kernel session.

Finally, Test case B demonstrated that the size of the hibernation file with the "HIBR" signature is equal to the size of installed physical memory (RAM), regardless of the configured type of hibernation file. The only difference in storage size between full and reduced hibernation files is the size of the "WAKE" hibernation file.

The results of test case D shows that by enabling the "HORM" feature, hibernating the system would create a new hibernation file with "HORM" signature. As the hibernation file is preserved with each shutdown, it is not possible to track any sessions activities, unless a new hibernation occurs. This test case answer to research question six.

5 Analysis of Windows 10 hibernation file using tools

We understood the hibernation file structure and characteristics using the Manual analysis. The aim of this chapter is demonstrating the forensics value of the file. This section shares the analysis and results of test case D. All the files tested on each of the mentioned tools are the files discussed in section 3.3.4. The files contain the same list of evidence discussed in section 3.3.4.

5.1 Tools findings and insights

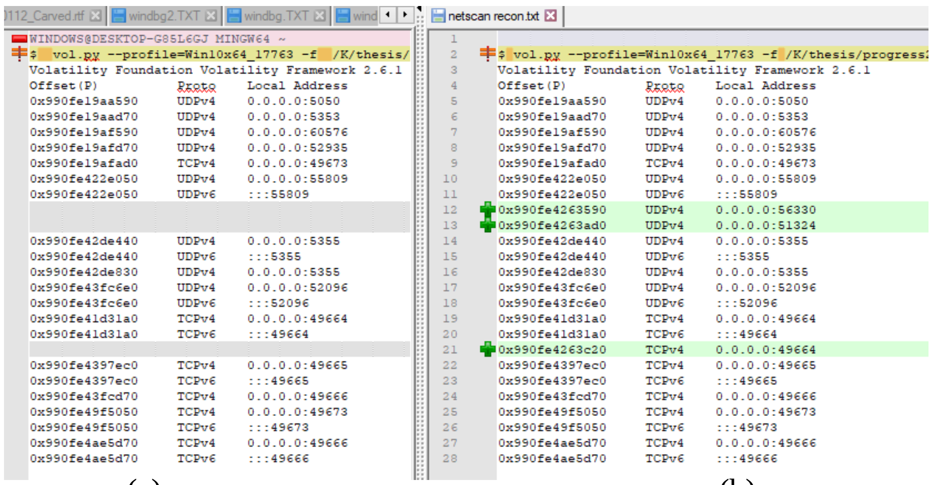
In this section, we would document our notes about each tool and its findings when analyzing the files of test case D. By the end of this section we should have an answer to the research question seven, and some insights about research question eight and nine. Please note that in this section when we mention, for example, the file D_File_2, we mean at least three files in real life (files extracted from v 1809, v1903, v1909). For some versions like v1903, we repeated test case D more than six times and each time some hibernation files where extracted and analyzed. This justifies why we mentioned that the single filename is analyzed at least three times.

The reader might notice that we provide more details about the features of the tools in sections 5.1.4, 5.1.5 and 5.1.6. These sections are related to the Group A test tool. We tried to share our experience with the reader, as we hope that this thesis gives a clear picture of each tool features, which might help the user in choosing a suitable tool for his requirement.

5.1.1 Volatility

Volatility does not support the analysis of modern hibernation files in its original format. It is still possible to analyze a hibernation file after decompressing it using Hibr2Bin or Hibernation Recon tools. Volatility is a CLI software that can extract different data from a memory image. As clarified in section 4.3.31, we analysed decompressed files using Hibr2Bin and Hibernation Recon using volatility, as the files extracted from the tools are not identical. We ran selected commands on both files and compared the output data. The tests showed differences in some commands outputs between the output of Hibr2Bin and Hibernation Recon. The results of the analysis done using Volatility for the outputs of both decompression tools results were in favor of Hibernation Recon. We excluded Hibr2Bin from the test plan, and we decided to continue the tests on Volatility using the decompressed file of Hibernation Recon. Figure 48 is an example of the output difference between Hibr2Bin and Hibernation Recon, showing a missing data in the output of netscan plugin, when analyzing test case D_File_2 extracted from Windows v1809.

Volatility has a considerable list of plugins [39]. We tested most of the plugins, except mac and Linux specific plugins. Details about each Volatility command functionality could be found at [84]. Volatility can identify the windows profile of a decompressed hibernation file. The commands “imageinfo” and “kdbgscan” proposed matching memory images profiles for Windows 10 v1809 and v1903. Windows 10 V1909 is not currently supported by Volatility 2.6.1, “imageinfo” and “kdbgscan” commands suggested the profile Win10x64_18362 for v1909, which is Windows 10 v1903 profile. We clarified in section 5.3.1 that WindDbg also detects v1909 as v1903, which justify why many tools can successfully analyze the hibernation file of Windows 10 v1909 using the v1903 profile.



```
WINDOWS@DESKTOP-G85L6GJ MINGW64 ~
$ vol.py --profile=Win10x64_17763 -f /K/thesis/
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Proto    Local Address
0x990fe19aa590     UDPv4    0.0.0.0:5050
0x990fe19aad70     UDPv4    0.0.0.0:5353
0x990fe19af590     UDPv4    0.0.0.0:60576
0x990fe19afd70     UDPv4    0.0.0.0:52935
0x990fe19afad0     TCPv4    0.0.0.0:49673
0x990fe422e050     UDPv4    0.0.0.0:55809
0x990fe422e050     UDPv6    :::55809


0x990fe42de440     UDPv4    0.0.0.0:5355
0x990fe42de440     UDPv6    :::5355
0x990fe42de830     UDPv4    0.0.0.0:5355
0x990fe43fc6e0     UDPv4    0.0.0.0:52096
0x990fe43fc6e0     UDPv6    :::52096
0x990fe41d31a0     TCPv4    0.0.0.0:49664
0x990fe41d31a0     TCPv6    :::49664

0x990fe4397ec0     TCPv4    0.0.0.0:49665
0x990fe4397ec0     TCPv6    :::49665
0x990fe43fcd70     TCPv4    0.0.0.0:49666
0x990fe49f5050     TCPv4    0.0.0.0:49673
0x990fe49f5050     TCPv6    :::49673
0x990fe4ae5d70     TCPv4    0.0.0.0:49666
0x990fe4ae5d70     TCPv6    :::49666
```

(a)

```

$ vol.py --profile=Win10x64_17763 -f /K/thesis/progress
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Proto    Local Address
0x990fe19aa590     UDPv4    0.0.0.0:5050
0x990fe19aad70     UDPv4    0.0.0.0:5353
0x990fe19af590     UDPv4    0.0.0.0:60576
0x990fe19afd70     UDPv4    0.0.0.0:52935
0x990fe19afad0     TCPv4    0.0.0.0:49673
0x990fe422e050     UDPv4    0.0.0.0:55809
0x990fe422e050     UDPv6    :::55809
0x990fe4263590     UDPv4    0.0.0.0:56330
0x990fe4263ad0     UDPv4    0.0.0.0:51324
0x990fe42de440     UDPv4    0.0.0.0:5355
0x990fe42de440     UDPv6    :::5355
0x990fe42de830     UDPv4    0.0.0.0:5355
0x990fe43fc6e0     UDPv4    0.0.0.0:52096
0x990fe43fc6e0     UDPv6    :::52096
0x990fe41d31a0     TCPv4    0.0.0.0:49664
0x990fe41d31a0     TCPv6    :::49664
0x990fe4263c20     TCPv4    0.0.0.0:49664
0x990fe4397ec0     TCPv4    0.0.0.0:49665
0x990fe4397ec0     TCPv6    :::49665
0x990fe43fcd70     TCPv4    0.0.0.0:49666
0x990fe49f5050     TCPv4    0.0.0.0:49673
0x990fe49f5050     TCPv6    :::49673
0x990fe4ae5d70     TCPv4    0.0.0.0:49666
0x990fe4ae5d70     TCPv6    :::49666
```

(b)

Figure 47. Volatility – Comparing the output of the netscan plugin.

Hibr2Bin (a) and Hibernation Recon (b)

Volatility can extract a list of processes using different plugins as “pslist”, “psscan”, “psxview” and “pstree”. It could also extract details about loaded “DLL” files using the command “dlllist”, and the command “dlldump” could extract DLL files successfully from a decompressed hibernation file. It is also possible to display the processes environment variables using “envars” plugin. The plugin “verinfo” can extract version information from processes and DLLs of a decompressed hibernation file. We have to mention that when comparing the list of running processes on the live system before hibernation, to the list of extracted processes by Volatility, it was not found to be the same. Some services might be running on the computer before hibernation but do not show in the processes analysis. For example, when analyzing the decompressed version of the file D_File2 extracted from v1903 we did not found any traces related to chrome processes or libre office while both processes were running before hibernation and where resumed once the system was resumed from hibernation.

Memory related information could be extracted from a decompressed hibernation file. The plugin “memmap”, for example, shows the physical and virtual address of memory pages. It is also possible to extract information related to Virtual Address Descriptors (VAD) [85] from a decompressed hibernation file using the commands “vaddump”, “vadinfo”, “vadtree” and “vadwalk”.

We were also able to extract data related to Kernel Memory and Objects using the plugins “modules” that list the kernel drivers loaded on the system. Plugins like “modscan”, “moddump”, “thrdscan” were helpful to extract data related to modules and threads.

The decompressed hibernation file contains details related to network connection that could be extracted using “netscan” command. This command shows a part of the connections that were open on the laptop before hibernation, while it did not show any details related to IP addresses, which is expected behavior, as explained in section 2.3.4.

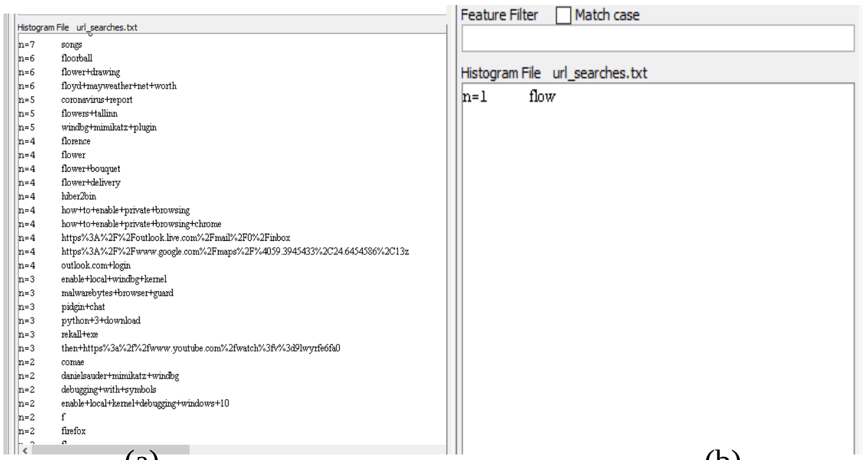
Volatility could not extract any information related to registry entries from a decompressed hibernation file. Plugins like “hivescan”, “hivelist”, “lsadump” and “userassist” do not provide an output.

Appendix 12 shows the list of plugins that were successfully able to extract data from the decompressed hibernation file extracted from Windows 10 v 1809, 1903, 1909.

5.1.2 Bulk_extractor

Bulk_extractor could extract data from a decompressed hibernation file. When processing Hibernation Recon output files using Bulk_extractor, the tool extracted a list of URLs that includes case D Evidence URLs. Bulk_extractor outputs include some information that was not found using any of the commercial tools. For example, A destination email address written in a draft message was extracted using Bulk_extractor, while none of the commercial tools detected that email address. Also, Bulk_extractor extracts different network connections in a file named "packet.pcap", this file contains the correct IP address that was used by the laptop before hibernation. We noticed that the mentioned MAC address in that ".pcap" file was not correct, while the correct one was found in "ether.txt" file. This IP information containing source and destination addresses and port numbers was not extracted by any of the commercial tools. Bulk_extractor extracts AES keys, and this information was also extracted by BlackLight as they use a bulk extraction tool in the background to extract it. The program also craved many SQLite databases.

We noticed that Bulk_extractor could also analyse a hibernation file in its original form, and it was able to extract valid data from it. However, Bulk_extractor analysis is more efficient when extracting data from a decompressed hibernation file. Figure 48, compare the content of url_searches file extracted from the analysis of a decompressed file to the output of a direct analysis of the hibernation file. The program was able to extract only part of a single search word "flow" (section 3.3.4, Table 8, E9). On the other hand, the decompressed file extracted a list of 58 search results, that contains relevant information to activities we did on the machine. It was surprising that Bulk_extractor was able to track all our browser activities, while some commercial tools were not.


Histogram File url_searches.txt
Feature Filter
Match case
Histogram File url_searches.txt

```
n=7     songs
n=6     floorball
n=6     flower+drawing
n=6     floyd+mayweather+net+worth
n=5     coronavirus+report
n=5     flowers+tallinn
n=5     windbg+mimikatz+plugin
n=4     florence
n=4     flower
n=4     flower+bouquet
n=4     flower+delivery
n=4     hiber2bin
n=4     how+to+enable+private+browsing
n=4     how+to+enable+private+browsing+chrome
n=4     https%3A%2F%2Foutlook.live.com%2Fmail%2F0%2Finbox
n=4     https%3A%2F%2Fwww.google.com%2Fmaps%2F%4059.3945433%2C24.6454586%2C13z
n=4     outlook.com+login
n=3     enable+local+windbg+kernel
n=3     malwarebytes+browser+guard
n=3     pidgin+chat
n=3     python+3+download
n=3     rekall+exe
n=3     then+https%3a%2f%2fwww.youtube.com%2fwatch%3fv%3d9lwyrfe6fa0
n=2     comae
n=2     danielsauder+mimikatz+windbg
n=2     debugging+with+symbols
n=2     enable+local+kernel+debugging+windows+10
n=2     f
n=2     firefox
```

```
n=1     flow
```

(a) (b)

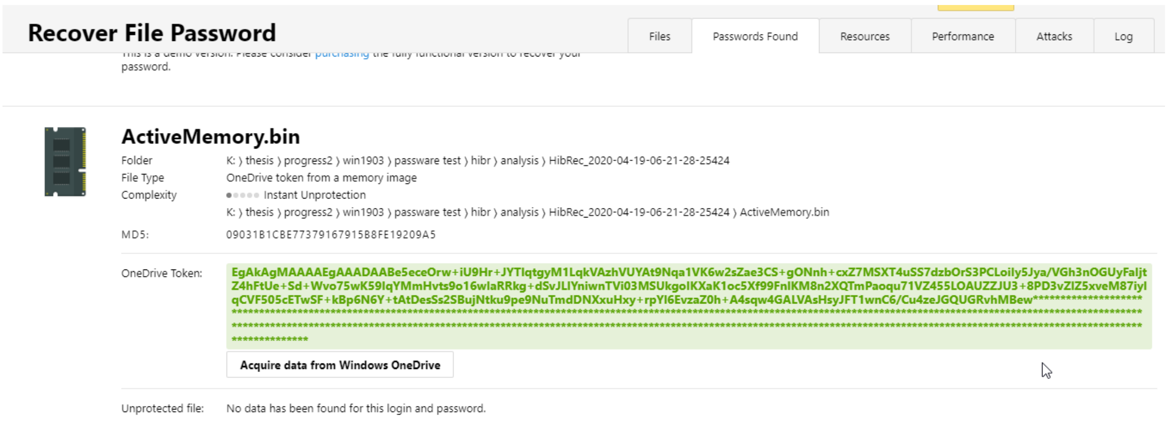
Figure 48. Bulk_extractor - url_searches - Hibernation Recon (a) , hiberfil.sys (b)

Unfortunately, due to time limitations sharing the results of the findings of free tools would stop at this point. We have a considerable amount of data and notes related to that topic that we hope published in the future. By this point, we got the answer to research question seven mentioned in section 1.1. Using free tools, we were able to extract different types of artifacts from a decompressed hibernation file.

5.1.3 Passware Kit Forensic

More than 25 files were analyzed using Passware. We analyzed different hibernation files extracted from test case D, as well as the decompressed versions of these hibernation files. Passware was not able to detect any passwords when processing the hiberfil.sys directly for all tested Windows 10 versions. Our results are based on the analysis of the decompressed files. The target of this test is confirming that it is possible to extract usernames and passwords from a hibernation file. Due to the demo version limitations, some passwords might not be extracted by this test, as the software would try each type of attack with limited functionality [65].

We analyzed decompressed hibernation files generated from hibernating and fast startup power states (D_File_2 and D_File_4). For all the three tested Windows versions and the tested power states, the “Windows user” memory analysis option was not able to extract any data. Passware was able to extract the first three letters of our target login named and password used account to access outlook webmail and Skype (account hibertest@outlook.com) using the “websites” option from the file D_File_2 only (Fast startup hibernation file did not contain that information). The software also detected other passwords, in both, hibernate and fast startup power states hibernation files. The extra passwords shown in the Passware findings seems to be false-positive, as we are not using any of the mentioned accounts or passwords. Passware was also able to extract OneDrive token using **OneDrive** memory analysis option from data extracted from D_File_2, while OneDrive data was not found in fast startup hibernation files (D_File_4) .



This is a demo version. Please consider purchasing the fully functional version to recover your password.

ActiveMemory.bin

Folder	K: 〉 thesis 〉 progress2 〉 win1903 〉 passware test 〉 hibr 〉 analysis 〉 HibRec_2020-04-19-06-21-28-25424
File Type	OneDrive token from a memory image
Complexity	●●●●● Instant Unprotection
	K: 〉 thesis 〉 progress2 〉 win1903 〉 passware test 〉 hibr 〉 analysis 〉 HibRec_2020-04-19-06-21-28-25424 〉 ActiveMemory.bin
MD5:	09031B1CBE77379167915B8FE19209A5
OneDrive Token:	EgAkAgMAAAAEgAAADAABe5eceOrw+iU9Hr+JYTlqtgyM1LqkVAzhVUYAt9Nqa1VK6w2sZae3CS+gONnh+cxZ7MSXT4uSS7dzbOrS3PCLoily5Jya/VGh3nOGUyFaljtZ4hFtUe+Sd+Wvo75wK59lqYMmHvts9o16wlaRRkg+dSvJLlYniwnTVi03MSUkgolKXaK1oc5Xf99FnlKM8n2XQTmPaoqu71VZ455LOAUZZJU3+8PD3vZlZ5xveM87iylqCVF505cETwSF+kBp6N6Y+tAtDesSs2SBujNtku9pe9NuTmdDNXxuHxy+rpYl6EvzaZ0h+A4sqw4GALVAsHsyJFT1wnC6/Cu4zeJGQUGRvhMBew**
Unprotected file:	No data has been found for this login and password.

Figure 49. Example of OneDrive Token recovered by Passware.

We also tested the disk decryption feature, to recover the BitLocker encryption key of a data drive with a password that starts with “123*****”. Passware quick guide [86] only provide instructions about decrypting Veracrypt container. We found a topic on Passware website with the title “How to decrypt BitLocker using Passware Kit” [87], while it does not provide clear instructions for the required task. The software required to provide an Encrypted BitLocker volume image file for such action, however we received error message “Disk contains no partitions” when uploading disk images extracted using FTK imager (we tested “.E01” and DD disk image with “.001” file extension as described in [88] and both were not accepted by the program). Finally, we were able to test that feature after creating a virtual hard drive using WinImage [89], as per unofficial instructions found on YouTube [90]. The software was not able to extract the drive password using the “.vhd” file plus hiberfil.sys extracted from the hibernation state (we tested the compressed and decompressed files with no success). To confirm the normal behavior, we analyzed the “.vhd” disk image with a memory image (“.mem” extension), and Passware was able to recover the first three letters of the decryption key. Table 15 summarize the results of Passware findings.

Table 14. Data extracted from hibernation files using Passware.

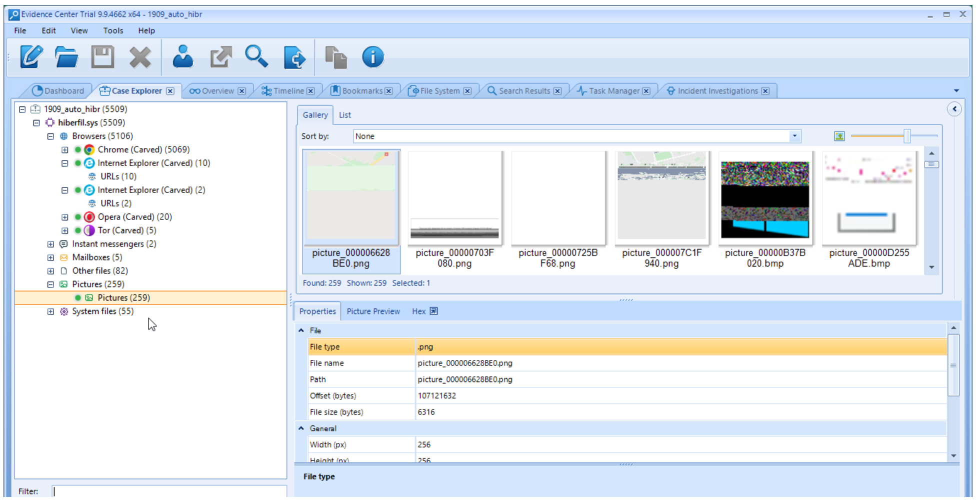
Hibernation file signature Windows power state	**Data extracted from decompressed hibernation files**			**Bitlocker Encryption Keys**
	Windows user	**websites**	**OneDrive**	
HIBR Hibernated	Not found	hibertest@outlook.com and other false positive accounts	Extracted OneDrive token	Not found
HIBR Fast startup	Not found	False-positive accounts	Not found	Not found

Passware can generate an HTML report with the results, and it also shows logs about types of attacks used on the file, which could be saved to a text file.

5.1.4 Belkasoft Evidence Centre

The analysis of 15 hibernation files extracted from different Windows 10 versions and states using Belkasoft Evidence Center were promising. We found that the tool can extract artifacts from a hibernation file up to Windows 10 version 1909, without decompression phase before analysis. Figure 50 shows an example of data extracted from v1909.

The tool can extract different types of low handing fruits artifacts as shown in Figure 48, low-level analysis is also supported by BEC, as the software has built-in Hex viewer, SQLite viewer, Registry viewer, and Pslist viewer. BEC has the capability of carve data files, and the user can add a file signature manually for carving files a particular type of file.



1909_auto_hibr (5509)
- hiberfil.sys (5509)
  - Browsers (5106)
    - Chrome (Carved) (5069)
    - Internet Explorer (Carved) (10)
      - URLs (10)
    - Internet Explorer (Carved) (2)
      - URLs (2)
    - Opera (Carved) (20)
    - Tor (Carved) (5)
  - Instant messengers (2)
  - Mailboxes (5)
  - Other files (82)
  - Pictures (259)
    - Pictures (259)
  - System files (55)

picture_000006628BE0.png, picture_00000703F080.png, picture_00000725BF68.png, picture_000007C1F940.png, picture_00000B37B020.bmp, picture_00000D255ADE.bmp

Found: 259 Shown: 259 Selected: 1

Properties | Picture Preview | Hex

File	
File type	.png
File name	picture_000006628BE0.png
Path	picture_000006628BE0.png
Offset (bytes)	107121632
File size (bytes)	6316
General	
Width (px)	256
Height (px)	256

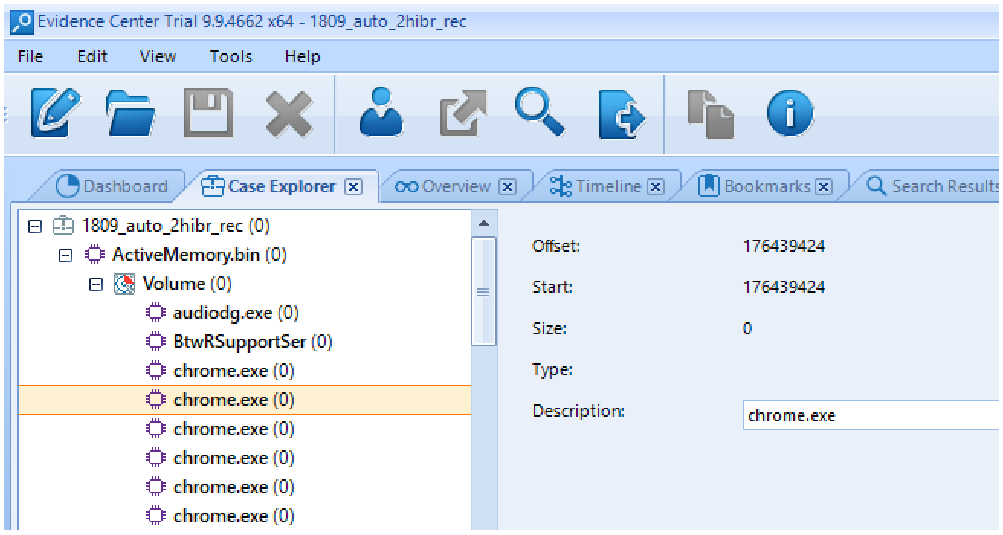
File type

Figure 50. BEC - example of data extracted from Windows 10 v1909.

During the analysis of test case D, carved ".db" files were found under "other files" category. The content of the database could be opened using the SQLite viewer, which is an advantageous option that helps to reduce investigation time.

When explorer BEC findings, we noticed that some false positives in the extracted evidence type. For example, our test laptop does not have Opera or TOR browser installed, while it was always reported in the results. Another example, for all the analyzed files, we did not found any entry for Firefox browser, while the browser is installed and was tested in our test case scenario.

The tool was not able to extract process list from different states of hibernation files of the three Windows versions. The file system tab was always empty, even after enabling the BelkaCarving option for all analyzed files, despite the fact that this feature is supported in RAM dump[69]. On the other hand, we analyzed a random sample of other types of files extracted from our test Windows versions, to understand the tool behavior toward other types of files. The sample included decompressed hibernation files extracted using Hir2bin and Hibernation Recon, page files, memory dump files with ".mem" extension and disk images with ".vhd" extension, we noticed that BEC extract process list from decompressed hibernation files for Windows 10 v1809(example shown in figure 49), while the analysis of decompressed hibernation files of Windows 10 v1903 and 1909 result contains data similar to the output of the analysis of hiberfil.sys (figure 51) and does not include any information about process list.



Evidence Center Trial 9.9.4662 x64 - 1809_auto_2hibr_rec

File Edit View Tools Help

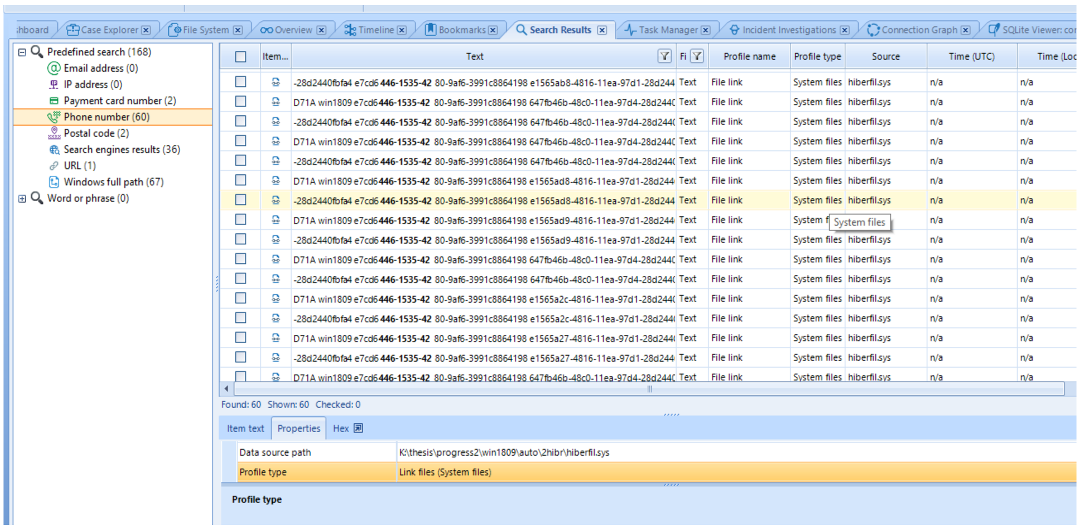
Dashboard | Case Explorer | Overview | Timeline | Bookmarks | Search Results

```
⊟ 1809_auto_2hibr_rec (0)
  ⊟ ActiveMemory.bin (0)
    ⊟ Volume (0)
        audiodg.exe (0)
        BtwRSupportSer (0)
        chrome.exe (0)
        chrome.exe (0)
        chrome.exe (0)
        chrome.exe (0)
        chrome.exe (0)
        chrome.exe (0)
```

Offset:	176439424
Start:	176439424
Size:	0
Type:	
Description:	chrome.exe

Figure 51. BEC - output of a Windows 10 v1809 decompressed hibernation file.

BEC has a list of predefined searches, that is used to automatically extract data related to phone numbers, Email address, IP address, and Payment cards, Postal codes, Search engine results and Windows full path. It also provides the possibility to configure customized searches. The Search engine results option was beneficial in our investigations. While other search options, like phone numbers and other kinds of numbers, seems to contain a high amount of false positive.



hboard | Case Explorer | File System | Overview | Timeline | Bookmarks | Search Results | Task Manager | Incident Investigations | Connection Graph | SQLite Viewer: co

```
⊟ Predefined search (168)
    Email address (0)
    IP address (0)
    Payment card number (2)
    Phone number (60)
    Postal code (2)
    Search engines results (36)
    URL (1)
    Windows full path (67)
⊞ Word or phrase (0)
```

☐	Item...	Text	Fi	Profile name	Profile type	Source	Time (UTC)	Time (Loc
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 e1565ab8-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 e1565ad8-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 e1565ad8-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 e1565ad9-4816-11ea-97d1-28d244	Text	File link	System f	System files	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 e1565ad9-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 e1565a2c-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 e1565a2c-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 e1565a27-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		-28d2440fbfa4 e7cd6 **446-1535-42** 80-9af6-3991c8864198 e1565a27-4816-11ea-97d1-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a
☐		D71A win1809 e7cd6**446-1535-42** 80-9af6-3991c8864198 647fb46b-48c0-11ea-97d4-28d244	Text	File link	System files	hiberfil.sys	n/a	n/a

Found: 60 Shown: 60 Checked: 0

Item text | Properties | Hex

Data source path	K:\thesis\progress2\win1809\auto\2hibr\hiberfil.sys
Profile type	Link files (System files)

Profile type

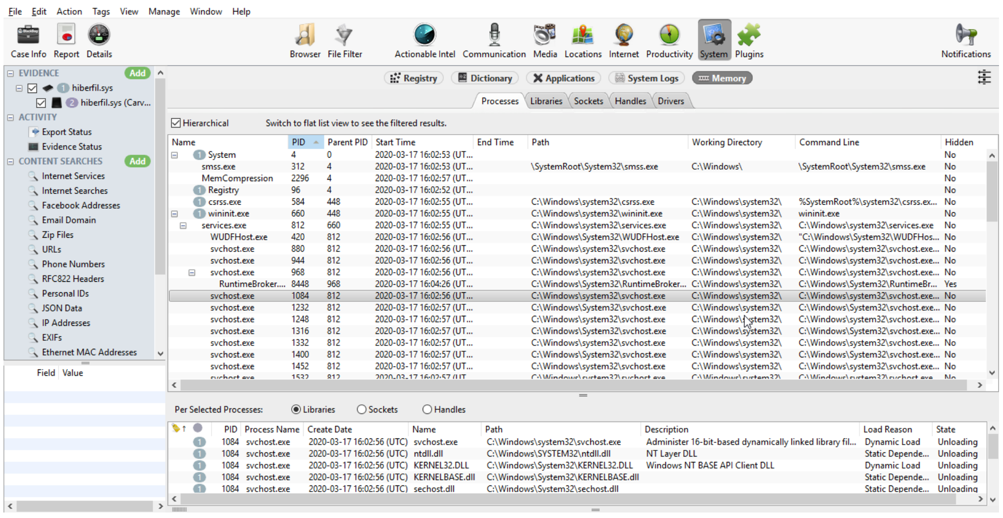
Figure 52. BEC - Example of false positives extracted by search results.

BEC has various reporting options. The user can extract a report using a right-click on selected data or using the Edit option in the toolbar. The report could be exported to different extensions.

5.1.5 BlackLight

From the three tested Windows 10 versions, BlackLight support only version 1809. While this eliminates a good part of the analyzed files, testing BlackLight software was challenging due to the hardware limitations of the analysis laptop.

BlackLight is the only tested software that was able to extract processes list and other information related to processes, when processing the hibernation file in its original format. We consider that the "memory" tab included in the "System" label is the most powerful memory analysis feature of BlackLight. These kinds of information are significant, as it could provide solid visibility about the activities that were performed on the computer before hibernation. The Processes tab, showed in Figure 52, could be listed in hierarchical or plain view. When selecting any process, BlackLight provides visibility about the processes list, start and end time, Path, and process ID as well as the parent process ID.



☑ Hierarchical   Switch to flat list view to see the filtered results.

Name	PID	Parent PID	Start Time	End Time	Path	Working Directory	Command Line	Hidden
System	4	0	2020-03-17 16:02:53 (UT...					No
smss.exe	312	4	2020-03-17 16:02:53 (UT...		\SystemRoot\System32\smss.exe	C:\Windows\	\SystemRoot\System32\smss.exe	No
MemCompression	2296	4	2020-03-17 16:02:57 (UT...					No
Registry	96	4	2020-03-17 16:02:52 (UT...					No
csrss.exe	584	448	2020-03-17 16:02:55 (UT...		C:\Windows\system32\csrss.exe	C:\Windows\system32\	%SystemRoot%\system32\csrss.ex...	No
wininit.exe	660	448	2020-03-17 16:02:55 (UT...		C:\Windows\system32\wininit.exe	C:\Windows\system32\	wininit.exe	No
services.exe	812	660	2020-03-17 16:02:55 (UT...		C:\Windows\system32\services.exe	C:\Windows\system32\	C:\Windows\system32\services.exe	No
WUDFHost.exe	420	812	2020-03-17 16:02:56 (UT...		C:\Windows\System32\WUDFHost.exe	C:\Windows\system32\	"C:\Windows\System32\WUDFHos...	No
svchost.exe	880	812	2020-03-17 16:02:56 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No
svchost.exe	944	812	2020-03-17 16:02:56 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No
svchost.exe	968	812	2020-03-17 16:02:56 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No
RuntimeBroker....	8448	968	2020-03-17 16:04:26 (UT...		C:\Windows\System32\RuntimeBroker...	C:\Windows\system32\	C:\Windows\System32\RuntimeBr...	Yes
svchost.exe	1084	812	2020-03-17 16:02:56 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No
svchost.exe	1232	812	2020-03-17 16:02:57 (UT...		C:\Windows\System32\svchost.exe	C:\Windows\system32\	C:\Windows\System32\svchost.exe...	No
svchost.exe	1248	812	2020-03-17 16:02:57 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No
svchost.exe	1316	812	2020-03-17 16:02:57 (UT...		C:\Windows\System32\svchost.exe	C:\Windows\system32\	C:\Windows\System32\svchost.exe...	No
svchost.exe	1332	812	2020-03-17 16:02:57 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No
svchost.exe	1400	812	2020-03-17 16:02:57 (UT...		C:\Windows\System32\svchost.exe	C:\Windows\system32\	C:\Windows\System32\svchost.exe...	No
svchost.exe	1452	812	2020-03-17 16:02:57 (UT...		C:\Windows\system32\svchost.exe	C:\Windows\system32\	C:\Windows\system32\svchost.exe...	No

Per Selected Processes: [✓ Libraries] Sockets Handles

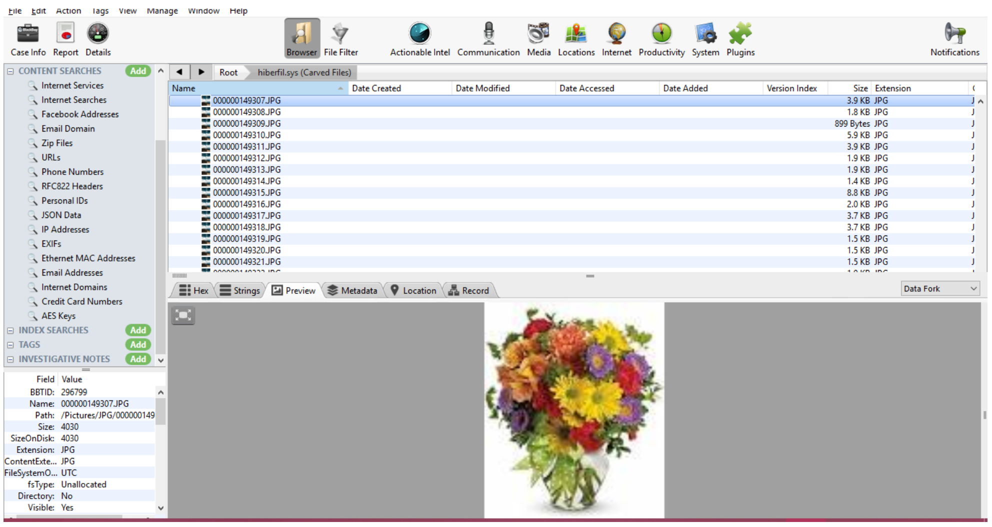
PID	Process Name	Create Date	Name	Path	Description	Load Reason	State
1084	svchost.exe	2020-03-17 16:02:56 (UTC)	svchost.exe	C:\Windows\system32\svchost.exe	Administer 16-bit-based dynamically linked library fil...	Dynamic Load	Unloading
1084	svchost.exe	2020-03-17 16:02:56 (UTC)	ntdll.dll	C:\Windows\SYSTEM32\ntdll.dll	NT Layer DLL	Static Depende...	Unloading
1084	svchost.exe	2020-03-17 16:02:56 (UTC)	KERNEL32.DLL	C:\Windows\System32\KERNEL32.DLL	Windows NT BASE API Client DLL	Dynamic Load	Unloading
1084	svchost.exe	2020-03-17 16:02:56 (UTC)	KERNELBASE.dll	C:\Windows\System32\KERNELBASE.dll		Static Depende...	Unloading
1084	svchost.exe	2020-03-17 16:02:56 (UTC)	sechost.dll	C:\Windows\System32\sechost.dll		Static Depende...	Unloading

Figure 53. BlackLight - List of extracted processes from v1809 hiberfil.sys.

The **Libraries** tab provides information related to used .dll files like process ID, name, library name, and creation date. The **Sockets** tab show network connection, the tab has data about process ID, name, creation date, and connection port, used protocol (TCP / UDP, and IPv4 /IPv6), as well as the state of each socket, these details similar to the details shown using the command "netstat". There are also **Drivers** tab include used system drivers and their paths. This information is also available when analyzing fast startup hibernation files (D_File_4).

We noticed that some tabs are not showing any details. For example, the Communication, Media, Locations, Internet, and productivity tabs were not showing details.

Carved files could be checked using the browser tab, while we could not export any of the craved files due to demo version limitations. We could preview the content of some files. It was also possible to check the strings available in the files, in case the content could not preview due to demo version limitations. The carved files types are Archive files (ex of extensions: .ZIP, .CAB, .TAR), Audio files (ex: .MIDI, .AMR), Documents files (ex: .APP, .EXE, .HTML, .SAM, .PST, .LNK, .SQLITE, .XML), File System files (MFT entries and NTFS _INDEX), Operating system files (.PF), Pictures and Videos. The test files analysis results included a large list of carved files that would be available for exporting in BlackLight full version. For example, when searching manually in the ".JPG" files for "flowers" pictures, we found a large number of flower pictures, either the full picture content as shown in the below figure or partial content, as shown in Figure 54.



Name	Date Created	Date Modified	Date Accessed	Date Added	Version Index	Size	Extension
000000149307.JPG						3.9 KB	JPG
000000149308.JPG						1.8 KB	JPG
000000149309.JPG						899 Bytes	JPG
000000149310.JPG						5.9 KB	JPG
000000149311.JPG						3.9 KB	JPG
000000149312.JPG						1.9 KB	JPG
000000149313.JPG						1.9 KB	JPG
000000149314.JPG						1.4 KB	JPG
000000149315.JPG						8.8 KB	JPG
000000149316.JPG						2.0 KB	JPG
000000149317.JPG						3.7 KB	JPG
000000149318.JPG						3.7 KB	JPG
000000149319.JPG						1.5 KB	JPG
000000149320.JPG						1.5 KB	JPG
000000149321.JPG						1.5 KB	JPG

Field	Value
BBTID:	296799
Name:	000000149307.JPG
Path:	/Pictures/JPG/000000149
Size:	4030
SizeOnDisk:	4030
Extension:	JPG
ContentExte...	JPG
FileSystemO...	UTC
fsType:	Unallocated
Directory:	No
Visible:	Yes

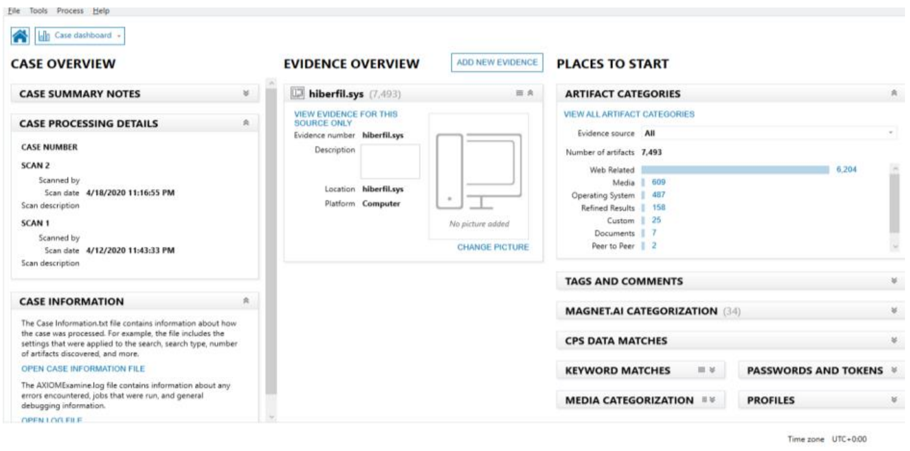
Figure 54. Example of carved files using BlackLight

Demo license limitations also prevented any visibility on the total number of artifacts extracted by BlackLight, as we cannot test any feature related to reporting, it also prevents preview some of the carved files, and show a notification about demo limitations instead.

BlackLight also has a predefined list of content search (bulk extractions), to perform a search in memory artifacts, user can check the defined criteria for each search type, and the tool also shows some statistics about each search hit count, and give the possibility to apply filters for the search results. Internet searches is an example of an available predefined search. This search results included many search keywords that we have used in test case D. The search also included a predefined search of URLs, email domains, phone and cards numbers, AES keys and other types of search. These different types of search helped0 us to the evidence list of test case D. The search features available in BlackLight were advantageous.

5.1.6 Magnet Axiom

Magnet Axiom Trial has no limitations on the tool usage, so we were able to test the full functionality of the software. Magnet Axiom "Examine" displays the analysis progress and results. The Dashboard page includes an overview of the total extracted number of artifacts with statistics about each category, as per Figure 55. The Dashboard also includes Magnet.AI statistics about categorization results for chats and pictures and other options. Compare to BEC (Dashboard), and BlackLight (Case info and Details tabs), we believe that the Magnet Axiom dashboard is very informative and gives the user many required links to quickly access the details about case information and processing logs, as well as results statistics and categorization.



CASE NUMBER
SCAN 2
Scanned by
Scan date 4/18/2020 11:16:55 PM
Scan description
SCAN 1
Scanned by
Scan date 4/12/2020 11:43:33 PM
Scan description

The Case Information.txt file contains information about how the case was processed. For example, the file includes the settings that were applied to the search, search type, number of artifacts discovered, and more.

The AXIOMExamine.log file contains information about any errors encountered, jobs that were run, and general debugging information.

Evidence number hiberfil.sys
Description
Location hiberfil.sys
Platform Computer

Evidence source All
Number of artifacts 7,493

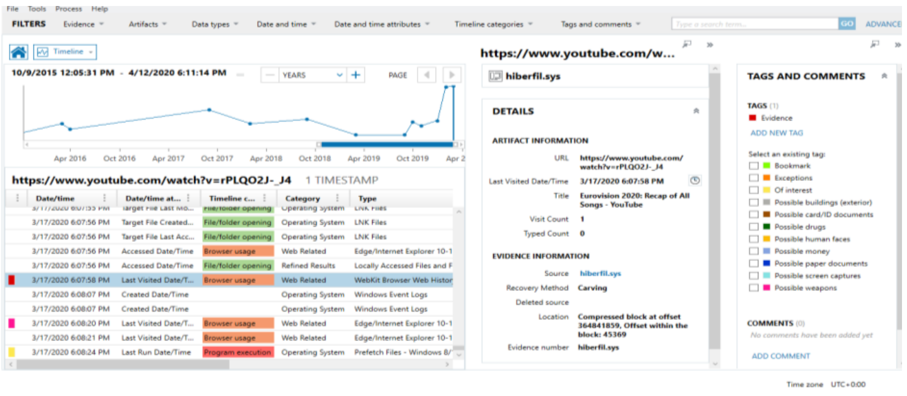
Web Related	6,204
Media	609
Operating System	487
Refined Results	158
Custom	25
Documents	7
Peer to Peer	2

Time zone UTC+0:00

Figure 55. Magnet Axiom Dashboard.

Magnet Axiom also provides a timeline explorer with many options, while BEC also provides the same option, Magnet Axiom timeline explorer give the user more powers. As shown in Figure 56, the user can view a graph for activities during a specific period.

User can zoom in the graph to get a more specific period. The user could also mark some activities with different colors and tags, and the program gives the possibility to filter the data using many criteria and categories, the timeline explorer also provides many details about user activities, with information about the data source and recovery method.



10/9/2015 12:05:31 PM - 4/12/2020 6:11:14 PM

https://www.youtube.com/watch?v=rPLQO2J-_J4   1 TIMESTAMP

	Date/time	Date/time at...	Timeline c...	Category	Type
	3/17/2020 6:07:55 PM	Target File Last Mo...	File/folder opening	Operating System	LNK Files
	3/17/2020 6:07:56 PM	Target File Created...	File/folder opening	Operating System	LNK Files
	3/17/2020 6:07:56 PM	Target File Last Acc...	File/folder opening	Operating System	LNK Files
	3/17/2020 6:07:56 PM	Accessed Date/Time	Browser usage	Web Related	Edge/Internet Explorer 10-1
	3/17/2020 6:07:56 PM	Accessed Date/Time	File/folder opening	Refined Results	Locally Accessed Files and F
🟥	3/17/2020 6:07:58 PM	Last Visited Date/T...	Browser usage	Web Related	WebKit Browser Web Histor
	3/17/2020 6:08:07 PM	Created Date/Time		Operating System	Windows Event Logs
	3/17/2020 6:08:07 PM	Created Date/Time		Operating System	Windows Event Logs
🟪	3/17/2020 6:08:20 PM	Last Visited Date/T...	Browser usage	Web Related	Edge/Internet Explorer 10-1
	3/17/2020 6:08:21 PM	Last Visited Date/T...	Browser usage	Web Related	Edge/Internet Explorer 10-1
🟨	3/17/2020 6:08:24 PM	Last Run Date/Time	Program execution	Operating System	Prefetch Files - Windows 8/

https://www.youtube.com/w...

hiberfil.sys

DETAILS

ARTIFACT INFORMATION

URL	https://www.youtube.com/watch?v=rPLQO2J-_J4
Last Visited Date/Time	3/17/2020 6:07:58 PM
Title	Eurovision 2020: Recap of All Songs - YouTube
Visit Count	1
Typed Count	0

EVIDENCE INFORMATION

Source	hiberfil.sys
Recovery Method	Carving
Deleted source	
Location	Compressed block at offset 364841859, Offset within the block: 45369
Evidence number	hiberfil.sys

TAGS AND COMMENTS

TAGS (1)
🟥 Evidence
ADD NEW TAG

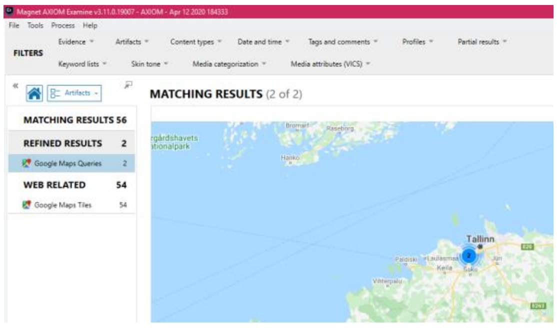
Select an existing tag:
- ☐ 🟩 Bookmark
- ☐ 🟧 Exceptions
- ☐ 🟨 Of interest
- ☐ ⬜ Possible buildings (exterior)
- ☐ 🟫 Possible card/ID documents
- ☐ 🟩 Possible drugs
- ☐ 🟧 Possible human faces
- ☐ 🟦 Possible money
- ☐ 🟦 Possible paper documents
- ☐ 🟦 Possible screen captures
- ☐ 🟪 Possible weapons

COMMENTS (0)
No comments have been added yet
ADD COMMENT

Time zone UTC+0:00

Figure 56. Magnet Axiom - Timeline explorer.

Like other tools, the software has its categorization for the extracted artifacts. As Magnet Axiom was the only commercial tool that was able to extract Windows event logs from the hibernation file, the tool was able to extract unique details that we did not find in other commercial tools findings. For example, device names and users account. The program also has some interesting features like different types of views, including a histogram view that can create a histogram for all the extracted artifacts types and numbers. The World map view is another type of available views. This view option might be similar to the location tab available in BlackLight. However, when checking the map search in BlackLight, the location tab was showing “Selected Item Does not contain Location Information”. On the other hand, Magnet Axiom was able to extract Google map queries as well as other data extracted from the web related details, then map the information as shown in Figure 57.

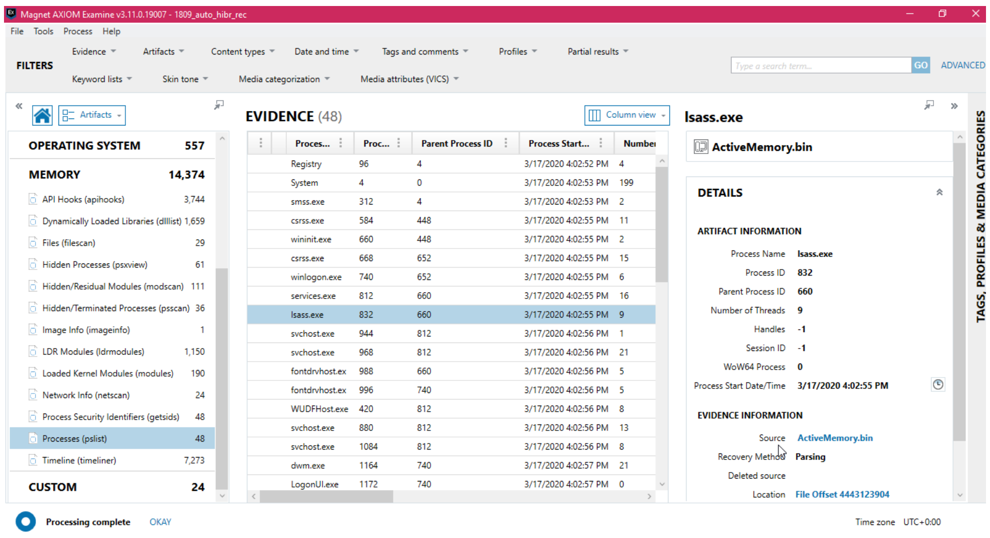


MATCHING RESULTS (2 of 2)

MATCHING RESULTS	56
REFINED RESULTS	2
Google Maps Queries	2
WEB RELATED	54
Google Maps Tiles	54

Figure 57. Magnet Axiom - World map view.

When processing a hibernation file with Magnet Axiom, the results do not include a “Memory” related artifact, like process lists and libraries. To overcome this limitation, we tested the processing of a decompressed hibernation file using Hibernation Recon, and the findings where promising. The result of decompressed file analysis includes results related to artifacts categories described earlier. In plus, a new category named “Memory” was added to the results and include the findings of Volatility plugins, As show in figure 58.



OPERATING SYSTEM	557
MEMORY	14,374
API Hooks (apihooks)	3,744
Dynamically Loaded Libraries (dlllist)	1,659
Files (filescan)	29
Hidden Processes (psxview)	61
Hidden/Residual Modules (modscan)	111
Hidden/Terminated Processes (psscan)	36
Image Info (imageinfo)	1
LDR Modules (ldrmodules)	1,150
Loaded Kernel Modules (modules)	190
Network Info (netscan)	24
Process Security Identifiers (getsids)	48
Processes (pslist)	48
Timeline (timeliner)	7,273
CUSTOM	24

EVIDENCE (48)

Proces...	Proc...	Parent Process ID	Process Start...	Number
Registry	96	4	3/17/2020 4:02:52 PM	4
System	4	0	3/17/2020 4:02:53 PM	199
smss.exe	312	4	3/17/2020 4:02:53 PM	2
csrss.exe	584	448	3/17/2020 4:02:55 PM	11
wininit.exe	660	448	3/17/2020 4:02:55 PM	2
csrss.exe	668	652	3/17/2020 4:02:55 PM	15
winlogon.exe	740	652	3/17/2020 4:02:55 PM	6
services.exe	812	660	3/17/2020 4:02:55 PM	16
lsass.exe	832	660	3/17/2020 4:02:55 PM	9
svchost.exe	944	812	3/17/2020 4:02:56 PM	1
svchost.exe	968	812	3/17/2020 4:02:56 PM	21
fontdrvhost.ex	988	660	3/17/2020 4:02:56 PM	5
fontdrvhost.ex	996	740	3/17/2020 4:02:56 PM	5
WUDFHost.exe	420	812	3/17/2020 4:02:56 PM	8
svchost.exe	880	812	3/17/2020 4:02:56 PM	13
svchost.exe	1084	812	3/17/2020 4:02:56 PM	8
dwm.exe	1164	740	3/17/2020 4:02:57 PM	21
LogonUI.exe	1172	740	3/17/2020 4:02:57 PM	0

lsass.exe

ActiveMemory.bin

DETAILS

ARTIFACT INFORMATION

Process Name	lsass.exe
Process ID	832
Parent Process ID	660
Number of Threads	9
Handles	-1
Session ID	-1
WoW64 Process	0
Process Start Date/Time	3/17/2020 4:02:55 PM

EVIDENCE INFORMATION

Source	ActiveMemory.bin
Recovery Method	Parsing
Deleted source	
Location	File Offset 4443123904

Processing complete

Time zone UTC+0:00

Figure 58. Magnet Axiom – Decompressed hiberfil.sys extracted from v1809.

As clarified earlier, the user must choose the image profile to proceed with hiberfil.sys analysis, the latest supported Windows 10 x64 image profile is Win10x64_18362 (v1903). We were able to extract artifacts from window 10 v1909 using the Win10x64_18362 profile.

The trial version of Magnet Axiom includes full reporting features. The reporting of Magnet Axiom has more options than BEC reporting. For example, the report could include a link for file contents. On the contrary to other commercial tools, after the expiration of the trial version, Magnet Axiom was still accessible for checking our previous test results, which helped us in the investigations.

5.1.7 Hive Recon

Due to time limitations, we were not able to analysis of outputs of Hive Recon data, and compare the findings or the tools to the full registry configuration. Hive Recon is able to extract Volatile hives as well as what they named stable hives. The analysis of the findings of this tool requires a good knowledge of registry entries. The program extracts each file name with two extensions “.CSV” and “hive”.

5.2 Comparing Group A tools

From section 5.1, we understood that each tool detects various types of artifacts. It is very complicated to map the finding between the different tools, due to the significant difference in scope and features. For this reason, we clarified in section 3.2.2 that the Group A tools results would be analyzed in more details, using comparative tables. Creating comparative tables was challenging, due limitations related to fields names and characteristics of each tool, that varies from a tool to another. We also faced other limitations related to the trial version features. For example, BlackLight limit all data related to reporting and data extraction, so it was challenging to have visibility on the total number of extracted Evidence and exact numbers of each category). We would present the results of the analysis of different hibernation file extracted using test case D in different tables.

5.2.1 Total number of artifacts

Table 16 reports the total numbers of evidence extracted by each tool. These numbers are a brute number that does not depends on the percentage of false positives and the efficiency of the findings, as each tool has its field naming and layout. The table could be categorized as the Quantitative result of the total number of artifacts extracted from test case D hibernation files.

Table 15. Group A - Quantitative results of test case D.

Version_Filename*	Total number of Evidence		
	BEC	BlackLight	Magnet Axiom
1809_D_File_2	7067	**	7493
1809_D_File_4	274	**	1666
1903_D_File_2	3633	Version not supported	6350
1903_D_File_4	208	Version not supported	1150
1909_D_File_2	5509	Version not supported	6423
1909_D_File_4	92	Version not supported	899

*Details were provided in section 3.3. **Reporting is limited in the BlackLight demo version.

Table 15, proves that the hibernation file extracted from the Windows 10 machine in the hibernation state contains more artifacts than the files extracted from a fast startup. This answers question 10.

5.2.2 Search for the Evidence list

Tables 16 and 17 are a sample of Qualitative test results that should assess the quality of the extracted artifcats. The target evidence list in this table was described in section 3.3.4.

Tables 16 and 17 demonstrate that each tool can extract different types of artifacts. Search for the list of evidence defined in section 3.3.4, using each tool, produced different results for the same processed file. These tables demonstrated that the list of supported artifacts types showed in Appendix 5, does not guarantee that the tool supports extracting the artifacts from a hibernation file. We conclude from these tables that there exist too many enhancements that could be done on the analysis tools. Also, we conclude to not take the output of any tools for granted. Always double-check. These two tabes answered to research question eight.

Table 16. A list of evidence that was only created on v1903.

Evidence	Application	Evidence	BEC	Magnet Axiom
			V1903	V1903
E16	Chrome	Gmail URL	√	√
		Draft email	X	X
E17	Chrome	Hangout Chat	X	X
E18	Edge	URL	√	√
E19	Edge - InPrivate	Bing Search	X	√
E20	Outlook	Draft email	X	X
E21	Notepad	Draft text file	X	X
E22	cmd	Commands	X	X
E23	BitTorrent	Remote connection IP	X	X

Table 17. Search for test case D evidence list.

Evidence number	Application	Artifact type	BEC			Black-Light	Magnet Axiom		
			V1809	V1903	V1909	V1809	V1809	V1903	V1909
E1	Firefox	Google search	X	X	X	√	√	√	√
E2	Firefox	URL	X	X	X	√	√	√	√
E3	Firefox	URL	X	X	√	√	√	√	√
		PDF	X	X	X	X	X	X	X
E4	chrome incognito	Google search	X	X	√	√	√	√	√
E5	chrome incognito	URL	√	X	√	√	√	√	√
E6	chrome incognito	URL	√	X	√	√	√	√	√
		picture	X	X	X	√*	√ *	√ *	X
E7	chrome incognito	Google maps URL	√	X	√	√	√	√	√
		Map image	√*	√*	√*	X	√*	√*	√*
		location	X	X	X	X	√	√	√
E8	chrome	video URL	√	√	√	√	√	√	√
		Parts of the video	X	X	X	**	X	X	X
E9	chrome	Google search	√	√	√		√	√	√
		Flowers pictures	√	X	X	√	√	X	√
E10	chrome	Download photo URL	√	√	√	√	√	√	√
		picture	X	X	X	√*	√*	X	X
E11	chrome	Webmail URL	√	√	√	√	√	√	√
		username	√	√	√	√	√	√	√
E12	chrome	Draft email	X	X	X	√***	X	X	X
E13	Skype	Skype messages	X	X	X	X	X	X	X
E14	Libre Office	Draft text file	X	X	X	X	X	X	X
E15	BitLocker	Encryption keys	X	X	X	X	X	X	X

* Partial parts of the pictures were found.

** Many video types extracted, while it is difficult to search for the parts of the targeted video due to demo limitations.

*** the content was extracted using a search in the carved documents and was found in an SQL database content as well as ".EFX" files.

5.2.3 Extracted artifacts types

Table 18 depends on our personal notes and findings of each tool behavior. For example, BlackLight is the only tool that was able to extract process list and MFT entries. Magnet Axiom is the only tool that was able to extract Windows event logs and data related to accounts and device names. BEC is the only tool that does not require any Windows profile selection before processing with hibernation file analysis.

Table 18. Group A - findings and features when processing hiberfil.sys.

	BEC[69],[68]			BlackLight			Magnet Axiom		
	V1809	V1903	V1909	V1809	V1903	V1909	V1809	V1903	V1909
Detect hibernation file image profile	√	√	√	X	X	X	X **	X	X
supported Windows version in hibernation file analysis*	√	√	√	√	X	X	√	√	X***
Process list	X	X	X	√	Versions not supported		X	X	X
Socket connections	X	X	X	√			X	X	X
Windows event logs	X	X	X	X			√	√	√
Identifiers related to devices and accounts	X	X	X	X			√	√	√
Extract Prefetch files	X			√			√		
Extract LNK files	√			√			√		
Extract MFT entries and NTFS _INDEX	X			√			X		
Predefined search	√			√			√		
Mapping data on the world map	X			X			√		
Identify the used Browser	√			X			X		
Visited paths using Windows file explorer	√			X			√		

* The latest supported version mentioned was found by testing as such information is not mentioned in the software guide, except for Magnet Axiom.

** This feature is only supported for memory images, while Hibernation file analysis requires manual identification of image profile.

*** The version is not officially supported while when processing using Windows 10 v1903 profile, we can extract data from the hibernation file.

Table 18 does not include all the types of data extracted by each tool. The table could provide a primary answer to research question nine (mentioned in section 1.1), as it shows different types of artifacts that could be extracted by each of the three commercial tools. Actually question nine is a summary of Chapter 5 and not limited to table 18. More details about types of artifacts to extract from a hibernation file could be found in section 6.4.

5.3 Summary of analysis using tool and results

Test case D demonstrated that there exist many free and commercial tools extract data from a hibernation file. Free tools and some of the commercial tool do not support the direct of a hibernation file. As a workaround, the user can decompress the file using a decompression tool. Many commercial tools can analyse a hibernation file in its original form. Each of the tested tools has its characteristics. The expected extracted artifacts list differ from a tool to another.

Chapter 5 answered to four research questions. Questions seven was answered in sections 5.1.1 and 5.1.2. Question eight was answered in section 5.2.2. Answer to question 9 was done in section 5.2.3, and the answer to question 10 found at section 5.2.1

6 Discussion

In this section, we would discuss the findings of the thesis research and summarize the results.

6.1 Manual analysis results

The results of the manual analysis show that the hibernation file layout described [11] is still applicable for the latest versions of Windows 10 hibernation files. Using manual analysis, we documented the known file structures content for our target Windows 10 versions, and we created a full datagram for the first page of the hibernation file. Such detailed documentation is not currently available in any of the hibernation file documentations that we have read.

During the analysis of test case A, we found undocumented pages in the file layout, and we were able to identify the usage of a part of these undocumented pages, which are used to store the addresses of free memory pages. While these undocumented pages do not seem to impact the hibernation file decompression and analysis, we believe that it should be mentioned in the hibernation file layout, so we created an updated layout for the hibernation file.

Our research confirmed that in order to conserve the modern hibernation file content, forensics practitioners should extract the hibernation file offline, as once the system is resumed, the file loses its value. Results of test case B show that contradictory to what was mentioned in [11], some Windows versions might include more than hibernation file header in the “WAKE” file. On the other hand, we found that regardless of the amount of data available in a resumed file, the file content would not contain any memory pages content. Such information confirms [11] a conclusion about collecting a hibernation file from a running machine and justifies the exclusion of a “WAKE” hibernation file from forensics investigations. Test case B results also showed that although a fast startup hibernation file does not include the user’s session, there is a probability to find some traces related to user activities in that hibernation file. During test case B, we created a Reduced hibernation file type and tracked its characteristics as this is not currently documented. We found that the only difference between a “Reduced” hibernation file and “Full” hibernation file is the size of the resumed hibernation file (file with “WAKE” signature). Regardless the type of hibernation file, a hibernated file size would equal to the size of available system memory (RAM).

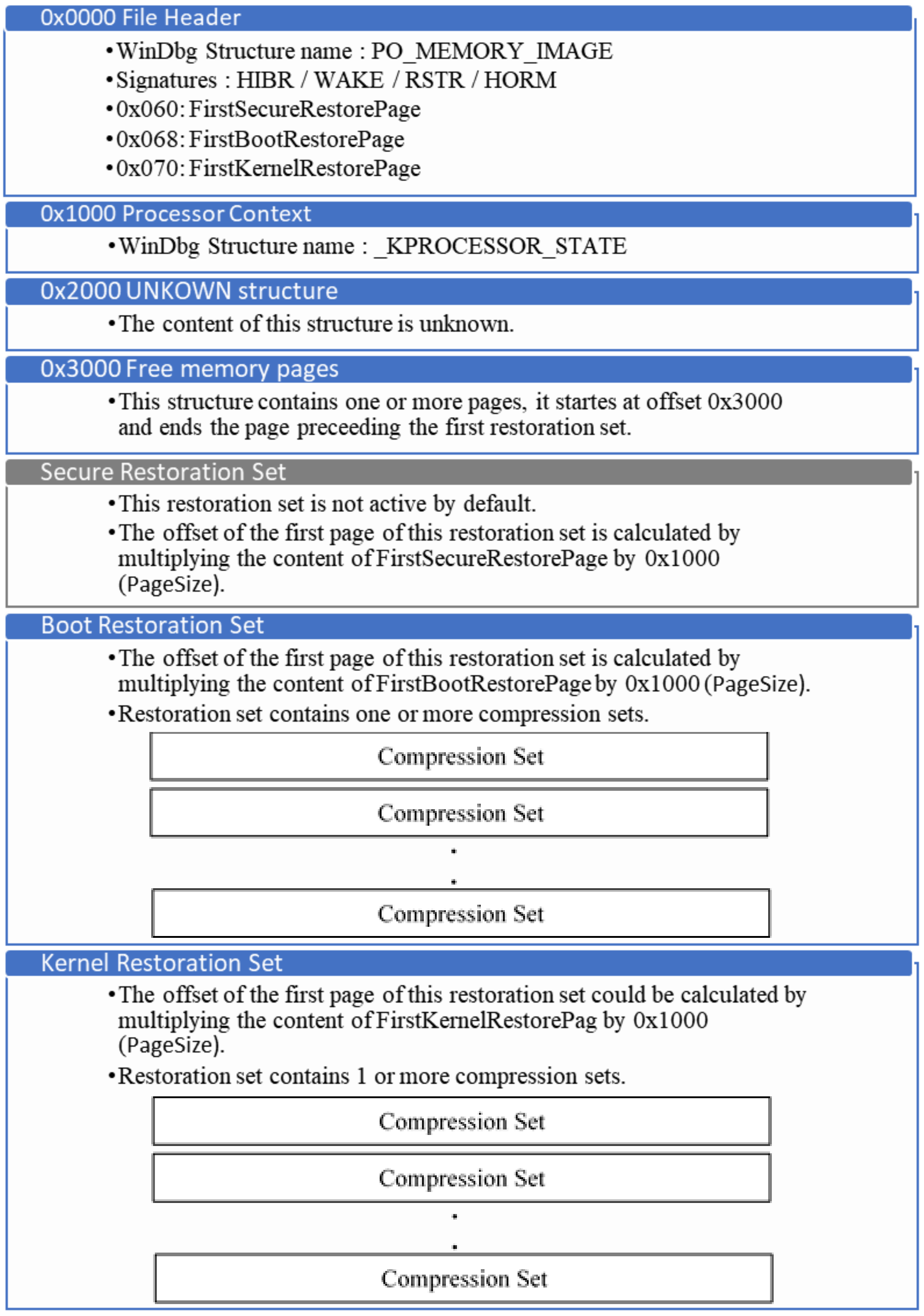
Test case C target was tracking the characteristics of a hibernation file when the “HORM” feature is enabled on the system. This case was out of the research scope of [11]. The test case results show that a “HORM” hibernation file content would remain the same even when the machine is resumed. Despite the technical limitations that we faced in the Test case C scenario, the preservation of the hibernation file hash value after each shutdown shows that “HORM” features would not conserve any of the user activities or machine states modifications in “HORM” hibernation file unless a new hibernation occurs.

6.2 Tools analysis results

The results of test case D using tools analysis confirmed that modern hibernation files could still be considered as a valuable source of data. Using different tools, we showed variable types of data that could be extracted from a hibernation file, and these data include unique artifacts that could be only extracted using memory forensics techniques. Running processes list and connections, volatile registry keys, as well as private browsing (chrome incognito, as an example) artifacts that are not saved in the browsing history, are examples of artifacts available in the hibernation file. The results of test case D include a comparison of the brute number of artifacts extracted by BEC and Magnet Axiom. During test case D, we searched for a predefined list of evidence that was created on each Windows version, and we tracked on different commercial tools then made a comparison between what different types of artifacts that BEC, BlackLight, and Magnet Axiom software were able to extract. For our knowledge, we are the first to do such a study on hibernation file content.

6.3 Hibernation file layout

We updated the hibernation file layout proposed in [11] based on our research results. The updated file layout contains the “Unknown structure” page and the “free map pages” that we detected during the analysis of test case A.



0x0000 File Header
•WinDbg Structure name : PO_MEMORY_IMAGE
•Signatures : HIBR / WAKE / RSTR / HORM
•0x060: FirstSecureRestorePage
•0x068: FirstBootRestorePage
•0x070: FirstKernelRestorePage
0x1000 Processor Context
•WinDbg Structure name : _KPROCESSOR_STATE
0x2000 UNKOWN structure
•The content of this structure is unknown.
0x3000 Free memory pages
•This structure contains one or more pages, it startes at offset 0x3000 and ends the page preceeding the first restoration set.
Secure Restoration Set
•This restoration set is not active by default.
•The offset of the first page of this restoration set is calculated by multiplying the content of FirstSecureRestorePage by 0x1000 (PageSize).
Boot Restoration Set
•The offset of the first page of this restoration set is calculated by multiplying the content of FirstBootRestorePage by 0x1000 (PageSize).
•Restoration set contains one or more compression sets.
Compression Set
Compression Set
Compression Set
Kernel Restoration Set
•The offset of the first page of this restoration set could be calculated by multiplying the content of FirstKernelRestorePag by 0x1000 (PageSize).
•Restoration set contains 1 or more compression sets.
Compression Set
Compression Set
Compression Set

Figure 59. Updated Windows 10 hibernation file layout.

6.4 Example of evidence types available in hibernation file

The tests performed during that research show that it is possible to extract many types of data from a hibernation file. We created a list of examples of evidence types available in a hibernation file, based on the results of our tests. We proposed tools to extract each type of artifacts, based on our experience with the tools.

The recommended ways to extract each type of artifacts does not mean that this is the only way to extract that artifact. For example, we excluded Hibr2Bin from our recommendations, as our tests show that Hibernation Recon was more accurate in hibernation file decompression. Some types of data might be extracted by commercial tools and processed transparently by the tool. Magnet Axiom, for example, does not show SQLite databases type in the output section, while from the user guide we understand that they do extract this type of database and process it, then include data extracted from SQLite databases in the results. For this reason, we excluded such types of artifacts to avoid confusion. Also, there might be a workaround to extract some data types that are not mentioned in the below table. For example, Bulk_extractor can extract JPEG Exif headers. Such information helps carve images, while we did not test this method, so it was not added to our recommendations. Most of the tools can extract credit card numbers and telephone numbers, which was not included in our tests' scenarios, so such information was also excluded from recommendations, as it was not tested.

Our tests show that some data is available in the hibernation file but was not detected by any of the tested tools. We found the content of chats messages and draft documents using a HxD and FTK tools search in the decompressed hibernation file, while they were not automatically detected by any tool. Such a result makes a strings extraction of the decompressed hibernation file content, a valid proposal for investigations. Strings could be extracted using too many ways and tools, including Bulk_extractor. This result also highlights a potential requirement of tools capabilities improvement when processing a hibernation file.

Table 19. Example of artifacts types and proposed tools to extract it.

Artifact type	Tools recommendation based on the results of our tests	
	Commercial tools	Free tools
Web access usernames + passwords	• Hibernation Recon + Passware Kit Forensics	
OneDrive token	• Hibernation Recon + Passware Kit Forensics	
Browser activities	• BEC • Magnet Axiom • BlackLight	• Hibernation Recon + Bulk_extractor
Processes, modules, and DLLs related information	• BlackLight • Hibernation Recon + Magnet Axiom	• Hibernation Recon + Volatility
Opened connections including source and destination IP		• Hibernation Recon + Bulk_extractor (packets.pcap)
Users Identifier	• Magnet Axiom	• Hibernation Recon + Volatility
Devices Identifier	• Magnet Axiom	
Event logs	• Magnet Axiom	
Pictures	• BEC • Magnet Axiom • BlackLight	
Videos	• Magnet Axiom, BlackLight	
LNK files	• BEC • Magnet Axiom • BlackLight	
Prefetch files	• Magnet Axiom • BlackLight	
NTFS _INDEX	• BlackLight • Hibernation Recon (Professional)	
Registry Hives	• Arsenal Hive Recon	
Visited paths using Windows file explorer	• BEC • Magnet Axiom	
Draft email content	• BlackLight (craved Document: SQLITE and EFX)	• Hibernation Recon + search using HxD

6.5 Hibernation file life cycle in digital forensics

Based on the research results, we could summarize the lifecycle of a hibernation file in digital forensics by 4 phases. This lifecycle does not include a “HORM” enabled feature’s hibernation file.

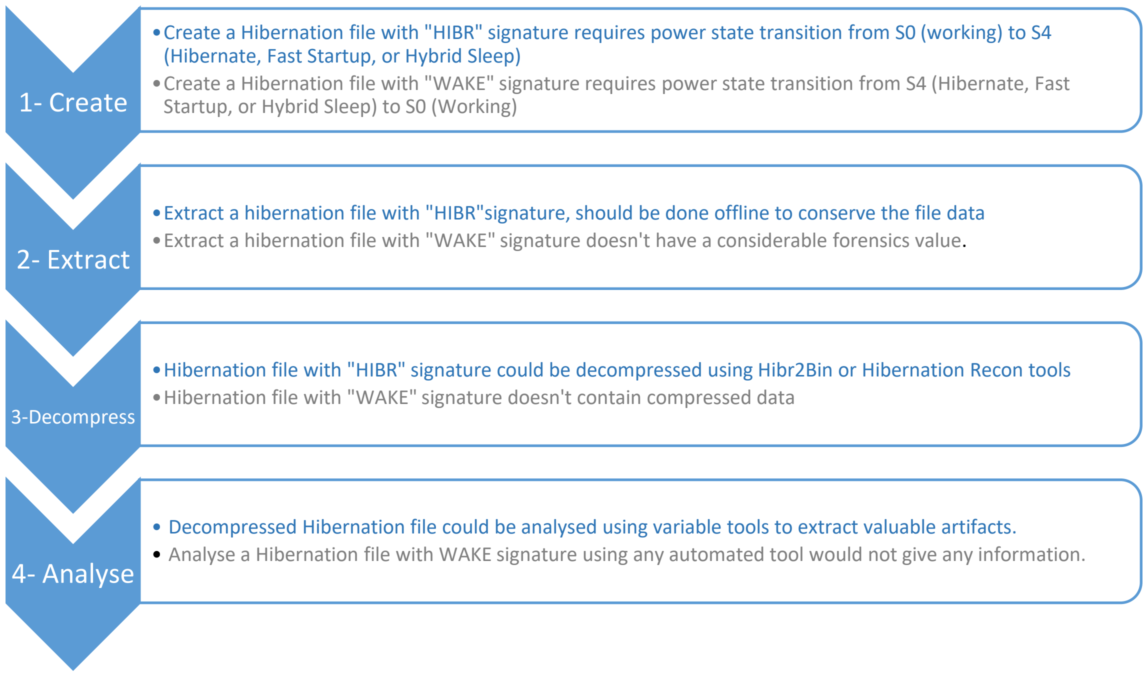

1- Create
•Create a Hibernation file with "HIBR" signature requires power state transition from S0 (working) to S4 (Hibernate, Fast Startup, or Hybrid Sleep)
•Create a Hibernation file with "WAKE" signature requires power state transition from S4 (Hibernate, Fast Startup, or Hybrid Sleep) to S0 (Working)
2- Extract
•Extract a hibernation file with "HIBR"signature, should be done offline to conserve the file data
•Extract a hibernation file with "WAKE" signature doesn't have a considerable forensics value.
3-Decompress
•Hibernation file with "HIBR" signature could be decompressed using Hibr2Bin or Hibernation Recon tools
•Hibernation file with "WAKE" signature doesn't contain compressed data
4- Analyse
• Decompressed Hibernation file could be analysed using variable tools to extract valuable artifacts.
• Analyse a Hibernation file with WAKE signature using any automated tool would not give any information.

Figure 60. Hibernation file life cycle.

The first phase is the creation of a new signature of the hibernation file, and modification of the file content, which is triggered by a power state transition. The transition from a hibernation file with the “HIBR” signature to “WAKE” signature does not happen directly, and there is a transition signature “RSTR” that appears during the system restart, while this signature was out of the scope of this research.

We considered the extraction phase of the hibernation file as the second phase of its life cycle; at this phase, the file could be extracted online (from a working machine) or offline (from a machine in hibernation state). Due to files permission, extracting the hibernation file might require special consideration, like using FTK imager to extract the file.

Phase 3 and 4 might be performed by the same tool, which makes phase three transparent for the end-user in some cases. As the content of the hibernation file is compressed, a decompression phase is required before performing the analysis of the file content. The decompressed version of the file would then be processed by the tools to extract variable types of artifacts depending on each tool's capabilities.

6.6 Research Questions Results

In this section, we would discuss the research questions results and recap each question's answers based on the research findings.

The first research question was “Is hibernation file created by default by Windows 10?” This question was answered in section 4.1. The hibernation file is created by default in Windows 10 latest versions, as fast startup power option is the default shutdown response to a GUI shut down. Fast startup hibernation file contains only the kernel session data, which contains many types of artifacts that might help in case investigation. The GUI hibernation option is not enabled by default, however, hibernating a Windows 10 could be done using CLI commands, without modifying any of Windows 10 default options.

We created datagrams for the structures that could be defined using WinDbg in section 4.2, and this answers the questions “Could we document the known hibernation file structures in a clear mapping/datagram?”

The answer to the question “Is the Modern hibernation file layout [11] still applicable for the latest versions of Windows 10 hibernation file?” is yes, and was provided in sections 4.2., and 4.3. We proved that the Modern hibernation file layout is still applicable for the latest versions of Windows 10 hibernation file, while we spotted two undocumented structures that should not impact the decompression of the hibernation file.

The answer for the question “Do the free tools decompressing Windows 10 hibernation file have the same output file when processing the same input hibernation file? ” is no, the conversion tools are not extracting the same output decompressed file. Manual analysis for decompressed shows that Hibr2Bin was not able to decompress some compressions types that are used in hiberfil.sys. Details were discussed in section 4.3.3.

The answer to the question “What are the impacts of the modifications of power configurations, as well as power states on a hibernation file content?” was provided in details in section 4.4 and summarized in section 4.6. The file size and content varies depending on the system power state and the power configuration.

Answer to “What is the effect of enabling the “HORM” feature on the hibernation file characteristics?” provided in 4.5 and summarized in 4.6. By enabling the “HORM” feature, and hibernating the Windows. The machine would resume using the same hibernation file after each shutdown. No user activities are track un such a situation.

The hibernation file is compressed, so forensics examiners cannot search for keywords or file signatures unless the file is decompressed using a decompression tool. However, the file header contains some details about the file like date and time of hibernation, memory page size. We found that despite the lack of direct analysis feature for modern hibernation files using well-known memory forensics tools like Rekall and Volatility, an applicable workaround could be used. Decompressing the files using Hibr2Bin or Hibernation Recon, then analyze the raw output binary file, is a logical solution. Volatility treats a decompressed hibernation file as a memory image and would be able to extract the image profile. Bulk_extractor can extract different types of artifacts from a decompressed hibernation file. By this, we answer the question “Could we extract artifacts from Windows 10 hibernation files using free tools??” More details were provided in section 5.1.1 and 5.1.2.

The answer to the question “Are there any differences found between commercial tools outputs?” was discussed in details in section 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.6, 5.1.7 and 5.2.2.1. there are differences between the list of artifacts types that each tool could extract, for example, the tested BlackLight version was the only commercial tool that could extract data related to processes from an analyzed hibernation file in its original form.

Modern hibernation files include various types of artifacts. Considering the nature of the volatile memory, hibernation file cannot always guarantee the presence of all kinds of artifacts, as some of the data might be offloaded to the page files or it might be destroyed. For this reason, we recommend using the hibernation file as a part of the source evidence list. Analyzing the hibernation file with the support of the page file would provide more visibility about the computer state before hibernation or shut down. Hibernation file can provide data related to running process lists, used libraries, open Sockets, search queries, and browser activities including private browsing sessions like sessions opened using Chrome Incognito and Firefox private Windows and its related media (audio, video, and pictures). The file also includes web passwords that were not saved in the browser, draft documents, as well as cloud services information like OneDrive tokens. Encryption keys and a list of AES keys were also extracted during our tests. The tests were not able to extract BitLocker encryption keys while this might be related to Passware demo version limitations. Extracted Hibernation files included other types of artifacts that could be found in a disk image like MFT entries, NTFS _Index entries, Windows event logs, LNK files, documents, web browsers activities, accessed Windows paths, and SQLite databases. This answers the questions "What kind of artifacts could be collected from a Windows 10 hibernation file?" More details in sections 5.2.3 and 6.4.

The last question was "In which Windows 10 power state, a hibernation file would contain the maximum number of artifacts?" Results of test case D showed that the hibernation file extracted from a hibernated computer contains the maximum number of artifacts that could be extracted using a hibernation file - Section 5.2.1.

6.7 Strengths of the study

The number of studies related to the modern hibernation file are minimal; for our knowledge, there exists a single study in that field at the time of writing this thesis. The thesis continues the work done in [11] and proved that the same file structure proposed in that paper is still valid for the latest versions of hibernation files. Full documentation of the first two pages of the hibernation file was provided in this thesis for the first time. We created a datagram of the full content of the hibernation file header and clarified differences between the two conversion tools used to decompress the hibernation file. For our knowledge, this paper is the first paper to compare the performance of commercial tools in the analysis of the hibernation file. We clarified by example that different types of artifacts could be extracted from a hibernation file. This research includes many details and information about Windows 10 hibernation file that we were not able to find it at the beginning of the research. The research covered many topics related to the hibernation file, which could give the reader a clear vision about the file value in digital forensics investigations. By the time of writing this research, we have not seen any other research giving such a variety of information about the Windows 10 hibernation file.

6.8 Weaknesses and limitations of the study

This study was done with limited resources, the number of samples taken, and the fact that the study is based on demo and trial versions of many programs should be considered as limitations. The findings of this study highlight the value of the hibernation file, while not all the findings could be taken as a rule due to a limited number of samples.

We extracted more than 100 different hibernation files during our study, with almost 2 TB of data. Such a large amount of data, with broad scope and limited time of software trials, resulted in a limited amount of information in each subject. Some of our tests were not included in the research due to time and thesis length limitations, so we decided to include results that were mainly tested more than one time. Our target was to give a big picture of the characteristics and usage of the hibernation file, while future studies could adapt many points of improvement to our research.

Doing our tests on a Windows 10 Home edition is another limitation of that study, as we were not able to test many features like enabling the TPM, and secure kernel-mode that were primarily included in our test plan. Enabling the HORM feature on our analysis laptop was also limited, as we were not able to extract a hibernation file during the hibernation or shutdown state, due to technical limitations related to our hardware. Hardware limitations also blocked the usage of Windows Kernel Debug. We were only able to test local kernel debug mode, as other debug modes were not supported by the used laptops. Thus, we lost the possibility to track the hibernation file during the resuming process and see more details about hibernation files with “RSTR” signatures.

Hardware limitations also caused many errors and delays of hibernation files analysis using some of the commercial tools. Although the hardware of the laptop used for analysis meets the minimum requirement of all the tested software, the laptop faced a degraded performance as it was processing many simultaneous tasks at the same time. This resulted in a loss of time and efforts that have impacted taking notes about all the artifacts that we already analyzed using the commercial tool before they lockdown. For example, some trial versions did not allow us to create reports or export data, to overcome this, we were taking print-screens for each activity done in the tool, while we were not able to take print-screens for every output of the tool. After the trial versions expired, we were in need to go through the data itself, not the print screens, to check for some details. As this was not possible anymore, we emitted some information that was already tested and even documented at some stage of the thesis, just because we currently do not have print screens to confirm some details about results.

6.9 Future Work

There is a clear research gap in the studies related to the hibernation file. This study spotted the value of the file and guided to understand the file structure described by [11]. We hope that our contribution helps software developers to implement a Volatility plugin or any other free tools that could directly analyze the hibernation file. Further research is required to test generating the third restoration set relate to FirstSecureRestorePage header entry, as per [11], this restoration set might be related to Secure Kernel Mode (SKM).

More researches might be done to document current compression types used in modern hibernation files with the guide of Windows Xpress algorithm documentation [32]. Studies could also be extended to include a comparison between the type of data that could be found when combining the analysis of hibernation file and page files compared to the data found in the disk image and data found using a memory capture. We already did primary tests related to point, while further research might be useful to document the differences between data types extracted from variable media types and clarify the unique keys findings of each of them. The impact of Encryption tools like BitLocker, VeraCrypt on the hibernation file, seems to the interesting research area. We did some quick investigations related to that topic that was not included in that research, as we found that it requires in-depth analysis and some test scenarios. It would help fellow researchers interested in the encryption topic to check a quick investigation related to BitLocker encryption that was released by Arsenal Recon by the end of April 2020 [91].

Studying other operating systems hibernation files, like macOS and Linux, and check the availability of such features in IoT systems might be another interesting area of research.

7 Conclusion

During this research, we extracted more than 100 hibernation files. Files were extracted from different Windows 10 versions, with variable Windows configurations and power states, to analyze the impact of these modifications on the hibernation files. Four test cases were created during the research. Three of the test cases helped to analyze the file manually. The results of test case A documented the missing structure in the current known Windows 10 hibernation file layout. Test case B documented the impact of power state modification on the content of the hibernation file. Test case C documented the impact of enabling the HORM feature on the hibernation file content. On the other hand, tools analysis was used on the hibernation files of test case D, we created a predefined list of activities for this test case, and we applied the same scenario on the three Windows 10 version in the scope of our research. Each hibernation file of test case D was analyzed using different commercial and free tools; some of the files were analyzed more than one time on the same tool using variable configuration. We documented the findings of test case D, highlighted the differences between the different analysis tools, and gave recommendations that might help digital forensics practitioners.

The amount of processed data during that research was a real challenge, as well as documentation contradictions, misleading information, and technical resources limitations. Our study highlighted many of these documentation contradictions and insights. The study should help readers to understand Windows 10 hibernation file structure and set his expectations toward different tools behavior during hibernation file analysis.

We recommend forensic examiners to extract the hibernation file offline, in case no memory image was taken for the evidence device. The fast startup feature is enabled by default in the Windows 10, which generates a limited content version of the hibernation file with each GUI shutdown. Fast startup hibernation file does not contain the user session, while many types of data related to Kernel sessions are still available in a fast startup hibernation file.

We hope that this research helps forensic examiners to give more care to the hibernation file and understand to required precautions to be taken, to get the most out of a Windows 10 hibernation file.

References

[1] M. Pollitt, “A history of digital forensics,” *IFIP Adv. Inf. Commun. Technol.*, vol. 337 AICT, pp. 3–15, 2010.

[2] “Power management/Suspend and hibernate - ArchWiki.” [Online]. Available: https://wiki.archlinux.org/index.php/Power_management/Suspend_and_hibernate . [Accessed: 23-Jan-2020].

[3] “Use the Energy Saver settings on your Mac - Apple Support.” [Online]. Available: https://support.apple.com/en-us/HT202824. [Accessed: 23-Jan-2020].

[4] U. EFI Forum, “Advanced Configuration and Power Interface (ACPI) Specification, Version 6.2,” 2017.

[5] “How to disable and re-enable hibernation on a computer that is running Windows.” [Online]. Available: https://support.microsoft.com/en-us/help/920730/how-to-disable-and-re-enable-hibernation-on-a-computer-that-is-running. [Accessed: 13-Jan-2020].

[6] “Desktop Operating System Market Share Worldwide, last visit 8 | StatCounter Global Stats.” [Online]. Available: https://gs.statcounter.com/os-market-share/desktop/worldwide, last visit 8/12/2019. [Accessed: 13-Jan-2020].

[7] “Desktop Windows Version Market Share Worldwide, last visit 8 | StatCounter Global Stats.” [Online]. Available: https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide, last visit 8/12/2019. [Accessed: 13-Jan-2020].

[8] “Windows lifecycle fact sheet - Windows Help.” [Online]. Available: https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet. [Accessed: 13-Jan-2020].

[9] “Magnet IEF - Artifact-First Investigations | Magnet Forensics.” [Online].

Available: https://www.magnetforensics.com/products/magnet-ief/. [Accessed: 13-Jan-2020].

[10] A. L. Ayers, “Windows Hibernation and Memory Forensics,” no. April, 2015.

[11] J. T. Sylve, V. Marziale, and G. G. Richard, “Modern windows hibernation file analysis,” *Digit. Investig.*, vol. 20, pp. 16–22, 2017.

[12] M. Suiche, “Windows Hibernation File for Fun ‘N’ Profit,” in *Black Hat*, 2008.

[13] A. Case and G. G. Richard, “Memory forensics : The path forward,” *Digit. Investig.*, vol. 20, pp. 23–33, 2017.

[14] “UEFI FAQs | Unified Extensible Firmware Interface Forum.” [Online]. Available: https://uefi.org/faq. [Accessed: 13-Jan-2020].

[15] “System Power States - Win32 apps | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/power/system-power-states. [Accessed: 10-Jan-2020].

[16] P. Yosifovich, A. Ionescu, and D. A. Solomon, *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. Pearson Education.

[17] N. A. Hassan, *Digital forensics basics : a practical guide using windows OS*. .

[18] “Introduction to the page file - Windows Client Management | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/windows/client-management/introduction-page-file. [Accessed: 02-Feb-2020].

[19] “How to Make Windows Clear Your Page File at Shutdown (and When You Should).” [Online]. Available: https://www.howtogeek.com/282049/how-to-make-windows-clear-your-page-file-at-shutdown-and-when-you-should/. [Accessed: 12-Mar-2020].

[20] “Windows 8 / Windows Server 2012: The New Swap File | Ask the Performance Team Blog.” [Online]. Available: https://web.archive.org/web/20190228152513/https://blogs.technet.microsoft.co

m/askperf/2012/10/28/windows-8-windows-server-2012-the-new-swap-file/. [Accessed: 02-Feb-2020].

[21] D. R. Tobergte and S. Curtis, *The Art of Memory Forensics*, vol. 53, no. 9. 2013.

[22] "Chapter 4. Maintain Windows - Exam Ref MD-100: Windows 10, First Edition." [Online]. Available: https://learning.oreilly.com/library/view/exam-ref-md-100/9780135560624/ch04.xhtml. [Accessed: 30-Jan-2020].

[23] "Reducing the Disk Footprint for Windows 7 Hibernation." [Online]. Available: http://download.microsoft.com/download/7/E/7/7E7662CF-CBEA-470B-A97E-CE7CE0D98DC2/HiberFootprint.docx. [Accessed: 13-Jan-2020].

[24] "Wayback Machine." [Online]. Available: https://web.archive.org/web/20091024155939/http://www.msuiche.net/pres/PacSec07-slides-0.4.pdf. [Accessed: 15-Jan-2020].

[25] "Windows Hibernation File for Fun and Profit - Video." [Online]. Available: https://media.blackhat.com/bh-usa-08/video/bh-us-08-Suiche/black-hat-usa-08-suiche-windowshibernation-hires.m4v. [Accessed: 14-Jan-2020].

[26] "(No Title)." [Online]. Available: https://www.blackhat.com/presentations/bh-usa-08/Suiche/BH_US_08_Suiche_Windows_hibernation.pdf. [Accessed: 28-Jan-2020].

[27] M. Suiche, "Sandman Project," pp. 1–11, 2008.

[28] P. Kleissner, "Hibernation File Format," 2009.

[29] "Windows 10 SDK - Windows app development." [Online]. Available: https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk. [Accessed: 05-Feb-2020].

[30] "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1," 2016.

[31] "Vergilius Project | _PO_MEMORY_RANGE_ARRAY." [Online]. Available: https://www.vergiliusproject.com/kernels/x64/Windows Vista %7C

2008/SP1/_PO_MEMORY_RANGE_ARRAY. [Accessed: 16-Jan-2020].

[32] "[MS-XCA]: Xpress Compression Algorithm | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-xca/a8b7cb0a-92a6-4187-a23b-5e14273b96f8. [Accessed: 04-Feb-2020].

[33] "Hibernate Once/Resume Many (HORM) | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/customize/enterprise/hibernate-once-resume-many-horm-. [Accessed: 04-Feb-2020].

[34] "The Chilling Reality of Cold Boot Attacks - F-Secure Blog." [Online]. Available: https://blog.f-secure.com/cold-boot-attacks/. [Accessed: 23-Jan-2020].

[35] P. Kleissner, "Stoned bootkit," *Black Hat USA*, pp. 5–7, 2009.

[36] P. Kleissner, "Hibernation File Attack – Reino de España," 2010.

[37] "(1278) DeepSec 2009: Stoned déjà vu - again - YouTube." [Online]. Available: https://www.youtube.com/watch?v=R7F8xKGGHI4. [Accessed: 27-Jan-2020].

[38] A. Singh and P. Sharma, "Role of Hibernation File in Memory Forensics of windows 10," vol. 7, no. 12, pp. 42–47, 2016.

[39] "GitHub - volatilityfoundation/volatility: An advanced memory forensics framework." [Online]. Available: https://github.com/volatilityfoundation/volatility. [Accessed: 19-Jan-2020].

[40] "Setting Up Local Kernel Debugging of a Single Computer Manually - Windows drivers | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/setting-up-local-kernel-debugging-of-a-single-computer-manually. [Accessed: 04-Feb-2020].

[41] "Setting Up Local Kernel Debugging of a Single Computer Manually - Windows drivers | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/setting-up-local-kernel-debugging-of-a-single-computer-manually. [Accessed: 14-Jan-2020].

[42] “FTK® Imager | AccessData.” [Online]. Available: https://accessdata.com/products-services/forensic-toolkit-ftk/ftkimager. [Accessed: 27-Mar-2020].

[43] N. Ruff and M. Suiche, “Enter SandMan,” *Pasec*, pp. 1–22, 2007.

[44] “GitHub - comaeio/Hibr2Bin: Comae Hibernation File Decompressor.” [Online]. Available: https://github.com/comaeio/Hibr2Bin. [Accessed: 13-Jan-2020].

[45] “Your favorite Memory Toolkit is back… FOR FREE! - Comae Technologies.” [Online]. Available: https://blog.comae.io/your-favorite-memory-toolkit-is-back-f97072d33d5c. [Accessed: 20-Mar-2020].

[46] “Comae - The Future Of Cybersecurity - Comae - The Future Of Cybersecurity.” [Online]. Available: https://www.comae.com/. [Accessed: 20-Mar-2020].

[47] “Memory compression and forensics – My DFIR Blog.” [Online]. Available: https://dfir.ru/2018/09/08/memory-compression-and-forensics/. [Accessed: 19-May-2020].

[48] “FAQ | Arsenal Recon.” [Online]. Available: https://arsenalrecon.com/faq/#HibernationFAQ. [Accessed: 10-Mar-2020].

[49] “Shutdown: Clear virtual memory pagefile | Windows security encyclopedia.” [Online]. Available: https://www.windows-security.org/16b1e5b3f0ad7ef90556eb81fca92575/shutdown-clear-virtual-memory-pagefile. [Accessed: 12-Mar-2020].

[50] “About | volatilityfoundation.” [Online]. Available: https://www.volatilityfoundation.org/about. [Accessed: 17-Jan-2020].

[51] “The Volatility Foundation - Open Source Memory Forensics.” [Online]. Available: https://www.volatilityfoundation.org/. [Accessed: 18-Jan-2020].

[52] “Git on Windows - Getting Started — Codebase.” [Online]. Available: https://support.codebasehq.com/articles/getting-started/git-on-windows. [Accessed: 19-Jan-2020].

[53] "Volatility Labs: Announcing the Volatility 3 Public Beta!" [Online]. Available: https://volatility-labs.blogspot.com/2019/10/announcing-volatility-3-public-beta.html. [Accessed: 19-Jan-2020].

[54] "GitHub - google/rekall: Rekall Memory Forensic Framework." [Online]. Available: https://github.com/google/rekall. [Accessed: 12-Mar-2020].

[55] "Rekall Forensics." [Online]. Available: http://www.rekall-forensic.com/. [Accessed: 18-Jan-2020].

[56] "Plugin Reference — Rekall Forensics 1.7.2 documentation." [Online]. Available: https://rekall.readthedocs.io/en/latest/plugins.html#windows. [Accessed: 24-Mar-2020].

[57] "Development - Rekall Forensics." [Online]. Available: http://www.rekall-forensic.com/documentation-1/rekall-documentation/development. [Accessed: 24-Mar-2020].

[58] "Finding Evil in Windows 10 Compressed Memory, Part One: Volatility and Rekall Tools | FireEye Inc." [Online]. Available: https://www.fireeye.com/blog/threat-research/2019/07/finding-evil-in-windows-ten-compressed-memory-part-one.html. [Accessed: 24-Mar-2020].

[59] "GitHub - fireeye/win10_rekall: Rekall Memory Forensic Framework." [Online]. Available: https://github.com/fireeye/win10_rekall. [Accessed: 24-Mar-2020].

[60] "error when run rekall · Issue #519 · google/rekall · GitHub." [Online]. Available: https://github.com/google/rekall/issues/519. [Accessed: 24-Mar-2020].

[61] S. L. Garfinkel, "Digital media triage with bulk data analysis and bulk_extractor," *Comput. Secur.*, vol. 32, pp. 56–72, 2013.

[62] "simsong/bulk_extractor: This is the development tree. For downloads please see:" [Online]. Available: https://github.com/simsong/bulk_extractor. [Accessed: 12-May-2020].

[63] "Decompressing and Extracting Artifacts from Windows 8/Server 2012+

Hibernation Files | Ponder The Bits.” [Online]. Available: https://ponderthebits.com/2017/07/decompressing-and-extracting-artifacts-from-windows-8-server-2012-hibernation-files/. [Accessed: 13-May-2020].

[64] “Passware Kit Forensic - complete electronic evidence discovery.” [Online]. Available: https://www.passware.com/kit-forensic/. [Accessed: 10-Apr-2020].

[65] “What are the limitations for the demo version? Why did it fail to recover my password? – Passware.” [Online]. Available: https://support.passware.com/hc/en-us/articles/221742828-What-are-the-limitations-for-the-demo-version-Why-did-it-fail-to-recover-my-password-. [Accessed: 20-Apr-2020].

[66] “How to extract Windows login passwords from hibernation file or memory image instantly – Passware.” [Online]. Available: https://support.passware.com/hc/en-us/articles/221742428-How-to-extract-Windows-login-passwords-from-hibernation-file-or-memory-image-instantly. [Accessed: 28-Mar-2020].

[67] “How to extract website passwords from hibernation file or memory image – Passware.” [Online]. Available: https://support.passware.com/hc/en-us/articles/221742408-How-to-extract-website-passwords-from-hibernation-file-or-memory-image. [Accessed: 28-Mar-2020].

[68] “Belkasoft Evidence Center 2020.” [Online]. Available: https://belkasoft.com/ec. [Accessed: 18-Jan-2020].

[69] “Belkasoft Evidence Center User Reference.”

[70] “BlackLight® | BlackBag.” [Online]. Available: https://www.blackbagtech.com/products/blacklight/. [Accessed: 18-Jan-2020].

[71] “(No Title).” [Online]. Available: https://www.blackbagtech.com/software-downloads/releaseNotes/bl2019r3.pdf. [Accessed: 17-Apr-2020].

[72] “Magnet AXIOM - Digital Investigation Platform | Magnet Forensics.” [Online]. Available: https://www.magnetforensics.com/products/magnet-axiom/. [Accessed: 18-Jan-2020].

[73] "How it works... - Windows Forensics Cookbook." [Online]. Available: https://learning.oreilly.com/library/view/windows-forensics-cookbook/9781784390495/15db22ce-119f-4fa7-b49e-7e9281aeb9aa.xhtml. [Accessed: 18-Apr-2020].

[74] "Product Documentation." [Online]. Available: https://support.magnetforensics.com/s/product-documentation?prod=axiom. [Accessed: 18-Apr-2020].

[75] "Products | Arsenal Recon." [Online]. Available: https://arsenalrecon.com/products/. [Accessed: 18-Jan-2020].

[76] "HiveRecon and HbinRecon Launched | Arsenal Recon." [Online]. Available: https://arsenalrecon.com/2018/08/insights-post-hiverecon-and-hbinrecon-launched/. [Accessed: 19-May-2020].

[77] "New Versions of HiveRecon and HbinRecon Launched | Arsenal Recon." [Online]. Available: https://arsenalrecon.com/2018/10/new-versions-of-hiverecon-and-hbinrecon-launched/. [Accessed: 19-May-2020].

[78] "How to Check Windows 10 Computer System Specs & Requirements - Microsoft." [Online]. Available: https://www.microsoft.com/en-us/windows/windows-10-specifications. [Accessed: 04-Feb-2020].

[79] "Overview of Windows 10 IoT - Windows IoT | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows/iot-core/windows-iot. [Accessed: 04-Feb-2020].

[80] "Windows 10 - release information - Windows Release Information | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows/release-information/. [Accessed: 18-Jan-2020].

[81] "dt (Display Type) - Windows drivers | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/dt--display-type-. [Accessed: 04-Mar-2020].

[82] "Secure boot | Microsoft Docs." [Online]. Available:

https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot. [Accessed: 06-Apr-2020].

[83] I. Property *et al.*, “Xpress Compression Algorithm,” pp. 1–29, 2018.

[84] “Command Reference · volatilityfoundation/volatility Wiki.” [Online]. Available: https://github.com/volatilityfoundation/volatility/wiki/Command-Reference. [Accessed: 11-May-2020].

[85] B. Dolan-gavitt and B. Dolan-gavitt, “The VAD Tree : A Process-Eye View of Physical Memory By The VAD tree : A process-eye view of physical memory 5,” 2007.

[86] “Passware Kit Forensic 2018 v1 Quick Start Guide Passware Kit Forensic 2018 Quick Start Guide.”

[87] “How to decrypt BitLocker using Passware Kit – Passware.” [Online]. Available: https://support.passware.com/hc/en-us/articles/360024316834-How-to-decrypt-BitLocker-using-Passware-Kit. [Accessed: 19-Apr-2020].

[88] “How to decrypt Full Disk Encryption – Passware.” [Online]. Available: https://support.passware.com/hc/en-us/articles/115002145727-How-to-decrypt-Full-Disk-Encryption. [Accessed: 20-Apr-2020].

[89] “Download WinImage.” [Online]. Available: http://www.winimage.com/htm/download.htm. [Accessed: 20-Apr-2020].

[90] “(2) Bitlocker Brute Force Cracking (without Dump or Hibernate File) - YouTube.” [Online]. Available: https://www.youtube.com/watch?v=zvaJxnvbGic. [Accessed: 20-Apr-2020].

[91] “The Interesting Case of Windows Hibernation and BitLocker | Arsenal Recon.” [Online]. Available: https://arsenalrecon.com/2020/04/the-interesting-case-of-windows-hibernation-and-bitlocker/. [Accessed: 14-May-2020].

[92] “(No Title).” [Online]. Available: https://docs.oracle.com/cd/E19253-01/817-6223/chp-typeopexpr-2/index.html. [Accessed: 20-Jan-2020].

[93] “Releases · google/rekall · GitHub.” [Online]. Available: https://github.com/google/rekall/releases. [Accessed: 12-Mar-2020].

[94] “error received with a memory dump for win 10 v1809 · Issue #1 · fireeye/win10_rekall.” [Online]. Available: https://github.com/fireeye/win10_rekall/issues/1. [Accessed: 25-Mar-2020].

[95] “Recommended hardware.” [Online]. Available: https://belkasoft.com/hardware. [Accessed: 13-Apr-2020].

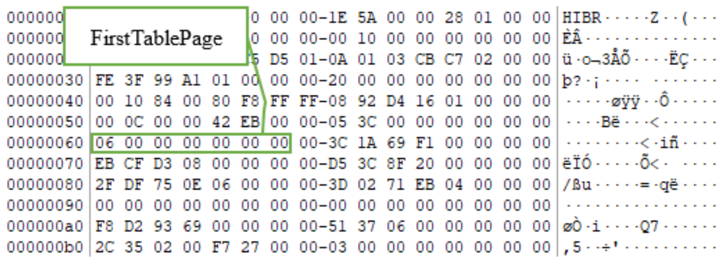
[96] I. BlackBag Technologies, “BlackLight User Guide Version 2019 R3,” 2019.

[97] “A COMPLETE DIGITAL INVESTIGATION PLATFORM, WITH THE PROCESSING POWER OF IEF.”

Appendix 1 – Partial manual analysis for a Windows 7 SP1 hibernation file

The Appendix 1 demonstrates a part of the manual analysis performed on Windows 7 SP1 x64. This demonstration might help the reader to understand the described structures and definitions in section 2.3.5.1. For our analysis, we assumed that in the case of x64 system, the "uintptr_t" length is 8 bytes as the description of "uintptr_t" is "Unsigned integer of size equal to a pointer" [92].

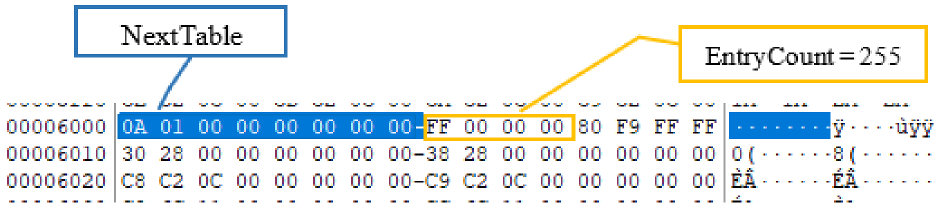
Figure 61 shows that the first table page address is calculated using the content of the "FirstTablePage" field. The field is available at offset (0x60) from the beginning of the hibernation file, as per the definition of "PO_MEMORY_IMAGE" for Windows 7 SP1 x64 (Figure 6). For this test hibernation file, the "FisrtTablePage" value is 0x6, and we multiplied this value by 0x1000 (the PageSize) to find that the location of the first "table page" is offset 0x6000.



```
000000                      0 00 00-1E 5A 00 00 28 01 00 00 HIBR····Z··(···
000000                      0 00 00-00 10 00 00 00 00 00 00 ÈÂ··············
000000                      5 D5 01-0A 01 03 CB C7 02 00 00 ü·○¬3ÅÕ····ËÇ···
00000030 FE 3F 99 A1 01 00 00 00-20 00 00 00 00 00 00 00 þ?·¡····  ·······
00000040 00 10 84 00 80 F8 FF FF-08 92 D4 16 01 00 00 00 ······øÿÿ··Ô·····
00000050 00 0C 00 00 42 EB 00 00-05 3C 00 00 00 00 00 00 ····Bë···<·······
00000060 06 00 00 00 00 00 00 00-3C 1A 69 F1 00 00 00 00 ········<·iñ····
00000070 EB CF D3 08 00 00 00 00-D5 3C 8F 20 00 00 00 00 ëÏÓ·····Õ<· ····
00000080 2F DF 75 0E 06 00 00 00-3D 02 71 EB 04 00 00 00 /ßu·····=·që····
00000090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
000000a0 F8 D2 93 69 00 00 00 00-51 37 06 00 00 00 00 00 øÒ·i····Q7······
000000b0 2C 35 02 00 F7 27 00 00-03 00 00 00 00 00 00 00 ,5··÷'··········
```

Figure 61. "FirstTablePage" entry - Windows 7 SP1 x64 hibernation file.

Figure 62 shows the content of the "table page" header - structure "_PO_MEMORY_RANGE_ARRAY". The "NextTable" value indicates that the next "table page" could be found at offset 0x010A000.



```
00006000 0A 01 00 00 00 00 00 00-FF 00 00 00 80 F9 FF FF ········ÿ····ùÿÿ
00006010 30 28 00 00 00 00 00 00-38 28 00 00 00 00 00 00 0(······8(······
00006020 C8 C2 0C 00 00 00 00 00-C9 C2 0C 00 00 00 00 00 ÈÂ······ÉÂ······
```

Figure 62. Windows 7 SP1 x64 - "_PO_MEMORY_RANGE_ARRAY" .

Appendix 2 – List of output files of Hibernation Recon

Table 24 is a comparison between the list of output files of Hibernation Recon.

Table 20. List of output filenames of Hibernation Recon as described in [48].

Output file	Description	Professional Mode	Free Mode
ActiveMemory.bin	Active memory decompressed & reconstructed.	✓	✓
DecompressedSlackModern.bin	All levels of slack (Modern format) decompressed & placed in one output file	✓	N/A
DecompressedSlackLevels/ DecompressedSlackLevelXXX Modern.bin	Slack (Modern format) decompressed & placed in multiple output files by slack level	✓	N/A
RawSlackModern.bin	Raw slack (Modern format) from all slack levels placed in one output file	✓	N/A
RawSlackChunks/RawSlackChunk_(Decimal Offset)_(Hex_Offset).bin	Raw slack placed in multiple output files by chunk	✓	✓
NonZeroAfterValidSlack.bin	Non-zero data after all valid levels of slack	✓	N/A
AllSlack.bin	All levels of slack (Modern & Legacy formats) decompressed, raw and non-zero in one output file	✓	N/A
Indx_I30_Entries.csv	Indexed folder content (a/k/a $I30 data) from active and slack space of NTFS INDX records	✓	N/A
Indx_ObjIdO_Entries.csv	Indexes of linked files (a/k/a $O data) from active and slack space of NTFS INDX records	✓	N/A
HibRec.log	Hibernation Recon log file	✓	✓

Appendix 3 – Results of Rekall tests

Tests done using Rekall were not successful for new Windows versions, to clarify the normal tool behavior, we first tested the tool with a Windows 7 hibernation file, and then we tested it with modern versions. Table 25 shows the result of tests done using Windows executable version 1.7.2.rc1 (Hurricane Ridge) downloaded from GitHub [93].

Table 21. Rekall - Tests results.

Windows version	Tested Files types	Analysis status	Comment / Error message
Win 7	**"hiberfil.sys"** – Original hibernated file	NOK	RuntimeError: Unable to find a valid profile for this image.
Win 7	**".bin"** files extracted using Hibr2Bin and using Hibernation Recon	OK	NT Build 7601.win7sp1_rtm.101119-1850
Win 10 v1809 and Win 10 v1903	**"hiberfil.sys", ".bin"** file converted by Hibr2Bin, **".bin"** file converted by Hibernation Recon, **".mem"** memory dump file captured using FTK imager	NOK	RuntimeError: Unable to find a valid profile for this image.

We also tested the FireEye updates applied to Rekall, and the results were unsuccessful as shown in Table 26 (latest supported version by FireEye updates) [59]. A support case was open for FireEye on GitHub.

Table 22. Rekall - Tests results of FireEye plugins

Windows version	File type	Analysis status	Comment / Error message
Win 7 and Win 10 (v1809 and v1903)	**"hiberfil.sys"** – Original hibernated file	NOK	RuntimeError: Unable to find a valid profile for this image.
Win 7	**".bin"** files extracted using Hibr2Bin and using Hibernation Recon	OK	NT Build 7601.win7sp1_rtm.101119-1850
Win 10 v1809 and v1903	- **".bin"** file converted by Hibr2Bin - **".bin"** file converted by Hibernation Recon - **". me**m**"** memory dump file captured using FTK imager	NOK	Part of the error received: *File "/root/win10_rekall/rekall-core/rekall/plugins/Windows/win10_memcompression.py", line 314, in __init__ super().__init__(**kwargs)* *TypeError: super() takes at least 1 argument (0 given)* **Full error could be found on GitHub** [94]**, as we opened an issue with FireEye.**

Appendix 4 – Minimum system requirement

Running memory image analysis might be high resources consuming. It is essential to check each tool's system requirement, as such criteria might cause system crashes and unexpected behaviors related to limitations of hardware resources. Our test laptop meets the minimum requirement of BlackLight software, while we faced repeated program crash and degradation of performance as the laptop was running other software at the same time. Table 7 provide the minimum system requirements for some of the used commercial tools.

Table 23. Minimum system requirements of some of the used commercial tools.

Required		Passware[1]	BEC* (Recommended)	BlackLight	Magnet Axiom
Operating System	Windows	• Windows Vista • Windows Server 2003/2008/2012 • Windows 7/8.x/10 (64-bit only)	• Minimum Windows XP • Recommended • Windows 7 • Windows 10	• Windows 10 or newer • Windows Server 2016 or newer	• Windows 10, Windows 8.1, Windows 8, or Windows 7 (64-bit only)
	Linux	Not Supported	Not Supported	Not Supported	Not Supported
	macOS	Not Supported	Not Supported	macOS Sierra (10.12.6)	Not Supported
Processor		1 GHz processor (2.4 GHz recommended)	4-core i7 processor with hyperthreading	2.7 GHz Intel Core i7 (Optimum Requirement: 3.1 GHz 6-Core Intel Xeon E5 or better)	4 logical cores (Recommended 8-16 logical cores)
RAM		512 MB of RAM (1 GB recommended)	16 Gb of RAM (per each instance of the product)	16 GB DDR3 (Optimum Requirement: 32 GB DDR3 or higher)	8 GB RAM (recommended 32 GB RAM)
Free hard disk space		150 MB (more if you use custom dictionaries)	SSD drive as a system disk and big magnetic drive for case data (1Tb or larger).	5GB of Disk Space (installation only) and 25 GB (Temp Space)	Enough space for storing images and cases

* There is no minimum hardware requirements for installing BEC[95], the mentioned information is for recommended hardware.

[1] https://www.passware.com/kit-forensic/buynow/

Appendix 5 – Supported types of artifacts

Each of the commercial tools has a list of supported types of extracted artifacts. We have chosen the below list to include the types of data targeted by our test case D. We excluded Passware from this comparison, as the software scope is extracting different types of data (passwords and encryption key). A tool might support a special type of data; while it is not able to extract that type of artifacts from hibernation files. Table 8 includes a list of artifacts types that each tool could extract, while this table does not mean that the tool can extract these artifacts from a hibernation file. We would explore what kind of artifacts were extracted from hibernation files in the results of test case D.

Table 24. Features comparison between some of the commercial tools.

Extracted artifacts	BEC[69],[68]	BlackLight[96]	Magnet Axiom[97]
Web-related artifacts	✓	✓	✓
Search Queries	✓	✓	✓
Device information*	✓	✓	✓
Windows event logs	✓	✓	✓
LNK files	✓	✓	✓
Process list	✓	✓	✓**
Timeline	✓	✓	✓
SQLite databases	✓	✓	✓
Users accounts	✓	✓	✓
Chats	✓	✓	✓
Detect encrypted files and volumes	✓	✓	✓
documents	✓	✓	✓
Pictures and videos	✓	✓	✓
Data Carving	✓	✓	✓

* Information like computer name, Mac address, Volume name.

**supported by Volatility Framework.

Appendix 6 – Hardware and OS Build specifications

Table 12 provides the list of hardware used during the research.

Table 25. List of hardware used during the research.

Item	Specifications
Test laptop	LENOVO ThinkPad T430 Processor: Intel Core i5-3320M (2.60GHz) Installed Physical Memory: 4.00 GB BIOS Mode: UEFI Function: The laptop used to generate the tests hibernation files
Test laptop Hard Drive	Model KINGSTON SA400S37120G Disk Size 111,79 GB Function: Three Windows 10 Home edition versions were installed on different partitions of the hard drive, as below. ▪ Microsoft Windows 10 Home, version 1809 (OS Build 17763.107) ▪ Microsoft Windows 10 Home, version 1903 (OS Build 18362.30) ▪ Microsoft Windows 10 Home, version 1909 (OS Build 18363.628)
External Bootable Hard Drive	Model: WD My Passport 25E1 USB Device Size: 1.82 TB OS: Microsoft Windows 10 Pro, version 1809 (OS Build 17763.253) Functions: The HD used for storage of the tests hibernation files, and as external bootable HD.
Analysis laptop	LENOVO ThinkPad E570 Processor: Intel(R) Core (TM) i7-7500U CPU @ 2.70GHz, 2901 MHz, 2 Core(s), 4 Logical Processor(s) Installed Physical Memory (RAM): 16.0 GB BIOS Mode: UEFI OS: ▪ Microsoft Windows 10 Education, version 1803 (OS Build 17134.1365) ▪ Upgraded to Windows 10 version 1909 (OS Build 18363.720) For testing the “HORM” feature, then downgraded back after the tests. Functions: The laptop used for analyzing the tests hibernation files. All the tools mentioned in section 2.6 were installed on the analysis laptop.
Virtual machine	OS: Windows version 1607 (OS Build 14393)

Appendix 7 – Hibernation file header

This Appendix is a documentation of the header content of Windows 10, using Windows Debugger Version 10.0.18362.1

Windows 10 version 1809

```
lkd> dt -r9 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
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
      +0x000 u                : <unnamed-tag>
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
      +0x000 QuadPart         : Int8B
   +0x028 InterruptTime    : Uint8B
   +0x030 FeatureFlags     : Uint8B
   +0x038 HiberFlags       : UChar
   +0x039 spare            : [3] UChar
   +0x03c NoHiberPtes      : Uint4B
   +0x040 HiberVa          : Uint8B
   +0x048 NoFreePages      : Uint4B
   +0x04c FreeMapCheck     : Uint4B
   +0x050 WakeCheck        : Uint4B
   +0x058 NumPagesForLoader : Uint8B
   +0x060 FirstSecureRestorePage : Uint8B
   +0x068 FirstBootRestorePage : Uint8B
   +0x070 FirstKernelRestorePage : Uint8B
   +0x078 FirstChecksumRestorePage : Uint8B
   +0x080 NoChecksumEntries : Uint8B
   +0x088 PerfInfo         : _PO_HIBER_PERF
      +0x000 HiberIoTicks     : Uint8B
      +0x008 HiberIoCpuTicks  : Uint8B
      +0x010 HiberInitTicks   : Uint8B
      +0x018 HiberHiberFileTicks : Uint8B
      +0x020 HiberCompressTicks : Uint8B
      +0x028 HiberSharedBufferTicks : Uint8B
      +0x030 HiberChecksumTicks : Uint8B
      +0x038 HiberChecksumIoTicks : Uint8B
      +0x040 TotalHibernateTime : _LARGE_INTEGER
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
         +0x000 u                : <unnamed-tag>
            +0x000 LowPart          : Uint4B
            +0x004 HighPart         : Int4B
         +0x000 QuadPart         : Int8B
      +0x048 HibernateCompleteTimestamp : _LARGE_INTEGER
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
         +0x000 u                : <unnamed-tag>
            +0x000 LowPart          : Uint4B
            +0x004 HighPart         : Int4B
         +0x000 QuadPart         : Int8B
      +0x050 POSTTime         : Uint4B
      +0x054 ResumeBootMgrTime : Uint4B
      +0x058 BootmgrUserInputTime : Uint4B
      +0x060 ResumeAppTicks   : Uint8B
      +0x068 ResumeAppStartTimestamp : Uint8B
      +0x070 ResumeLibraryInitTicks : Uint8B
      +0x078 ResumeInitTicks  : Uint8B
      +0x080 ResumeRestoreImageStartTimestamp : Uint8B
      +0x088 ResumeHiberFileTicks : Uint8B
      +0x090 ResumeIoTicks    : Uint8B
      +0x098 ResumeDecompressTicks : Uint8B
      +0x0a0 ResumeAllocateTicks : Uint8B
      +0x0a8 ResumeUserInOutTicks : Uint8B
      +0x0b0 ResumeMapTicks   : Uint8B
      +0x0b8 ResumeUnmapTicks : Uint8B
      +0x0c0 ResumeChecksumTicks : Uint8B
      +0x0c8 ResumeChecksumIoTicks : Uint8B
      +0x0d0 ResumeKernelSwitchTimestamp : Uint8B
      +0x0d8 CyclesPerMs      : Uint8B
      +0x0e0 WriteLogDataTimestamp : Uint8B
      +0x0e8 KernelReturnFromHandler : Uint8B
      +0x0f0 TimeStampCounterAtSwitchTime : Uint8B
      +0x0f8 HalTscOffset     : Uint8B
      +0x100 HvlTscOffset     : Uint8B
      +0x108 SleeperThreadEnd : Uint8B
      +0x110 PostCmosUpdateTimestamp : Uint8B
      +0x118 KernelReturnSystemPowerStateTimestamp : Uint8B
      +0x120 IoBoundedness    : Uint8B
      +0x128 KernelDecompressTicks : Uint8B
      +0x130 KernelIoTicks    : Uint8B
      +0x138 KernelCopyTicks  : Uint8B
      +0x140 ReadCheckCount   : Uint8B
      +0x148 KernelInitTicks  : Uint8B
      +0x150 KernelResumeHiberFileTicks : Uint8B
      +0x158 KernelIoCpuTicks : Uint8B
      +0x160 KernelSharedBufferTicks : Uint8B
      +0x168 KernelAnimationTicks : Uint8B
      +0x170 KernelChecksumTicks : Uint8B
      +0x178 KernelChecksumIoTicks : Uint8B
      +0x180 AnimationStart   : _LARGE_INTEGER
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
         +0x000 u                : <unnamed-tag>
            +0x000 LowPart          : Uint4B
            +0x004 HighPart         : Int4B
         +0x000 QuadPart         : Int8B
      +0x188 AnimationStop    : _LARGE_INTEGER
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
         +0x000 u                : <unnamed-tag>
            +0x000 LowPart          : Uint4B
            +0x004 HighPart         : Int4B
         +0x000 QuadPart         : Int8B
      +0x190 DeviceResumeTime : Uint4B
      +0x198 SecurePagesProcessed : Uint8B
      +0x1a0 BootPagesProcessed : Uint8B
      +0x1a8 KernelPagesProcessed : Uint8B
```

```
      +0x1b0 BootBytesWritten : Uint8B
      +0x1b8 KernelBytesWritten : Uint8B
      +0x1c0 BootPagesWritten : Uint8B
      +0x1c8 KernelPagesWritten : Uint8B
      +0x1d0 BytesWritten     : Uint8B
      +0x1d8 PagesWritten     : Uint4B
      +0x1dc FileRuns         : Uint4B
      +0x1e0 NoMultiStageResumeReason : Uint4B
      +0x1e4 MaxHuffRatio     : Uint4B
      +0x1e8 AdjustedTotalResumeTime : Uint8B
      +0x1f0 ResumeCompleteTimestamp : Uint8B
   +0x280 FirmwareRuntimeInformationPages : Uint4B
   +0x288 FirmwareRuntimeInformation : [1] Uint8B
   +0x290 SpareUlong       : Uint4B
   +0x294 NoBootLoaderLogPages : Uint4B
   +0x298 BootLoaderLogPages : [24] Uint8B
   +0x358 NotUsed          : Uint4B
   +0x35c ResumeContextCheck : Uint4B
   +0x360 ResumeContextPages : Uint4B
   +0x364 Hiberboot        : UChar
   +0x365 SecureLaunched   : UChar
   +0x366 SecureBoot       : UChar
   +0x368 HvCr3            : Uint8B
   +0x370 HvEntryPoint     : Uint8B
   +0x378 HvReservedTransitionAddress : Uint8B
   +0x380 HvReservedTransitionAddressSize : Uint8B
   +0x388 BootFlags        : Uint8B
   +0x390 RestoreProcessorStateRoutine : Uint8B
   +0x398 HighestPhysicalPage : Uint8B
   +0x3a0 BitlockerKeyPfns : [4] Uint8B
   +0x3c0 HardwareSignature : Uint4B
   +0x3c8 SMBiosTablePhysicalAddress : _LARGE_INTEGER
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
      +0x000 u                : <unnamed-tag>
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
      +0x000 QuadPart         : Int8B
   +0x3d0 SMBiosTableLength : Uint4B
   +0x3d4 SMBiosMajorVersion : UChar
   +0x3d5 SMBiosMinorVersion : UChar
   +0x3d6 HiberResumeXhciHandoffSkip : UChar
   +0x3d7 InitializeUSBCore : UChar
   +0x3d8 ValidUSBCoreId   : UChar
   +0x3d9 USBCoreId        : UChar
   +0x3da SkipMemoryMapValidation : UChar
```

Windows 10 versions 1903 and 1909

This output was extracted from a Windows 10 version 1903. We eliminated the output of Windows 10 version 1909 as both windows version provides the same output.

```
Microsoft (R) Windows Debugger Version 10.0.18362.1 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.
Connected to Windows 10 18362 x64 target at (Sun Feb 16 01:13:43.373 2020 (UTC + 2:00)), ptr64
TRUE
Symbol search path is: srv*
Executable search path is:
```

```
Windows 10 Kernel Version 18362 MP (4 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS Personal
Built by: 18362.1.amd64fre.19h1_release.190318-1202
Machine Name:
Kernel base = 0xfffff806`68400000 PsLoadedModuleList = 0xfffff806`68843290
Debug session time: Sun Feb 16 01:13:44.514 2020 (UTC + 2:00)
System Uptime: 5 days 11:12:12.527
lkd> dt -r9 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
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
      +0x000 u                : <anonymous-tag>
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
      +0x000 QuadPart         : Int8B
   +0x028 InterruptTime    : Uint8B
   +0x030 FeatureFlags     : Uint8B
   +0x038 HiberFlags       : UChar
   +0x039 spare            : [3] UChar
   +0x03c NoHiberPtes      : Uint4B
   +0x040 HiberVa          : Uint8B
   +0x048 NoFreePages      : Uint4B
   +0x04c FreeMapCheck     : Uint4B
   +0x050 WakeCheck        : Uint4B
   +0x058 NumPagesForLoader : Uint8B
   +0x060 FirstSecureRestorePage : Uint8B
   +0x068 FirstBootRestorePage : Uint8B
   +0x070 FirstKernelRestorePage : Uint8B
   +0x078 FirstChecksumRestorePage : Uint8B
   +0x080 NoChecksumEntries : Uint8B
   +0x088 PerfInfo         : _PO_HIBER_PERF
      +0x000 HiberIoTicks     : Uint8B
      +0x008 HiberIoCpuTicks  : Uint8B
      +0x010 HiberInitTicks   : Uint8B
      +0x018 HiberHiberFileTicks : Uint8B
      +0x020 HiberCompressTicks : Uint8B
      +0x028 HiberSharedBufferTicks : Uint8B
      +0x030 HiberChecksumTicks : Uint8B
      +0x038 HiberChecksumIoTicks : Uint8B
      +0x040 TotalHibernateTime : _LARGE_INTEGER
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
         +0x000 u                : <anonymous-tag>
            +0x000 LowPart          : Uint4B
            +0x004 HighPart         : Int4B
         +0x000 QuadPart         : Int8B
      +0x048 HibernateCompleteTimestamp : _LARGE_INTEGER
         +0x000 LowPart          : Uint4B
         +0x004 HighPart         : Int4B
         +0x000 u                : <anonymous-tag>
            +0x000 LowPart          : Uint4B
            +0x004 HighPart         : Int4B
         +0x000 QuadPart         : Int8B
```

```
+0x050 POSTTime         : Uint4B
+0x054 ResumeBootMgrTime : Uint4B
+0x058 BootmgrUserInputTime : Uint4B
+0x060 ResumeAppTicks   : Uint8B
+0x068 ResumeAppStartTimestamp : Uint8B
+0x070 ResumeLibraryInitTicks : Uint8B
+0x078 ResumeInitTicks  : Uint8B
+0x080 ResumeRestoreImageStartTimestamp : Uint8B
+0x088 ResumeHiberFileTicks : Uint8B
+0x090 ResumeIoTicks    : Uint8B
+0x098 ResumeDecompressTicks : Uint8B
+0x0a0 ResumeAllocateTicks : Uint8B
+0x0a8 ResumeUserInOutTicks : Uint8B
+0x0b0 ResumeMapTicks   : Uint8B
+0x0b8 ResumeUnmapTicks : Uint8B
+0x0c0 ResumeChecksumTicks : Uint8B
+0x0c8 ResumeChecksumIoTicks : Uint8B
+0x0d0 ResumeKernelSwitchTimestamp : Uint8B
+0x0d8 CyclesPerMs      : Uint8B
+0x0e0 WriteLogDataTimestamp : Uint8B
+0x0e8 KernelReturnFromHandler : Uint8B
+0x0f0 TimeStampCounterAtSwitchTime : Uint8B
+0x0f8 HalTscOffset     : Uint8B
+0x100 HvlTscOffset     : Uint8B
+0x108 SleeperThreadEnd : Uint8B
+0x110 PostCmosUpdateTimestamp : Uint8B
+0x118 KernelReturnSystemPowerStateTimestamp : Uint8B
+0x120 IoBoundedness    : Uint8B
+0x128 KernelDecompressTicks : Uint8B
+0x130 KernelIoTicks    : Uint8B
+0x138 KernelCopyTicks  : Uint8B
+0x140 ReadCheckCount   : Uint8B
+0x148 KernelInitTicks  : Uint8B
+0x150 KernelResumeHiberFileTicks : Uint8B
+0x158 KernelIoCpuTicks : Uint8B
+0x160 KernelSharedBufferTicks : Uint8B
+0x168 KernelAnimationTicks : Uint8B
+0x170 KernelChecksumTicks : Uint8B
+0x178 KernelChecksumIoTicks : Uint8B
+0x180 AnimationStart   : _LARGE_INTEGER
   +0x000 LowPart          : Uint4B
   +0x004 HighPart         : Int4B
   +0x000 u                : <anonymous-tag>
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
   +0x000 QuadPart         : Int8B
+0x188 AnimationStop    : _LARGE_INTEGER
   +0x000 LowPart          : Uint4B
   +0x004 HighPart         : Int4B
   +0x000 u                : <anonymous-tag>
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
   +0x000 QuadPart         : Int8B
+0x190 DeviceResumeTime : Uint4B
+0x198 SecurePagesProcessed : Uint8B
+0x1a0 BootPagesProcessed : Uint8B
+0x1a8 KernelPagesProcessed : Uint8B
+0x1b0 BootBytesWritten : Uint8B
+0x1b8 KernelBytesWritten : Uint8B
+0x1c0 BootPagesWritten : Uint8B
```

```
   +0x1c8 KernelPagesWritten : Uint8B
   +0x1d0 BytesWritten     : Uint8B
   +0x1d8 PagesWritten     : Uint4B
   +0x1dc FileRuns         : Uint4B
   +0x1e0 NoMultiStageResumeReason : Uint4B
   +0x1e4 MaxHuffRatio     : Uint4B
   +0x1e8 AdjustedTotalResumeTime : Uint8B
   +0x1f0 ResumeCompleteTimestamp : Uint8B
+0x280 FirmwareRuntimeInformationPages : Uint4B
+0x288 FirmwareRuntimeInformation : [1] Uint8B
+0x290 SpareUlong       : Uint4B
+0x294 NoBootLoaderLogPages : Uint4B
+0x298 BootLoaderLogPages : [24] Uint8B
+0x358 NotUsed          : Uint4B
+0x35c ResumeContextCheck : Uint4B
+0x360 ResumeContextPages : Uint4B
+0x364 Hiberboot        : UChar
+0x365 SecureLaunched   : UChar
+0x366 SecureBoot       : UChar
+0x368 HvPageTableRoot  : Uint8B
+0x370 HvEntryPoint     : Uint8B
+0x378 HvReservedTransitionAddress : Uint8B
+0x380 HvReservedTransitionAddressSize : Uint8B
+0x388 BootFlags        : Uint8B
+0x390 RestoreProcessorStateRoutine : Uint8B
+0x398 HighestPhysicalPage : Uint8B
+0x3a0 BitlockerKeyPfns : [4] Uint8B
+0x3c0 HardwareSignature : Uint4B
+0x3c8 SMBiosTablePhysicalAddress : _LARGE_INTEGER
   +0x000 LowPart          : Uint4B
   +0x004 HighPart         : Int4B
   +0x000 u                : <anonymous-tag>
      +0x000 LowPart          : Uint4B
      +0x004 HighPart         : Int4B
   +0x000 QuadPart         : Int8B
+0x3d0 SMBiosTableLength : Uint4B
+0x3d4 SMBiosMajorVersion : UChar
+0x3d5 SMBiosMinorVersion : UChar
+0x3d6 HiberResumeXhciHandoffSkip : UChar
+0x3d7 InitializeUSBCore : UChar
+0x3d8 ValidUSBCoreId   : UChar
+0x3d9 USBCoreId        : UChar
+0x3da SkipMemoryMapValidation : UChar
```

Appendix 8 – Comparing the header's fields across the versions

Table 27 compare the content of the PO_MEMORY_IMAGE structure for Windows 10 versions 1607,1809,1903,1909. When Mentioning "O:0x*xxx*" for version 1607, this means that the variable existed in that versions at this offset.

Table 26. Comparing the content of PO_MEMORY_IMAGE structure.

Offset	Offset from start of the file	Length	Offset content	V1607	V1809	V1903	V1909
0x000	0x000	Uint4B	Signature	✓	✓	✓	✓
0x004	0x004	Uint4B	ImageType	✓	✓	✓	✓
0x008	0x008	Uint4B	CheckSum	✓	✓	✓	✓
0x00c	0x00c	Uint4B	LengthSelf	✓	✓	✓	✓
0x010	0x010	Uint8B	PageSelf	✓	✓	✓	✓
0x018	0x018	Uint4B	PageSize	✓	✓	✓	✓
0x020	0x020	_LARGE_INTEGER	SystemTime	✓	✓	✓	✓
0x028	0x028	Uint8B	InterruptTime	✓	✓	✓	✓
0x030	0x030	Uint8B	FeatureFlags	✓	✓	✓	✓
0x038	0x038	UChar	HiberFlags	✓	✓	✓	✓
0x039	0x039	[3] UChar	spare	✓	✓	✓	✓
0x03c	0x03c	Uint4B	NoHiberPtes	✓	✓	✓	✓
0x040	0x040	Uint8B	HiberVa	✓	✓	✓	✓
0x048	0x048	Uint4B	NoFreePages	✓	✓	✓	✓
0x04c	0x04c	Uint4B	FreeMapCheck	✓	✓	✓	✓
0x050	0x050	Uint4B	WakeCheck	✓	✓	✓	✓
0x058	0x058	Uint8B	NumPagesForLoader	✓	✓	✓	✓
0x060	0x060	Uint8B	FirstSecureRestorePage	✓	✓	✓	✓
0x068	0x068	Uint8B	FirstBootRestorePage	✓	✓	✓	✓
0x070	0x070	Uint8B	FirstKernelRestorePage	✓	✓	✓	✓
0x078	0x078	Uint8B	FirstChecksumRestorePage	✓	✓	✓	✓
0x080	0x080	Uint8B	NoChecksumEntries	✓	✓	✓	✓
0x088	0x088	_PO_HIBER_PERF	PerfInfo	✓	✓	✓	✓
0x280	0x280	Uint4B	FirmwareRuntimeInformationPages	O: 0x270	✓	✓	✓
0x288	0x288	[1] Uint8B	FirmwareRuntimeInformation	O: 0x278	✓	✓	✓
0x290	0x290	Uint4B	SpareUlong	N/A	✓	✓	✓
		Uint4B	SiLogOffset	O: 0x280	N/A	N/A	N/A
0x294	0x294	Uint4B	NoBootLoaderLogPages	O: 0x284	✓	✓	✓
0x298	0x298	[24] Uint8B	BootLoaderLogPages	O: 0x288	✓	✓	✓
0x358	0x358	Uint4B	NotUsed	O: 0x348	✓	✓	✓
0x35c	0x35c	Uint4B	ResumeContextCheck	O: 0x34c	✓	✓	✓
0x360	0x360	Uint4B	ResumeContextPages	O: 0x350	✓	✓	✓
0x364	0x364	UChar	Hiberboot	O: 0x354	✓	✓	✓
0x365	0x365	UChar	SecureLaunched	N/A	✓	✓	✓
0x366	0x366	UChar	SecureBoot	N/A	✓	✓	✓
0x368	0x368	Uint8B	HvPageTableRoot	N/A	✓	N/A	N/A
0x368	0x368	Uint8B	HvCr3	O: 0x358	N/A	✓	✓
0x370	0x370	Uint8B	HvEntryPoint	O: 0x360	✓	✓	✓
0x378	0x378	Uint8B	HvReservedTransitionAddress	O: 0x368	✓	✓	✓
0x380	0x380	Uint8B	HvReservedTransitionAddressSize	O: 0x370	✓	✓	✓
0x388	0x388	Uint8B	BootFlags	O: 0x378	✓	✓	✓
		Uint8B	HalEntryPointPhysical	O 0x380	N/A	N/A	N/A
0x390	0x390	Uint8B	RestoreProcessorStateRoutine	N/A	✓	✓	✓
0x398	0x398	Uint8B	HighestPhysicalPage	O: 0x388	✓	✓	✓
0x3a0	0x3a0	[4] Uint8B	BitlockerKeyPfns	O: 0x390	✓	✓	✓
0x3c0	0x3c0	Uint4B	HardwareSignature	O: 0x3b0	✓	✓	✓
0x3c8	0x3c8	_LARGE_INTEGER	SMBiosTablePhysicalAddress	O: 0x3b8	✓	✓	✓
0x3d0	0x3d0	Uint4B	SMBiosTableLength	O: 0x3c0	✓	✓	✓
0x3d4	0x3d4	UChar	SMBiosMajorVersion	O: 0x3c4	✓	✓	✓
0x3d5	0x3d5	UChar	SMBiosMinorVersion	O: 0x3c5	✓	✓	✓
0x3d6	0x3d6	UChar	HiberResumeXhciHandoffSkip	N/A	✓	✓	✓
0x3d7	0x3d7	UChar	InitializeUSBCore	N/A	✓	✓	✓
0x3d8	0x3d8	UChar	ValidUSBCoreId	N/A	✓	✓	✓
0x3d9	0x3d9	UChar	USBCoreId	N/A	✓	✓	✓
0x3da	0x3da	UChar	SkipMemoryMapValidation	N/A	✓	✓	✓

Appendix 9 – “ _ KPROCESSOR_STATE” structure

This output was extracted from Windows 10 version 1809. We eliminated the results of Windows 10 versions 1903 and 1909 as they are the same as the below results.

```
lkd> dt  _Kprocessor_state
nt!_KPROCESSOR_STATE
   +0x000 SpecialRegisters : _KSPECIAL_REGISTERS
   +0x0f0 ContextFrame     : _CONTEXT

lkd> dt -b _Kprocessor_state
nt!_KPROCESSOR_STATE
   +0x000 SpecialRegisters : _KSPECIAL_REGISTERS
      +0x000 Cr0              : Uint8B
      +0x008 Cr2              : Uint8B
      +0x010 Cr3              : Uint8B
      +0x018 Cr4              : Uint8B
      +0x020 KernelDr0        : Uint8B
      +0x028 KernelDr1        : Uint8B
      +0x030 KernelDr2        : Uint8B
      +0x038 KernelDr3        : Uint8B
      +0x040 KernelDr6        : Uint8B
      +0x048 KernelDr7        : Uint8B
      +0x050 Gdtr             : _KDESCRIPTOR
         +0x000 Pad              : Uint2B
         +0x006 Limit            : Uint2B
         +0x008 Base             : Ptr64
      +0x060 Idtr             : _KDESCRIPTOR
         +0x000 Pad              : Uint2B
         +0x006 Limit            : Uint2B
         +0x008 Base             : Ptr64
      +0x070 Tr               : Uint2B
      +0x072 Ldtr             : Uint2B
      +0x074 MxCsr            : Uint4B
      +0x078 DebugControl     : Uint8B
      +0x080 LastBranchToRip  : Uint8B
      +0x088 LastBranchFromRip : Uint8B
      +0x090 LastExceptionToRip : Uint8B
      +0x098 LastExceptionFromRip : Uint8B
      +0x0a0 Cr8              : Uint8B
      +0x0a8 MsrGsBase        : Uint8B
      +0x0b0 MsrGsSwap        : Uint8B
      +0x0b8 MsrStar          : Uint8B
      +0x0c0 MsrLStar         : Uint8B
      +0x0c8 MsrCStar         : Uint8B
      +0x0d0 MsrSyscallMask   : Uint8B
      +0x0d8 Xcr0             : Uint8B
      +0x0e0 MsrFsBase        : Uint8B
      +0x0e8 SpecialPadding0  : Uint8B
   +0x0f0 ContextFrame     : _CONTEXT
      +0x000 P1Home           : Uint8B
      +0x008 P2Home           : Uint8B
      +0x010 P3Home           : Uint8B
      +0x018 P4Home           : Uint8B
      +0x020 P5Home           : Uint8B
      +0x028 P6Home           : Uint8B
      +0x030 ContextFlags     : Uint4B
```

```
+0x034 MxCsr           : Uint4B
+0x038 SegCs           : Uint2B
+0x03a SegDs           : Uint2B
+0x03c SegEs           : Uint2B
+0x03e SegFs           : Uint2B
+0x040 SegGs           : Uint2B
+0x042 SegSs           : Uint2B
+0x044 EFlags          : Uint4B
+0x048 Dr0             : Uint8B
+0x050 Dr1             : Uint8B
+0x058 Dr2             : Uint8B
+0x060 Dr3             : Uint8B
+0x068 Dr6             : Uint8B
+0x070 Dr7             : Uint8B
+0x078 Rax             : Uint8B
+0x080 Rcx             : Uint8B
+0x088 Rdx             : Uint8B
+0x090 Rbx             : Uint8B
+0x098 Rsp             : Uint8B
+0x0a0 Rbp             : Uint8B
+0x0a8 Rsi             : Uint8B
+0x0b0 Rdi             : Uint8B
+0x0b8 R8              : Uint8B
+0x0c0 R9              : Uint8B
+0x0c8 R10             : Uint8B
+0x0d0 R11             : Uint8B
+0x0d8 R12             : Uint8B
+0x0e0 R13             : Uint8B
+0x0e8 R14             : Uint8B
+0x0f0 R15             : Uint8B
+0x0f8 Rip             : Uint8B
+0x100 FltSave         : _XSAVE_FORMAT
   +0x000 ControlWord      : Uint2B
   +0x002 StatusWord       : Uint2B
   +0x004 TagWord          : UChar
   +0x005 Reserved1        : UChar
   +0x006 ErrorOpcode      : Uint2B
   +0x008 ErrorOffset      : Uint4B
   +0x00c ErrorSelector    : Uint2B
   +0x00e Reserved2        : Uint2B
   +0x010 DataOffset       : Uint4B
   +0x014 DataSelector     : Uint2B
   +0x016 Reserved3        : Uint2B
   +0x018 MxCsr            : Uint4B
   +0x01c MxCsr_Mask       : Uint4B
   +0x020 FloatRegisters   : _M128A
      +0x000 Low           : Uint8B
      +0x008 High          : Int8B
   +0x0a0 XmmRegisters     : _M128A
      +0x000 Low           : Uint8B
      +0x008 High          : Int8B
   +0x1a0 Reserved4        : UChar
+0x100 Header          : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x120 Legacy          : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x1a0 Xmm0            : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x1b0 Xmm1               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x1c0 Xmm2               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x1d0 Xmm3               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x1e0 Xmm4               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x1f0 Xmm5               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x200 Xmm6               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x210 Xmm7               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x220 Xmm8               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x230 Xmm9               : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x240 Xmm10              : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x250 Xmm11              : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x260 Xmm12              : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x270 Xmm13              : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x280 Xmm14              : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x290 Xmm15              : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x300 VectorRegister   : _M128A
   +0x000 Low             : Uint8B
   +0x008 High            : Int8B
+0x4a0 VectorControl    : Uint8B
+0x4a8 DebugControl     : Uint8B
+0x4b0 LastBranchToRip  : Uint8B
+0x4b8 LastBranchFromRip : Uint8B
+0x4c0 LastExceptionToRip : Uint8B
+0x4c8 LastExceptionFromRip : Uint8B
```

Appendix 10 – Comparing the processor context field's names

Tables 28 and 29 compare the content of the process context page across Windows 10 versions 1607, 1809, 1903, 1909.

Table 27. Content of the _KSPECIAL_REGISTERS structure (without recursion).

Offset	Offset from beginning of the file	Length	Offset content	Version 1607	Version 1809	Version 1903	Version 1909
+0x000	+0x1000	Uint8B	Cr0	✓	✓	✓	✓
+0x008	+0x1008	Uint8B	Cr2	✓	✓	✓	✓
+0x010	+0x1010	Uint8B	Cr3	✓	✓	✓	✓
+0x018	+0x1018	Uint8B	Cr4	✓	✓	✓	✓
+0x020	+0x1020	Uint8B	KernelDr0	✓	✓	✓	✓
+0x028	+0x1028	Uint8B	KernelDr1	✓	✓	✓	✓
+0x030	+0x1030	Uint8B	KernelDr2	✓	✓	✓	✓
+0x038	+0x1038	Uint8B	KernelDr3	✓	✓	✓	✓
+0x040	+0x1040	Uint8B	KernelDr6	✓	✓	✓	✓
+0x048	+0x1048	Uint8B	KernelDr7	✓	✓	✓	✓
+0x050	+0x1050	_KDESCRIPTOR	Gdtr	✓	✓	✓	✓
+0x060	+0x1060	_KDESCRIPTOR	Idtr	✓	✓	✓	✓
+0x070	+0x1070	Uint2B	Tr	✓	✓	✓	✓
+0x072	+0x1072	Uint2B	Ldtr	✓	✓	✓	✓
+0x074	+0x1074	Uint4B	MxCsr	✓	✓	✓	✓
+0x078	+0x1078	Uint8B	DebugControl	✓	✓	✓	✓
+0x080	+0x1080	Uint8B	LastBranchToRip	✓	✓	✓	✓
+0x088	+0x1088	Uint8B	LastBranchFromRip	✓	✓	✓	✓
+0x090	+0x1090	Uint8B	LastExceptionToRip	✓	✓	✓	✓
+0x098	+0x1098	Uint8B	LastExceptionFromRip	✓	✓	✓	✓
+0x0a0	+0x10a0	Uint8B	Cr8	✓	✓	✓	✓
+0x0a8	+0x10a8	Uint8B	MsrGsBase	✓	✓	✓	✓
+0x0b0	+0x10b0	Uint8B	MsrGsSwap	✓	✓	✓	✓
+0x0b8	+0x10b8	Uint8B	MsrStar	✓	✓	✓	✓
+0x0c0	+0x10c0	Uint8B	MsrLStar	✓	✓	✓	✓
+0x0c8	+0x10c8	Uint8B	MsrCStar	✓	✓	✓	✓
+0x0d0	+0x10d0	Uint8B	MsrSyscallMask	✓	✓	✓	✓
+0x0d8	+0x10d8	Uint8B	Xcr0	✓	✓	✓	✓
+0x0e0	+0x10e0	Uint8B	MsrFsBase	N/A	✓	✓	✓
+0x0e8	+0x10e8	Uint8B	SpecialPadding0	N/A	✓	✓	✓

Table 28. “ _CONTEXT” structure (without recursion).

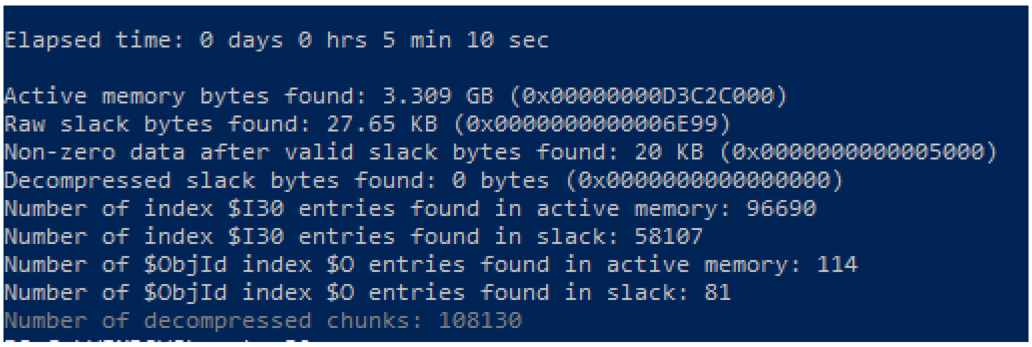
Offset	Length	Offset content	V1607	V1809	V1903	V1909
+0x000	Uint8B	P1Home	✓	✓	✓	✓
+0x008	Uint8B	P2Home	✓	✓	✓	✓
+0x010	Uint8B	P3Home	✓	✓	✓	✓
+0x018	Uint8B	P4Home	✓	✓	✓	✓
+0x020	Uint8B	P5Home	✓	✓	✓	✓
+0x028	Uint8B	P6Home	✓	✓	✓	✓
+0x030	Uint4B	ContextFlags	✓	✓	✓	✓
+0x034	Uint4B	MxCsr	✓	✓	✓	✓
+0x038	Uint2B	SegCs	✓	✓	✓	✓
+0x03a	Uint2B	SegDs	✓	✓	✓	✓
+0x03c	Uint2B	SegEs	✓	✓	✓	✓
+0x03e	Uint2B	SegFs	✓	✓	✓	✓
+0x040	Uint2B	SegGs	✓	✓	✓	✓
+0x042	Uint2B	SegSs	✓	✓	✓	✓
+0x044	Uint4B	EFlags	✓	✓	✓	✓
+0x048	Uint8B	Dr0	✓	✓	✓	✓
+0x050	Uint8B	Dr1	✓	✓	✓	✓
+0x058	Uint8B	Dr2	✓	✓	✓	✓
+0x060	Uint8B	Dr3	✓	✓	✓	✓
+0x068	Uint8B	Dr6	✓	✓	✓	✓
+0x070	Uint8B	Dr7	✓	✓	✓	✓
+0x078	Uint8B	Rax	✓	✓	✓	✓
+0x080	Uint8B	Rcx	✓	✓	✓	✓
+0x088	Uint8B	Rdx	✓	✓	✓	✓
+0x090	Uint8B	Rbx	✓	✓	✓	✓
+0x098	Uint8B	Rsp	✓	✓	✓	✓
+0x0a0	Uint8B	Rbp	✓	✓	✓	✓
+0x0a8	Uint8B	Rsi	✓	✓	✓	✓
+0x0b0	Uint8B	Rdi	✓	✓	✓	✓
+0x0b8	Uint8B	R8	✓	✓	✓	✓
+0x0c0	Uint8B	R9	✓	✓	✓	✓
+0x0c8	Uint8B	R10	✓	✓	✓	✓
+0x0d0	Uint8B	R11	✓	✓	✓	✓
+0x0d8	Uint8B	R12	✓	✓	✓	✓
+0x0e0	Uint8B	R13	✓	✓	✓	✓
+0x0e8	Uint8B	R14	✓	✓	✓	✓
+0x0f0	Uint8B	R15	✓	✓	✓	✓
+0x0f8	Uint8B	Rip	✓	✓	✓	✓
+0x100	_XSAVE_FORMAT	FltSave	✓	✓	✓	✓
+0x100	[2] _M128A	Header	✓	✓	✓	✓
+0x120	[8] _M128A	Legacy	✓	✓	✓	✓
+0x1a0	_M128A	Xmm0	✓	✓	✓	✓
+0x1b0	_M128A	Xmm1	✓	✓	✓	✓
+0x1c0	_M128A	Xmm2	✓	✓	✓	✓
+0x1d0	_M128A	Xmm3	✓	✓	✓	✓
+0x1e0	_M128A	Xmm4	✓	✓	✓	✓
+0x1f0	_M128A	Xmm5	✓	✓	✓	✓
+0x200	_M128A	Xmm6	✓	✓	✓	✓
+0x210	_M128A	Xmm7	✓	✓	✓	✓
+0x220	_M128A	Xmm8	✓	✓	✓	✓
+0x230	_M128A	Xmm9	✓	✓	✓	✓
+0x240	_M128A	Xmm10	✓	✓	✓	✓
+0x250	_M128A	Xmm11	✓	✓	✓	✓
+0x260	_M128A	Xmm12	✓	✓	✓	✓
+0x270	_M128A	Xmm13	✓	✓	✓	✓
+0x280	_M128A	Xmm14	✓	✓	✓	✓
+0x290	_M128A	Xmm15	✓	✓	✓	✓
+0x300	[26] _M128A	VectorRegister	✓	✓	✓	✓
+0x4a0	Uint8B	VectorControl	✓	✓	✓	✓
+0x4a8	Uint8B	DebugControl	✓	✓	✓	✓
+0x4b0	Uint8B	LastBranchToRip	✓	✓	✓	✓
+0x4b8	Uint8B	LastBranchFromRip	✓	✓	✓	✓
+0x4c0	Uint8B	LastExceptionToRip	✓	✓	✓	✓
+0x4c8	Uint8B	LastExceptionFromRip	✓	✓	✓	✓

Appendix 11 – Comparison between Hibr2Bin and the free mode of Hibernation Recon

Table 29. Comparison between Hibr2Bin and the free mode of Hibernation Recon.

	Hibr2Bin	Hibernation Recon (Free mode)
Tool Layout	CLI	GUI, and CLI
Automatic detection of Windows version profile	N/a	Available
CLI Trace options	No specific options	Trace options in CLI: no messages, errors, warnings, informational and verbose
Output file	Single binary file for the decompressed hibernation file	Binary file for decompressed Active memory One or more binary files for Slack data
Showing the time taken for file conversion	N/A	Available in GUI and CLI

We also noticed a minor difference in conversion time between both tools, to the favor of Hibernation Recon. Hibernation Recon provides some statistics about the content of the file, even when using free mode.



```
Elapsed time: 0 days 0 hrs 5 min 10 sec

Active memory bytes found: 3.309 GB (0x00000000D3C2C000)
Raw slack bytes found: 27.65 KB (0x0000000000006E99)
Non-zero data after valid slack bytes found: 20 KB (0x0000000000005000)
Decompressed slack bytes found: 0 bytes (0x0000000000000000)
Number of index $I30 entries found in active memory: 96690
Number of index $I30 entries found in slack: 58107
Number of $ObjId index $O entries found in active memory: 114
Number of $ObjId index $O entries found in slack: 81
Number of decompressed chunks: 108130
```

Figure 63. Example of statistics shown by the free mode of Hibernation Recon CLI.

It is to be mentioned that both tools do not proceed with hibernation files with "WAKE" signatures, as such files do not contain memory pages. Both tools provide a notification that the file has a "WAKE" signature.

Appendix 12 – List of successfully tested Volatility plugins

The plugins mentioned in tables 30, were successfully tested on decompressed hibernation files of Windows 10 version 1809 , 1903. For v1909, it was successfully tested using Volatility profile for Win10x64_18362.

Table 30. List of Volatility plugins successfully tested on Test case D files.

Plugin category [84]	Plugin name	Description[39]
Image Identification	imageinfo	Identify information for the image
	kdbgscan	Search for and dump potential KDBG values
Processes and DLLs	pslist	Print all running processes by following the EPROCESS lists
	psscan	Pool scanner for process objects
	pstree	Print process list as a tree
	psxview	Find hidden processes with various process listings
	dlllist	Print list of loaded DLLs for each process
	dlldump	Dump DLLs from a process address space
	envars	Display process environment variables
	verinfo	Prints out the version information from PE images
	ldrmodules	Detect unlinked DLLs
	apihooks	Detect API hooks in process and kernel memory
Process Memory	memmap	Print the memory map
	memdump	Dump the addressable memory for a process
	procdump	Dump a process to an executable file sample
	vadinfo	Dump the VAD info
	vadtree	Walk the VAD tree and display in tree format
	vadwalk	Walk the VAD tree
	vaddump	Dumps out the vad sections to a file
Kernel Memory and Objects	moddump	Dump a kernel driver to an executable file sample
	modscan	Pool scanner for kernel modules
	modules	Print list of loaded modules
	ssdt	Display SSDT entries
	threads	Investigate _ETHREAD and _KTHREADs
	thrdscan	Pool scanner for thread objects
	driverscan	Pool scanner for driver objects
	filescan	Pool scanner for file objects
	mutantscan	Pool scanner for mutex objects
	dumpfiles	Extract memory mapped and cached files
Registry	getservicesids	Get the names of services in the Registry and return Calculated SID
Networking	netscan	Scan a Vista (or later) image for connections and sockets
File System	mftparser	Scans for and parses potential MFT entries
Miscellaneous	timeliner	Creates a timeline from various artifacts in memory
	bigpools	Dump the big page pools using BigPagePoolScanner
	callbacks	Print system-wide notification routines
	cmdline	Display process command-line arguments
	devicetree	Show device tree
	drivermodule	Associate driver objects to kernel modules
	driverscan	Pool scanner for driver objects
	joblinks	Print process job link information
	objtypescan	Scan for Windows object type objects
	pooltracker	Show a summary of pool tag usage
	filescan	Pool scanner for file objects
	timeliner	Creates a timeline from various artifacts in memory
	timers	Print kernel timers and associated module DPCs modules
	devicetree	Show device tree
	hivescan	Pool scanner for registry hives
	hivelist	Print list of registry hives.