Windowsのメモリイメージ取得の注意点

2.1 Windowsにおけるメモリイメージ取得

本レポートVol.45では、Linuxにおけるフォレンジック向けメモリイメージ取得について解説しました[*1]。今回はWindowsのメモリイメージ取得について解説します。

Windowsでシステム全体のメモリイメージを取得するには、表-1にあるようなツールを使用します。取得したメモリイメージの解析には、Volatility Framework[*2]やRekall Memory Forensic Framework[*3]を使用します。

しかし、Windowsのバージョンアップに伴い、セキュリティやシステムパフォーマンス向上のため、メモリ管理の仕組みが変更される場合があります。従って、メモリイメージの取得や解析に使用するツールも新しい仕様に合ったものを使う必要があります。本稿では、単にメモリイメージを取得するツールを紹介するのではなく、メモリイメージの取得時や解析時に注意すべき点について解説します。また、個々のプロセスダンプを確実に取得する方法も提案します。

2.2 メモリイメージ取得及び解析時の注意点

今回は、ページングファイル、メモリ圧縮、Virtual Secure Modeの3つの機能について、対応方法を解説します。ページングファイルはWindows 10より前のバージョンから存在していましたが、その他の機能は、Windows 10がリリースされた後のアップデートで追加された機能です。

■ ページングファイル

Windowsはページアウトするプロセスの仮想メモリのページを、C:¥pagefile.sysというページングファイルに保存します。図-1はWindows 10 1809のメモリイメージからVolatilityのprocdumpプラグインでnotepad.exeを抽出しようとした結果です。この時点ではnotepad.exeが起動直後であったため、プロセスのダンプに成功しています。一方、図-2は同じ環境でメモリ使用率が高くなった後のメモリイメージから、notepad.exeをダンプしようとした結果ですが、ページアウトによってプロセスダンプが失敗しています。なお、図-1、図-2で解析に使用したメモリイメージはVMware Workstationのメモリスナップ

表-1 メモリイメージ取得ツールの例

ツール	入手先
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
```

図-1 Volatility procdumpの成功例

```
>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)
```

図-2 Volatility procdumpの失敗例

＊1 Internet Infrastructure Review (IIR) Vol.45 2章フォーカス・リサーチ Linuxのフォレンジック向けメモリイメージ取得(https://www.iij.ad.jp/dev/report/iir/045/02.html)。
＊2 The Volatility Foundation(https://www.volatilityfoundation.org/)。
＊3 Rekall Forensics(http://www.rekall-forensic.com/)。

ショットです。本稿の執筆にあたり、ある程度意図した状態のメモリイメージを容易に取得するために使用しました。

ページングファイルにはページアウトされたメモリデータが保存されているため、可能であれば解析に活用したいところです。しかし、多くのメモリイメージ取得ツールは、このファイルを収集しません。WinPmemであれば、「-p」オプションでpagefile.sysを指定することで、メモリイメージとページングファイルをほぼ同時に取得することができます(図-3)。The Sleuth KitやFTK Imagerなどを使用して取得することもできますが、メモリイメージとpagefile.sysを取得する間隔はなるべく短い方が、内容の齟齬が生じにくいでしょう。

pagefile.sysを取得しても、メモリイメージ解析ツールが対応していなければ意味がありません。Rekallはメモリイメージとpagefile.sysを併せて透過的に1つのメモリイメージとして解析することができます。図-2の状態の直後に図-3のコマンドによりWinPmemでメモリイメージとpagefile.sysを取得し、Rekallのprocdumpプラグインを使って、それらのファイルからnotepad.exeをプロセスダンプすることができました(図-4)。ファイルの先頭部分を確認すると、MZヘッダであることが分かります(procdumpプラグインは指定したプロセスをPEフォーマットでダンプします)。

一方、Volatility 2は、Rekallのようにpagefile.sysを解析することはできません。しかし、現在開発中のVolatility 3に関するOSDFCon 2019での発表資料[*4]には、ページングファイルと後述するメモリ圧縮に対応することを示唆する記述があるため、今後はVolatilityでもページングファイルを使用した解析が可能になると思われます。

なお、図-3で使用したWinPmemはバージョン2.1 post4です。本稿執筆時(2020年2月)のWinPmemの最新バージョンは3.3 rc3ですが、これで生成したaff4ファイルをRekallで解析しようとすると、図-6のエラーが発生します。関連するソースコードをすべて調査したわけではありませんが、このエラー

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

図-3 WinPmemでメモリイメージとpagefile.sysを取得

```
$ 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
```

図-4 pagefile.sysを利用したprocdumpの実行例

```
$ 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|
```

図-5 Rekall procdumpでダンプしたファイル

```
$ 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
```

図-6 RekallはWinPmem 3.xで取得したaff4ファイルの解析時にエラーが発生する

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

は、WinPmem 3.3 rc2以降の保存データのデフォルト圧縮フォーマットがdeflateに変更され、Rekallがそれに未対応であることが原因のようです（WinPmem 2.xのデフォルトはzlibでした）。WinPmemの「-c」オプションで圧縮フォーマットを指定できますが、zlibを指定しても同様のエラーで解析することはできませんでした。なお、WinPmem 3.3 rc1以前のWinPmem 3.xもzlibがデフォルトの圧縮フォーマットですが、これらのバージョンでページングファイルを含むaff4ファイルを生成し、Rekallで解析した場合、上記とは異なるエラーが発生します（ページングファイルを含まないaff4ファイルは解析できます）。

従って、解析ツールとしてRekallを選択する場合、WinPmem 2.1 post4を使用する方が解析時に問題が起きにくいメモリイメージを作成することができます。しかし、WinPmem 2.xは開発が終了していること、Windows 10環境では強制終了してしまう場合があること、また、後述するVirtual Secure Modeに対応していないこともあり、使用はお勧めできません。なお、Rekallも2017年12月以降、新しいバージョンがリリースされておらず、事実上の開発停止状態にあります。

今後、VolatilityやRekallの開発が進めば、WinPmem 3.xで生成したaff4ファイルも解析できるようになることが期待できますので、それまではWinPmem 3.xでメモリイメージとページングファイルを取得し、図-7のようにエクスポートして解析するのがベターな対応でしょう。エクスポートしたメモリイメージはRAWフォーマットなので、VolatilityとRekallのどちらでも解析することができます。

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

図-7 WinPmem 3.xで作成したaff4ファイルから メモリイメージをエクスポート

■ メモリ圧縮

プロセスの仮想メモリのページングにはページングファイルの読み書きが発生するため、その分、どうしてもシステムのパフォーマンスが低下してしまいます。近年はSSDが普及したため、HDDほどのレイテンシは発生しませんが、それでもパフォーマンスが低下することに違いはありません。そこで、メモリ上に専用の領域を設け、そこにページアウトされるページを圧縮して保存することで、ページイン及びページアウト時のパフォーマンスの向上が図られました。圧縮されたページの容量はタスクマネージャのパフォーマンスタブ内にあるメモリの項目で確認できます（図-8中の赤枠）。このフレームワークは、Windows 10 1511以降で採用されています。同様のフレームワークは、macOSやLinuxにも存在します。

メモリ圧縮のデータが含まれたメモリイメージの解析には、当然ながら、それに対応した解析ツールが必要になります。残念ながら、現状のVolatility 2やRekallでは各OSのメモリ圧縮のデータを他のメモリページと併せて、透過的に解析することはできません。

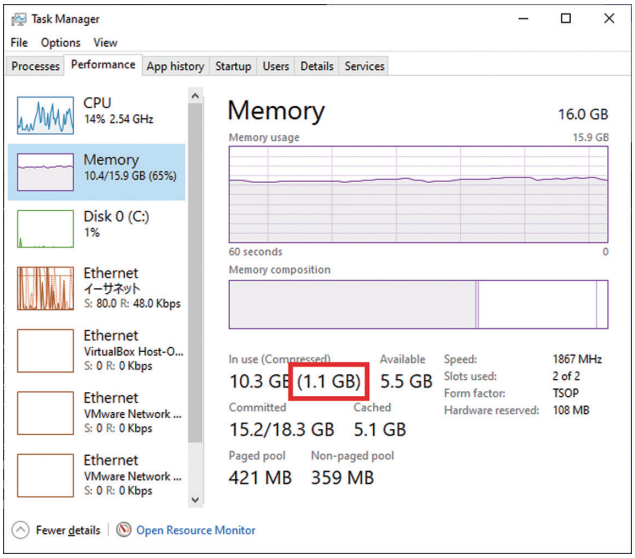

図-8 圧縮されたメモリ容量

しかし、FireEye社のOmar Sardar氏とDimiter Andonov氏がSANS DFIR Summit Austin 2019[*5]及びBlackHat USA 2019[*6]で、Windows 10のメモリ圧縮に対応したVolatility[*7]とRekall[*8]の実装を発表しました。

なお、両実装ともWindows 10 1607から1809までしかサポートしていませんので、Windows 10 1903以降のメモリイメージを解析することはできません。本稿執筆時点で、発表の成果が開発元のソースコードに取り込まれる様子はありませんが、上記したようにVolatilityは新しいバージョンで対応する予定があるようです。

図-9と図-10は、それぞれオリジナルのVolatilityとメモリ圧縮に対応したVolatilityのhashdumpプラグインの実行結果です。hashdumpプラグインはメモリに読み込まれたレジストリハイブから、ユーザのパスワードハッシュを取得します。ユーザパスワードのハッシュがメモリ圧縮に保存されているため、オリジナルのVolatilityでは実行結果に何も出力されていませんが、メモリ圧縮に対応したVolatilityではハッシュが出力されています。

■ Virtual Secure Mode

Windows 10 1511以降のEnterpriseエディションとEducationエディション、及びWindows Server 2016以降では、Virtualization Based Security(VBS)と呼ばれる仮想マシンを使用する分離機構が導入されています。Device GuardやCredential Guardといったセキュリティ機構は、VBSを利用し、Virtual Secure Mode(VSM)と呼ばれる特定機能用の仮想マシンを実行することで実装されています。これらの機能により、ドライバのロード時の検証やアプリケーションの実行制限、また、認証情報を保護することができます。

WimPmem 2.xのような、VSMに対応していないツールを実行すると、図-11のようにブルースクリーンが発生します。

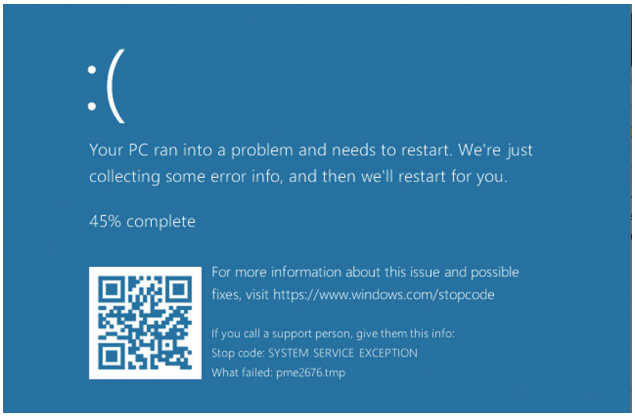

図-11 VSM未対応のメモリイメージ取得ツールを実行するとブルースクリーンが発生する

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

図-9 オリジナルのVolatilityのhashdumpプラグインの実行結果

```
> 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:::
```

図-10 メモリ圧縮対応VolatilityのhashdumpプラグインのIn実行結果

*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)。

ブルースクリーンが起きた場合、デフォルトではホストは自動的に再起動されます。当然ながらメモリの内容はすべて消えてしまうため、使用しているツールがVSMに対応しているか事前に検証しておかなければなりません。なお、表-1のツールの最新バージョンであればブルースクリーンは発生しませんので、古いバージョンを使用している場合はアップデートを検討してください。

■ どのメモリイメージ解析ツールを使うべきか

ここまでに解説したメモリイメージ解析ツールが対応しているデータの種類を表-2にまとめました。この中で開発が継続されているのはVolatility 2のみであるため、基本的にこれを使用することをお勧めします。ただし、Windows 10 1809以前のメモリイメージを解析する場合は、メモリ圧縮対応版Volatilityを使用する方がよいでしょう。

ページングファイルも解析したい場合、Rekallを使う必要がありますが、開発が停止しており、また、WinPmem 3.xとの互換性の問題もあるため積極的に使う場面はあまりないと思われます。残念ながら、あらゆるケースに対応できるツールは現状ありません。しかし、Volatility 3がリリースされれば、この状況は改善されると思われます。

2.3 プロセスを確実にダンプする

ここまで、メモリイメージを取得する際の注意点とその対策を解説してきました。しかし、このような対策を行っても、整合性のとれたメモリイメージの取得は難しいのが現状です。解析対象ホストが仮想マシンであれば、スナップショットを取ることで、その時点の完全な状態のメモリイメージを取得することができます。しかし、ライブシステムのメモリイメージを取得する場合、その最中にも様々なプロセスが動作しているため、メ

表-2 メモリイメージ解析ツール比較

解析ツール	メモリイメージ		ページングファイル	メモリ圧縮	備考
	RAW	AFF4			
Volatility 2	✓				
Rekall	✓	✓(※1)	✓(※2)		開発停止
メモリ圧縮対応版 Volatility	✓			✓	Windows 10 1809まで対応
メモリ圧縮対応版 Rekall	✓	✓(※1)	✓(※2)	✓	Windows 10 1809まで対応

(※1) WinPmem 3.3 rc2 以降で作成した AFF4 ファイルは解析できない。
(※2) WinPmem 3.x で AFF4 ファイルに含めたページングファイルは解析できない。

モリ中のデータ変更やページアウトなどが発生します。このため、メモリイメージを解析しても、プロセスのメモリ内容の整合性がとれていない可能性があります。

図-12は図-1でダンプしたファイル(左側)と図-4でダンプしたファイル(右側)の.textセクションを比較しているところです。図-5で示したように、Rekallでダンプしたファイルの MZヘッダは正しく抽出されましたが、両ファイルの.textセクションを比較するとRekallでダンプしたファイルではデータが0x00になってしまっていることが分かります(図-12の赤い領域)。上記のとおり、このような状態は致し方ないのですが、プロセスを解析する際に障害となる可能性があります。このような場合、ユーザランドよりプロセスを個別にダンプすることで整合性のとれたプロセスダンプを取得することができます(プロセスダンプの際にメモリアクセスが誘発されページアウトしているページがOSによりページインされるため、プロセスの仮想メモリのページをすべて取得することができます)。

プロセスダンプを行うためのツールはいくつかありますが、代表的なツールはWindows Sysinternals ProcDump[*9]です。VolatilityやRekallにも、同じ名前のプラグインがありますが、それらとは異なります。また、VolatilityとRekallのprocdumpはPEフォーマットのファイルを出力しますが、Sysinternals ProcDumpはクラッシュダンプフォーマットのファイルを出力します。

図-13のように実行すると、プロセスIDが4596のプロセスをダンプすることができます。プロセスIDの代わりにプロセス名を指定することもできます。また、プロセスが使用するすべてのメモリをダンプする「-ma」オプションも指定すると良

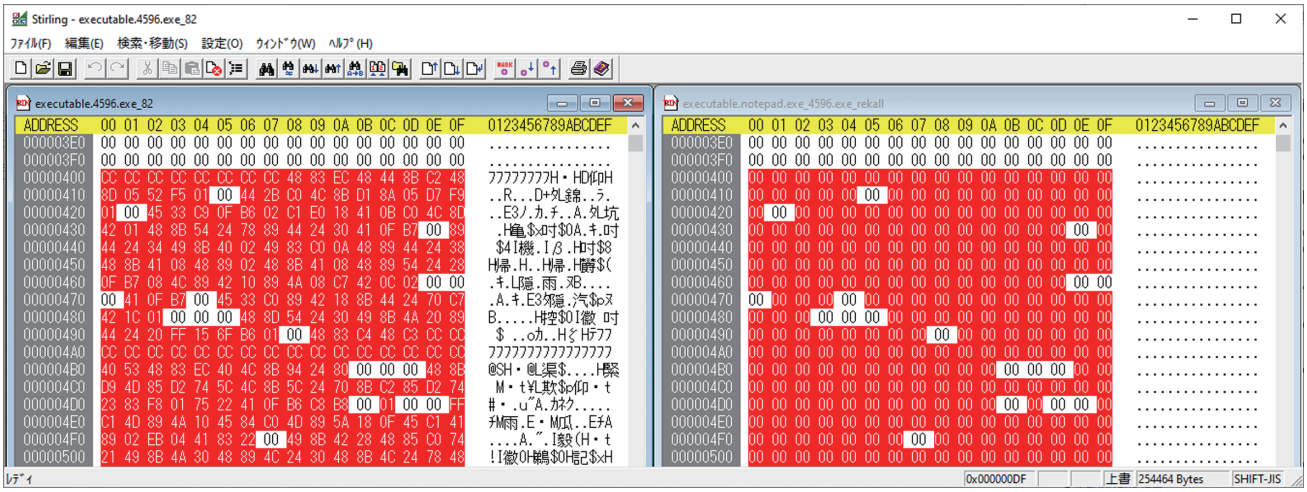

図-12 プロセスダンプの差異

```
>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.
```

図-13 ProcDumpコマンドでプロセスをダンプする

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

いでしょう。生成されたクラッシュダンプはWinDbg[*10][*11]などのツールで読み込むことが可能です。図-14は、図-4の後にProcDumpで生成したクラッシュダンプをWinDbgで読み込んだスクリーンショットです。Rekallのprocdumpで欠けていたコードも含まれていることが分かります（図-14中の赤枠）。このように、ProcDumpを使用すれば、確実にプロセスダンプできます。

2.4 プロセスダンプのスクリプト化とアーティファクト保全の順番

PowerShellなどでスクリプトを作成すれば、容易にすべてのプロセスに対してProcDumpを実行することができます。ただし、すべてのプロセスをダンプすると数十GB以上の容量になります。データ保全先の容量に余裕がある場合は問題ありませんが、容量が小さなUSBメモリや外付けSSDに保全する場合はダンプを取得した後に7-Zip[*12]などで随時、圧縮を行う方が良いでしょう。また、ファイル圧縮時に一時ファイルが作られますので、それらが調査対象のディスクに書き込まれないよう、圧縮コマンドのオプションなどで指定する必要があります。7-Zipの場合、「-w」オプションで作業フォルダを指定できますので、アーティファクトを保全するディスクを指定すると良いでしょう。

ProcDumpを実行する際、SystemやRegistryなどの保護されたプロセスはスキップするようにします（プロセスダンプしようとしてもエラーになります）。また、このようなスクリプトは様々な環境で動作しなければならないため、Windowsにデフォルトでインストールされ ているPowerShellやバッ

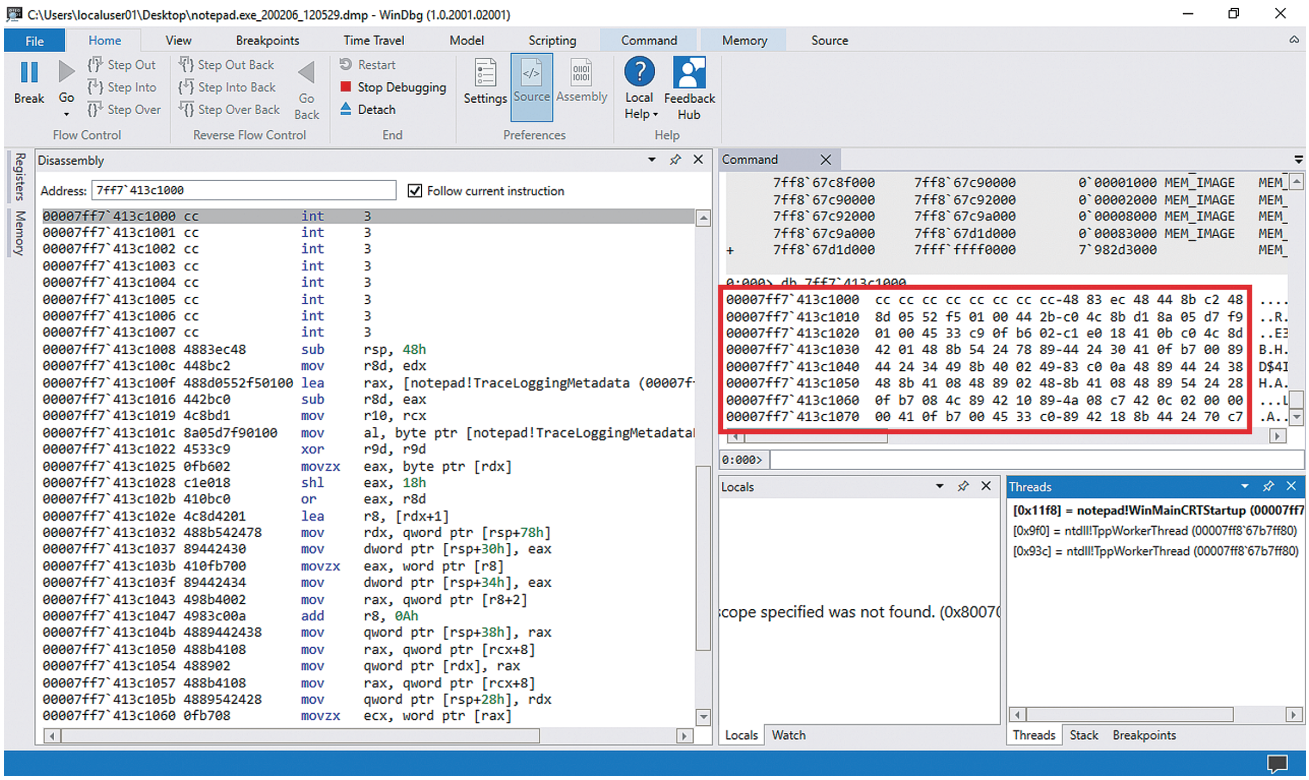

図-14 WinDbgでプロセスダンプを確認

*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/）。

チファイルで作成することが多いと思いますが、注意が必要な点があります。

PowerShlellのプロンプト(powershell.exe)やコマンドプロンプト(cmd.exe)を起動すると、コンソールウィンドウホスト(conhost.exe)も起動します。ProcDumpを実行するPowerShellスクリプトやバッチファイルを起動したプロンプトに対応するconhost.exeをダンプしようとすると、ProcDumpの処理が停止してしまいます。これは、ProcDumpがプロセスダンプを行う際、該当のプロセスをサスペンドさせることが原因です。conhost.exeはコンソールの入出力バッファや表示などを処理するプロセスであるため、このプロセスをサスペンドすることで、それを利用しているProcDumpも停止してしまいます(図-15)。必要であればWinPmemなどで取得したメモリイメージで、conhost.exeを解析することが可能です。

スクリプトを実行すると環境によっては数百回以上、プロセスダンプとファイル圧縮を行うことになります。従って、フォレンジックの観点から考えると、一例として、図-16の順番でアーティファクトの保全を図るのが良いと考えられます。

2.5 まとめ

今回はWindowsでメモリイメージの取得と解析を行う際の注意点を解説しました。また、メモリイメージ取得時の整合性を補完する観点でプロセスダンプにも触れました。メモリフォレンジック関連の記事では、WinPmemなどを使ってメモリイメージを取得することで保全を完了とするものが多いですが、それだけでは十分ではない場合があることを理解いただけたのではないかと思います。ただし、すべてのプロセスのダンプと圧縮には時間がかかりますので、インシデント対応時の状況やポリシーに応じて、実施の可否を検討する必要があります。

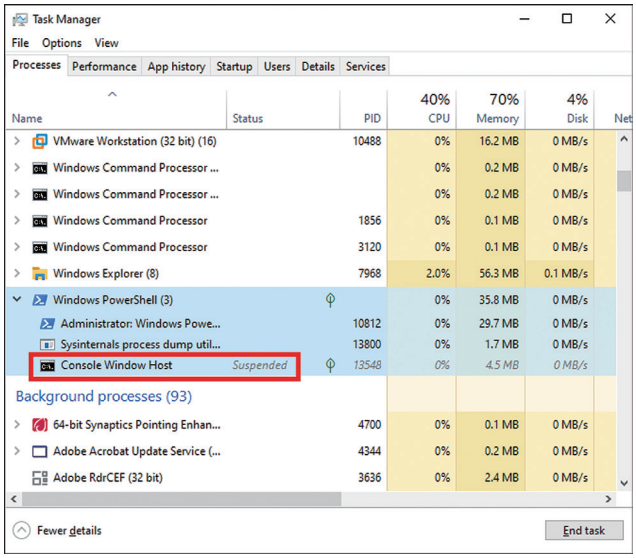

図-15 conhost.exeをダンプするとProcDumpが停止

1. メモリイメージ
2. MFTやPrefetch、イベントログなどのディスクからファイル単位で保全できるアーティファクト
3. プロセスダンプ
4. ディスクイメージ

図-16 アーティファクトを保全する順番の例

執筆者:
小林 稔(こばやし みのる)
IIJセキュリティ本部セキュリティ情報統括室フォレンジックインベスティゲーター。
IIJ-SECTメンバーで主にデジタルフォレンジックを担当し、インシデントレスポンスや社内の技術力向上に努める。
Black HatやFIRST TC、JSAC、セキュリティキャンプなどの国内外のセキュリティ関連イベントで講演やトレーニングを行う。