Points to Watch when Acquiring Windows Memory Images

2.1 Acquiring Memory Images on Windows

In Vol. 45, we discussed the acquisition of forensic memory images on Linux[*1]. In this edition, we discuss the acquisition of memory images on Windows.

We use tools such as those listed in Table 1 to acquire full memory images of Windows systems. We use the Volatility Framework[*2] and Rekall Memory Forensic Framework[*3] to analyze the images.

Windows version upgrades, however, can come with changes to the memory management framework to improve security and performance. So you need to use tools compatible with the new specifications to acquire and analyze memory images. Not only do we cover memory image acquisition tools here, we also discuss some key points to watch when actually acquiring and analyzing images. We also suggest reliable ways of acquiring individual process dumps.

2.2 Points to Watch when Acquiring/Analyzing Memory Images

In this edition, we explain how to deal with the three features: the paging files, memory compression, and Virtual Secure Mode. Paging files existed in Windows prior to version 10, but other features were added in updates after the Windows 10 release.

■ Paging Files

Windows saves a process's paged-out virtual memory pages in a paging file called C:\pagefile.sys. Figure 1 shows the result of extracting notepade.exe from a Windows 10 1809 memory image using Volatility's procdump tool. This was done right after notepad.exe started, so the process dump was successful. Figure 2, meanwhile, shows the result of trying to dump notepad.exe from a memory image taken on the same system after memory usage had jumped. This was unsuccessful because of a page-out. The memory

Table 1: Examples of Memory Image Acquisition Tools

Tool	Vendor
WinPmem memory imager	https://winpmem.velocidex.com/
Comae Technologies	https://www.comae.com/
MAGNET RAM Capture	https://www.magnetforensics.com/resources/magnet-ram-capture/
Belkasoft RAM Capturer	https://belkasoft.com/ram-capturer

```
>vol.py --profile Win10x64_17763 -f "Windows 10 1809 Feb x64 en-Snapshot82.vmem" procdump --pid=4596 -D procdump
Volatility Foundation Volatility Framework 2.6.1
Process(V)         ImageBase          Name                 Result
------------------ ------------------ -------------------- ------
0xffffaf86ab950480 0x00007ff7413c0000 notepad.exe          OK: executable.4596.exe
```

Figure 1: A Successful Volatility procdump

```
>vol.py --profile Win10x64_17763 -f "Windows 10 1809 Feb x64 en-Snapshot83.vmem" procdump --pid=4596 -D procdump
Volatility Foundation Volatility Framework 2.6.1
Process(V)         ImageBase          Name                 Result
------------------ ------------------ -------------------- ------
0xffffaf86ab950480 0x00007ff7413c0000 notepad.exe          Error: ImageBaseAddress at 0x7ff7413c0000 is unavailable (possibly due to paging)
```

Figure 2: An Unsuccessful Volatility procdump

*1 Internet Infrastructure Review (IIR) Vol. 45, Focused Research (2): Acquiring Forensic Memory Images on Linux (https://www.iij.ad.jp/en/dev/iir/045.html).
*2 The Volatility Foundation (https://www.volatilityfoundation.org/).
*3 Rekall Forensics (http://www.rekall-forensic.com/).

images used for analysis in Figures 1 and 2 are VMware Workstation memory snapshots. We used these when preparing this article to make it a bit easier to acquire memory images in the state that we wanted.

The paging file stores paged-out memory data, so we want to use it in our analysis if possible. But many memory image acquisition tools do not collect this file. With WinPmem, the memory image and paging file can be obtained almost simultaneously by specifying pagefile.sys with the "-p" option (Figure 3). It can also be obtained using The Sleuth Kit or FTK Imager, but keeping the time interval between the memory image and pagefile.sys as short as possible averts discrepancies.

Obtaining pagefile.sys is pointless if the memory image analysis tool does not support it. Rekall can analyze memory images and pagefile.sys seamlessly as a single, complete memory image. Immediately after the outcome in Figure 2, we used the command in Figure 3 to acquire a memory image and pagefile.sys using WinPmem, and then using Rekall's procdump plugin, we were able to dump the notebook.exe process from those files (Figure 4). A look at the first part of the file shows that it is an MZ header (the procdump plugin dumps the specified process in PE format).

Volatility 2, on the other hand, cannot analyze pagefile.sys files like Rekall. Presentation slides released for OSDFCon 2019[*4], however, contain hints that the currently in-development Volatility 3 will support paging files as well as memory compression as described below, so it looks like Volatility will support paging file analysis in future.

The WinPmem version used in Figure 3 is 2.1 post4. The latest WinPmem as of this writing (Feb. 2020) is 3.3 rc3, but analyzing the generated AFF4 file with Rekall produced the error in Figure 6. We have not inspected all of the relevant

```
>winpmem-2.1.post4.exe -p c:\pagefile.sys -o memdump.aff4
```

Figure 3: Acquiring a Memory Image and pagefile.sys with WinPmem

```
$ rekal -f memdump.aff4 procdump --proc_regex="notepad*" --dump_dir="./"
Webconsole disabled: cannot import name 'webconsole_plugin'
                    _EPROCESS                       Filename
------------------------------------------------------ --------
0xaf86ab950480 notepad.exe                    4596 executable.notepad.exe_4596.exe
```

Figure 4: Running procdump with pagefile.sys

```
$ hexdump -C executable.notepad.exe_4596.exe | head -10
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 f8 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  15 48 94 65 51 29 fa 36  51 29 fa 36 51 29 fa 36  |.H.eQ).6Q).6Q).6|
00000090  58 51 69 36 4f 29 fa 36  34 4f f9 37 52 29 fa 36  |XQi6O).640.7R).6|
```

Figure 5: File Dumped by Rekall's procdump

```
$ rekal -f winpmem3.aff4 pslist
Traceback (most recent call last):
  File "/home/user01/Downloads/rekall/rekall-core/rekall/addrspace.py", line 519, in read_partial
    data = self._cache.Get(chunk_number)
  File "/home/user01/Downloads/rekall/rekall-lib/rekall_lib/utils.py", line 147, in NewFunction
    return f(self, *args, **kw)
  File "/home/user01/Downloads/rekall/rekall-lib/rekall_lib/utils.py", line 336, in Get
    raise KeyError(key)
KeyError: 0
(snip)
  File "/home/mkobayashi/envs/win10_rekall2/lib/python3.6/site-packages/pyaff4/aff4_image.py", line 432, in _ReadChunkFromBevy
    "Unable to process compression %s" % self.compression)
RuntimeError: Unable to process compression https://tools.ietf.org/html/rfc1951
```

Figure 6: Rekall Throws an Error when Passed an AFF4 File Acquired with WinPmem 3.x

*4 Volatility 3 Public Beta: The Insider's Preview (https://www.osdfcon.org/events_2019/volatility-3-public-beta-the-insiders-preview/).

source code, but it looks like the error is due to the default compression format for saved data as of WinPmem 3.3 rc2 having been changed to deflate, which Rekall does not support (the default in WinPmem 2.x is zlib). WinPmem's "-c" option specifies compression format, but we had the same processing error even when using zlib. Also, zlib was the default compression format in WinPmem 3.x versions before WinPmem 3.3 rc1, but using these versions to generate AFF4 files that contain the paging file and running them through Rekall produced a different error (AFF4 files without the paging file do work).

So when using Rekall as the analysis tool, WinPmem 2.1 post4 can create memory images that are less likely to cause problems during analysis. That said, we do not recommend using WinPmem 2.x because it is no longer being developed, it can cause forced shutdowns on Windows 10, and it does not support Virtual Secure Mode, which we discuss below. Note that Rekall development is effectively on hold; a new version has not been released since December 2017.

Hopefully, if development of Volatility and Rekall moves forward, they will eventually be able to analyze AFF4 files generated by WinPmem 3.x, but until such time, acquiring memory images and paging files using WinPmem 3.x and exporting the image as shown in Figure 7 is probably the better option. The exported memory image is in RAW format, so you can use either Volatility or Rekall for analysis.

Memory Compression

Paging of a process's virtual memory involves paging file reads and writes, which inevitably degrades system performance. SSDs have become widespread in recent years, so latency is not what it was with HDDs, but performance still unmistakably drops. But page-in and page-out performance can be improved by creating a dedicated area in memory to store paged-out pages in compressed form. The size of the compressed pages can be viewed in the Memory section of the Task Manager's Performance tab (red box in Figure 8). This framework was adopted from Windows 10 1511. Similar frameworks exist in macOS and Linux.

Analyzing memory images containing compressed memory data naturally requires a tool that can cope. Unfortunately, Volatility 2 and Rekall cannot currently analyze compressed memory data from any OS along with other memory pages in a seamless fashion.

```
>winpmem_v3.3.rc3.exe -dd -e */PhysicalMemory -D <export_dir> <image_file>.aff4
```

Figure 7: Exporting a Memory Image from an AFF4 File Generated by WinPmem 3.x

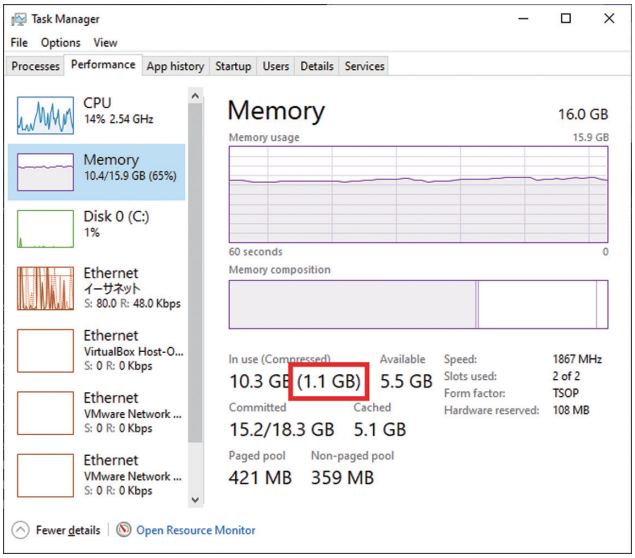

Figure 8: Size of the Compressed Memory

However, at SANS DFIR Summit Austin 2019[*5] and BlackHat USA 2019[*6], FireEye's Omar Sardar and Dimiter Andonov announced implementations of Volatility[*7] and Rekall[*8] that support Windows 10 memory compression.

Note that as both implementations only support Windows 10 1607 through 1809, memory images from Windows 10 1903 or later cannot be analyzed. As of this writing, the announced capability does not seem to have been incorporated into the developers' source code, but as mentioned earlier, Volatility is set to add support in the new version.

Figures 9 and 10 show the results of running the hashdump plugin on the original Volatility and Volatility with support for memory compression. The hashdump plugin retrieves a user's password hash from the registry hive read into memory. Since the user password hash is stored in compressed memory, original Volatility gives no output, but Volatility with support for memory compression is able to print out the hash.

Virtual Secure Mode

The Enterprise and Education editions of Windows 10 1511 and later, and Windows Server 2016 and later, introduce a virtualization-based security (VBS) isolation mechanism that uses virtual machines. Security mechanisms such as Device Guard and Credential Guard are implemented using VBS by executing virtual machines for specific functions in what is called Virtual Secure Mode (VSM). These features allow you to run integrity checks when loading drivers, place execution restrictions on applications, and protect credentials.

Running a tool without support for VSM, such as WimPmem 2.x, yields a BSoD as shown in Figure 11.

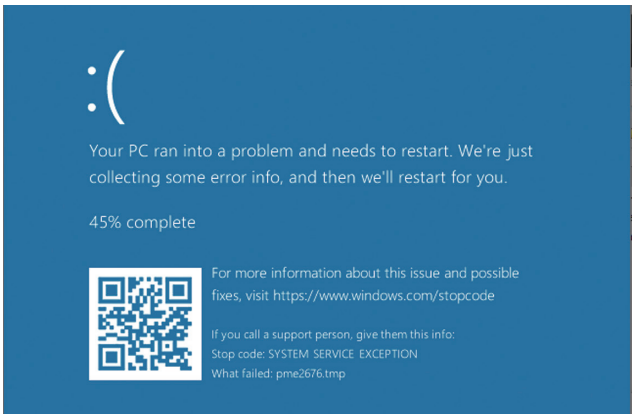

Figure 11: Running Memory Image Acquisition Tools with no VSM Support Yields a BSoD

```
> vol.py --profile Win10x64_17763 -f "Windows 10 1809 Feb x64 en-Snapshot64.vmem" hashdump
Volatility Foundation Volatility Framework 2.6.1
```

Figure 9: Result of Running hashdump Plugin on Original Volatility

```
> vol.py --profile Win10x64_17763 -f "Windows 10 1809 Feb x64 en-Snapshot64.vmem" hashdump
Volatility Foundation Volatility Framework 2.6.1
localuser01:1001:8492e81f418ee4da82b19ef1f27d39af:17e26cdcb1cf786246e7cf8373a540ca:::
```

Figure 10: Result of Running hashdump Plugin on Volatility with Memory Compression Support

*5 SANS DFIR Summit 2019 (https://www.sans.org/event/digital-forensics-summit-2019/summit-agenda).
*6 Paging All Windows Geeks – Finding Evil in Windows 10 Compressed Memory (https://www.blackhat.com/us-19/briefings/schedule/#paging-all-windows-geeks--finding-evil-in-windows--compressed-memory-15582).
*7 win10_volatility (https://github.com/fireeye/win10_volatility).
*8 win10_rekall (https://github.com/fireeye/win10_rekall).

By default, a BSoD causes the host to automatically reboot. The entire contents of memory will of course be erased, so you need to determine beforehand whether the tool you are using is compatible with VSM. Note that the latest versions of the tools in Table 1 do not trigger a BSoD, so consider updating if you are using an older version.

■ Which Memory Image Analysis Tool Should You Use?

Table 2 summarizes the types of data supported by the memory image analysis tools we have discussed. Of these, only Volatility 2 remains in active development and is thus essentially our recommendation. But Volatility with memory compression support should be used when analyzing memory images from Windows 10 1809 or earlier.

You need to use Rekall if you want to analyze paging files as well, but as development has stopped and it has compatibility issues with WinPmem 3.x, it's probably not a go-to tool for many situations. Unfortunately, none of the tools currently available cater to every case. But the situation looks set to improve with the release of Volatility 3.

2.3 Reliable Process Dumpings

So far, we have discussed precautions and strategies for acquiring memory images. But even with these measures, ensuring the integrity of captured memory images is difficult. If the host being analyzed is a VM, you can obtain a complete memory image by taking a snapshot. But on live systems, various processes will be running when you

Table 2: Comparison of Memory Image Acquisition Tools

Tool	Memory images		Paging file	Memory compression	Notes
	RAW	AFF4			
Volatility 2	✓				
Rekall	✓	✓*1	✓*2		Development has stopped
Volatility with memory compression support	✓			✓	Supports up to Windows 10 1809
Rekall with memory compression support	✓	✓*1	✓*2	✓	Supports up to Windows 10 1809

*1 Cannot parse AFF4 files generated by WinPmem 3.3 rc2 and later
*2 Cannot parse AFF4 files containing paging files generated by WinPmem 3.3 rc2 and later

capture the memory image, so the data in memory can change and page-outs can occur. So even if you analyze the memory image, you may find that the contents of process memory are not internally consistent.

Figure 12 compares the .text sections of the files dumped in Figure 1 (left) and Figure 4 (right). As Figure 5 showed, the MZ header was extracted correctly from the file dumped by Rekall, but a look at the .text sections of both files shows that the data values in the file dumped by Rekall are all 0x00 (red area in Figure 12). As discussed, this kind of situation is unavoidable, but it can hinder process analysis. In cases like this, dumping each process from userland individually can give internally consistent process dumps (dumping a process triggers memory access, causing the OS to page-in anything that has been paged out, enabling you to capture all of the process's virtual memory pages).

Several tools for dumping processes exist. Windows Sysinternals ProcDump*9 is a common one. Note that it is different from the plugins of the same name that exist for Volatility and Rekall. Also, Volatility's and Rekall's procdump generate PE format files, whereas Sysinternals ProcDump uses the crash dump format.

Running this with the command in Figure 13 will dump the process with an ID of 4596. You can also specify process name instead of process ID. It's also useful to pass in the

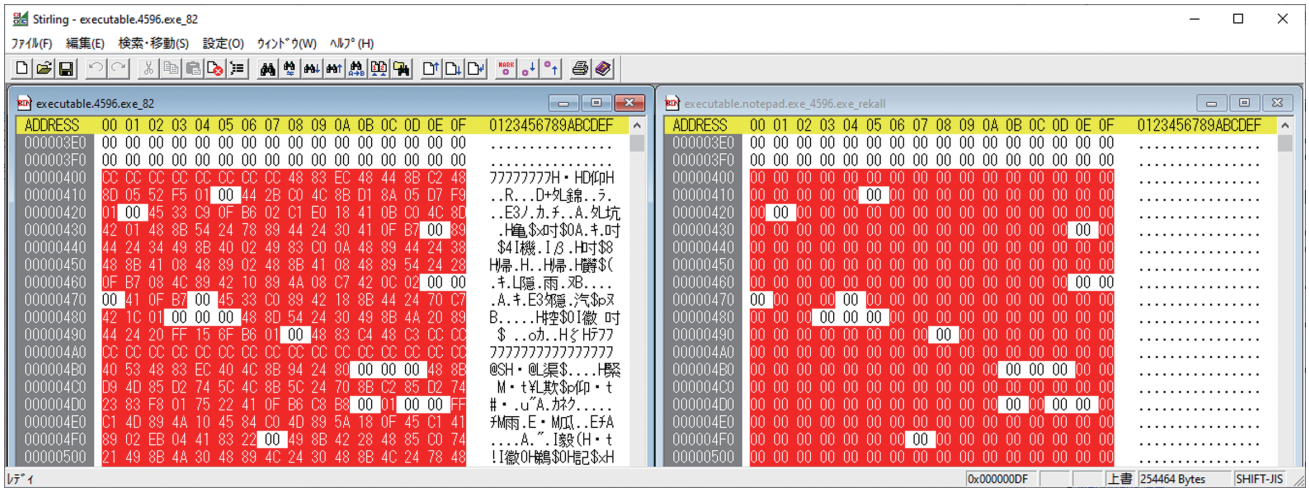

Figure 12: Process Dump Discrepancies

```
>procdump64.exe -ma 4596

ProcDump v9.0 - Sysinternals process dump utility
Copyright (C) 2009-2017 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com

[12:05:29] Dump 1 initiated: C:\Users\localuser01\Desktop\tools\notepad.exe_200206_120529.dmp
[12:05:29] Dump 1 writing: Estimated dump file size is 107 MB.
[12:05:32] Dump 1 complete: 107 MB written in 3.8 seconds
[12:05:33] Dump count reached.
```

Figure 13: Dumping a Process Using the ProcDump Command

*9 ProcDump - Windows Sysinternals (https://docs.microsoft.com/en-us/sysinternals/downloads/procdump).

"-ma" option to dump all of the memory used by the process. The crash dump generated can be read using tools like WinDbg[*10][*11]. Figure 14 is a screenshot of WinDbg reading in the crash dump generated by ProcDump after Figure 4. The code that was missing with Rekall's procdump is now present (red area in Figure 14). Using ProcDump like this ensures a reliable process dump.

2.4 Scripted Process Dumps and Order of Steps to Preserve Artifacts

Creating a script in PowerShell or the like makes it easy to run ProcDump for all processes. Dumping all processes, however, will yield tens of GB or more. This is no problem if there is enough space on the storage destination, but if you want to keep the data on a small USB memory stick or external SSD, compressing the data with a tool like 7-Zip[*12] after each dump is the way to go. You also need to specify the right compression command option to ensure that the temporary files created when compressing files are not written to the disk being analyzed. With 7-Zip, the "-w" option specifies the working folder, so you should use this to specify the disk on which you will be preserving the artifacts.

When running ProcDump, we skip protected processes like System and Registry (trying to dump those results in an error). These scripts need to run in various system environments, so they are often created using PowerShell (installed on Windows by default) or as a batch file, but there are some key points to note.

Launching the PowerShell prompt (powershell.exe) or command prompt (cmd.exe) will also launch the console

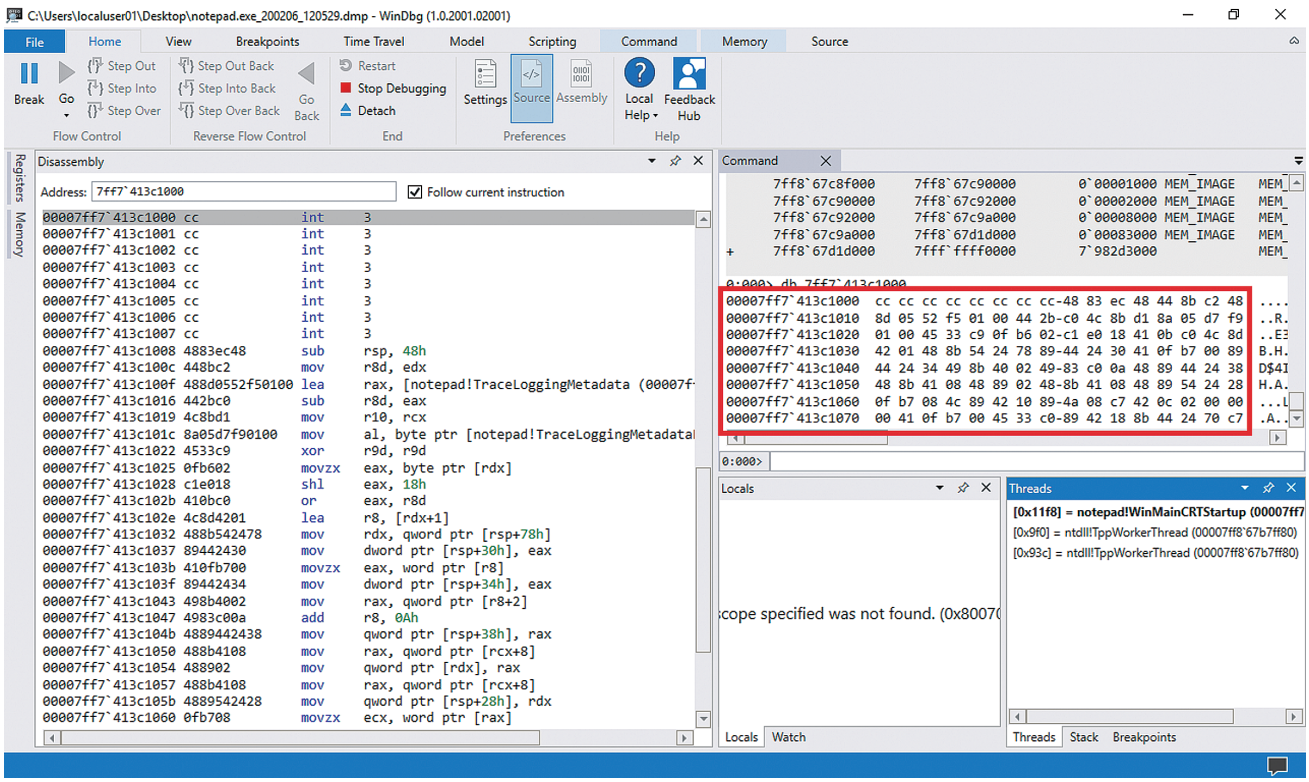

Figure 14: Inspecting the Process Dump in WinDbg

*10 Download Debugging Tools for Windows - WinDbg (https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools).

*11 Debugging Using WinDbg Preview (https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-using-windbg-preview).

*12 7-Zip (https://www.7-zip.org/).

window host (conhost.exe). If you try to dump the conhost.exe hosting the prompt that launched the PowerShell script or batch file that runs ProcDump, the ProcDump process will stop. This is because ProcDump suspends a process when dumping it. Since conhost.exe is the process handling the console's IO buffer and display, suspending this process also stops the ProcDump instance running in the console (Figure 15). If necessary, you can analyze conhost.exe by acquiring a memory image using WinPmem.

Depending on the system environment, executing the script can result in several hundred, or more, process dump and file compression cycles. So from a forensics point of view, it may be good practice to preserve the artifacts according to ordered steps such as those in Figure 16.

2.5 Conclusion

We have discussed key points to note when acquiring and analyzing memory images on Windows. We also looked at process dumping with a view to aiding image integrity when acquiring memory images. Many articles dealing with memory forensics imply that the preservation task can be completed by acquiring a memory image using WinPmem or the like, but we hope it is now evident that this is not always sufficient. That said, dumping and compressing all processes is time consuming, so you need to decide on whether to do this in accord with the situation and policies in effect when the incident response is initiated.

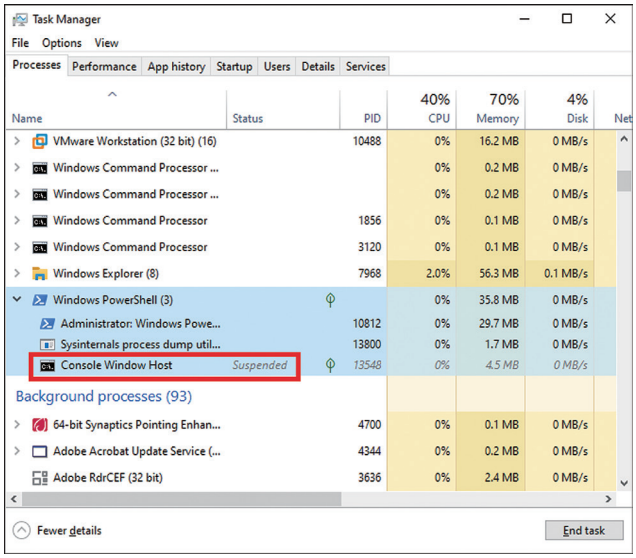

Figure 15: Dumping conhost.exe Causes ProcDump to stop

1. Memory image
2. Artifacts that can be preserved as individual files from disk, such as MFT, Prefetch, and event logs
3. Process dumps
4. Disk image

Figure 16: Example of Ordered Steps for Artifact Preservation

Minoru Kobayashi

Forensic Investigator, Office of Emergency Response and Clearinghouse for Security Information, Advanced Security Division, IIJ
Mr. Kobayashi is a member of IIJ-SECT, mainly dealing with digital forensics. He works to improve incident response capabilities and in-house technical capabilities.
He gives lectures and training sessions at security events both in Japan and abroad, including Black Hat, FIRST TC, JSAC, and Security Camp events.