ForensicMemory Case01

Omar Hsiba Busquets

INTRODUCCIÓN

En este caso trabajamos con un volcado de memoria que se logró recuperar del ordenador de la hermana del usuario después de que el equipo se bloqueara de forma inesperada. Por lo que se recuerda, justo antes del crash apareció una ventana negra ejecutando algo y, en ese mismo momento ella estaba dibujando en el ordenador.

Con tan poca información, la misión del análisis es revisar todo el contenido de la memoria para intentar recuperar los archivos que pudieran ser importantes para ella y averiguar qué actividad había en el sistema cuando ocurrió el fallo. Esto implica buscar procesos que estuvieran activos, identificar cualquier rastro de archivos abiertos y, en general, reconstruir lo que estaba pasando en el equipo en el instante del incidente.

ÍNDICE

HERRAMIENTAS

Durante el análisis utilicé las siguientes herramientas:

Volatility 3

- Para la mayor parte del análisis general.
- Comandos como pslist, psscan, pstree, cmdline, filescan, memmap, dumpfiles, etc.
- Aunque útil, algunos plugins no funcionaban con Windows 7 (como consoles y hashdump).

Volatility 2

- Necesaria para:
    - consoles → recuperar la actividad de la ventana negra (Flag 1).
    - hashdump → extraer hashes del sistema para la contraseña del RAR (Flag 3).
- Se usó porque ciertos plugins no tenían soporte en Volatility 3.

Python

- Para crear un pequeño script que decodificaba la cadena en Base64 del Stage1.
- También para manejar pequeños ajustes y pruebas rápidas.

GIMP

- Fundamental para reconstruir la imagen sin cabecera del proceso mspaint.exe.
- Permite abrir archivos RAW especificando ancho, alto, canales y offset manualmente.

Hashcat

- Lo utilicé para intentar crackear la contraseña asociada al usuario, aunque finalmente no fue necesario porque el propio hash era la contraseña válida

para el RAR.

Wordlists / foryou.txt

- Usadas temporalmente para intentar romper el hash.

En conjunto, estas herramientas permitieron extraer completamente las tres flags y reconstruir lo que estaba pasando en el sistema en el momento del crash.

FLAG 1

Lo primero que hice fue usar el comando *windows.imageInfo* para ver desde qué sistema operativo venía el volcado.

```
Volatility 3 Framework 2.27.0

Variable    Value

Kernel Base 0xf8000261f000
DTB 0x187000
Symbols file:///D:/volatility3/volatility3/symbols/windows/ntkrnlmp.pdb/3844DBB920174967BE7AA4A2C20430FA-2.json.xz
Is64Bit True
IsPAE   False
layer_name  0 WindowsIntel32e
memory_layer    1 FileLayer
KdDebuggerDataBlock 0xf800028100a0
NTBuildLab  7601.17514.amd64fre.win7sp1_rtm.
CSDVersion  1
KdVersionBlock  0xf80002810068
Major/Minor 15.7601
MachineType 34404
KeNumberProcessors  1
SystemTime  2019-12-11 14:38:00+00:00
NtSystemRoot    C:\Windows
NtProductType   NtProductWinNt
NtMajorVersion  6
NtMinorVersion  1
PE MajorOperatingSystemVersion  6
PE MinorOperatingSystemVersion  1
PE Machine  34404
PE TimeDateStamp    Sat Nov 20 09:30:02 2010
```

Una vez tuve eso claro, empecé a revisar procesos con pslist, psscan y pstree para ver si había algo raro.

```
python vol.py -f MemoryDump_Lab1.raw windows.pslist
python vol.py -f MemoryDump_Lab1.raw windows.psscan
python vol.py -f MemoryDump_Lab1.raw windows.pstree
```

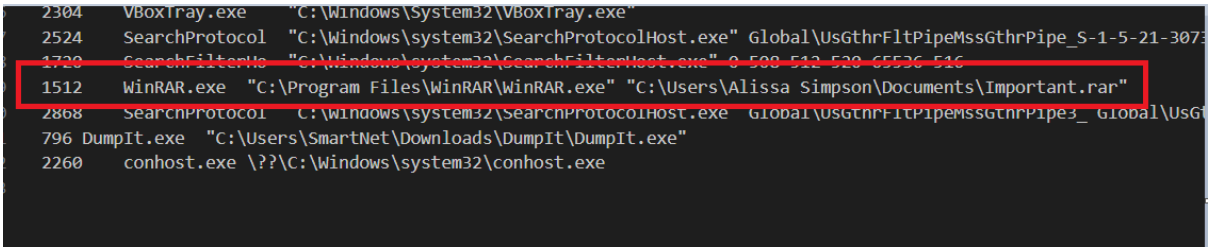
En pslist ya me encontré con un par de cosas interesantes:

- Una **cmd.exe** que colgaba directamente de explorer.exe, lo cual encajaba totalmente con lo que decía el enunciado de que apareció una “ventana negra” antes de que el ordenador muriera,

- Y también estaba **mspaint.exe**, que tiene sentido porque la niña estaba dibujando justo antes del crash.

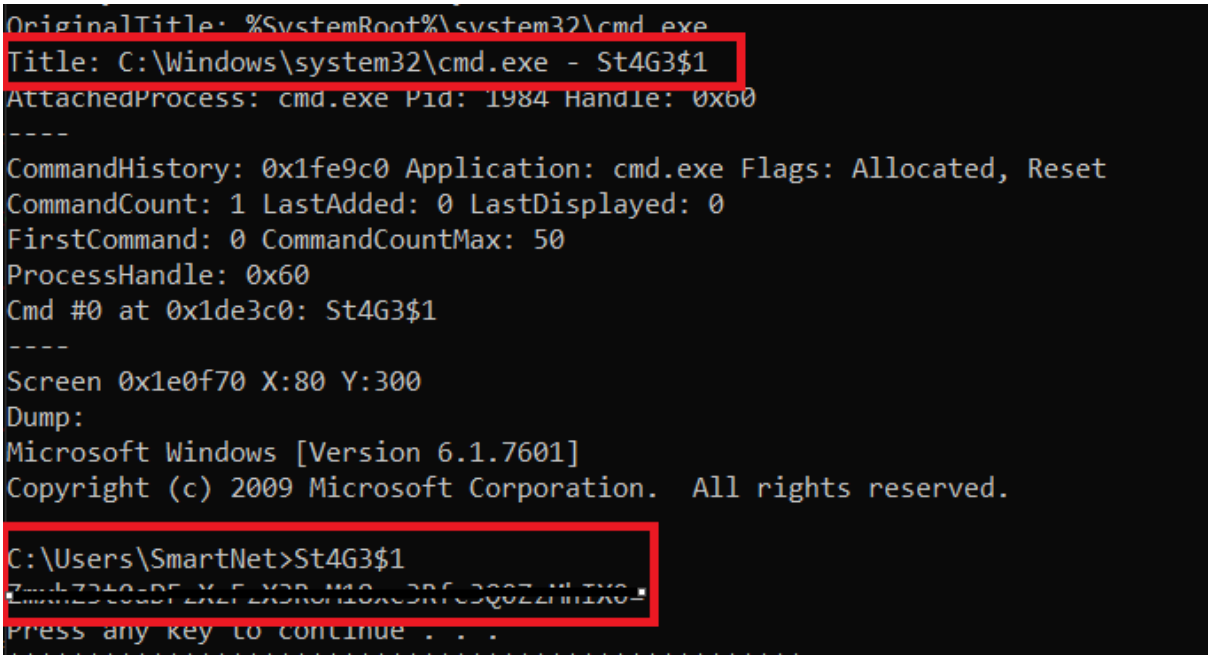
```
2368    484 svchost.exe 0xfa80022199e0  9   365 0   False   2019-12-11 14:32:51.000000 UTC  N/A Disabled
1984    604 cmd.exe 0xfa8002222780  1   21  1   False   2019-12-11 14:34:54.000000 UTC  N/A Disabled
2424    604 mspaint.exe 0xfa80022bab30  6   128 1   False   2019-12-11 14:35:14.000000 UTC  N/A Disabled
2660    484 svchost.exe 0xfa8000eac770  6   100 0   False   2019-12-11 14:35:14.000000 UTC  N/A Disabled
2760    2680    csrss.exe   0xfa8001e68060  7   172 2   False   2019-12-11 14:37:05.000000 UTC  N/A Disabled
2808    2680    winlogon.exe    0xfa8000ecbb30  4   119 2   False   2019-12-11 14:37:05.000000 UTC  N/A Disabled
```

Aparte de esos dos, no vi nada demasiado sospechoso. Pero al pasar a psscan, varios procesos aparecían repetidos y uno que me llamó mucho la atención fue WinRAR.exe. Así que probé cmdline para ver si había lanzado algún comando extraño. No vi nada raro desde cmd.exe ni desde WinRAR, pero sí descubrí que WinRAR había abierto un archivo llamado Important.rar, así que ya me lo guardé como posible pista.

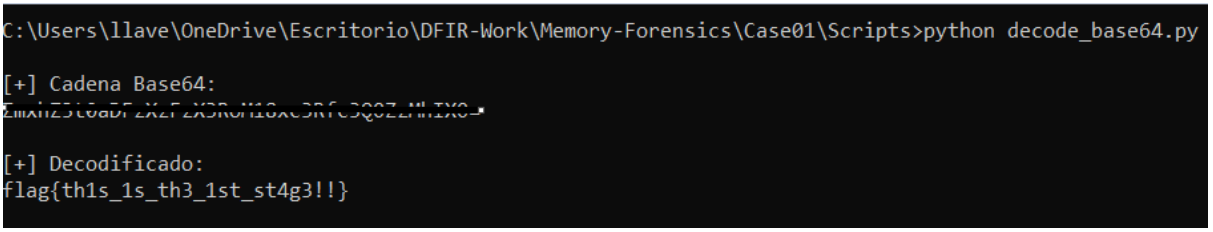


Mi idea era usar windows.consoles con Volatility 3 para ver exactamente qué se había escrito en la terminal, pero resulta que no funciona bien con Windows 7, que era el sistema del volcado. Como no avanzaba, instalé Volatility 2 y probé allí el plugin consoles.

Y ahí sí: en la salida apareció una línea que ponía Stage1 junto a un texto en Base64.



Para ver qué era, hice un script rápido en Python para decodificarlo, y al hacerlo me salió claramente la flag:



FLAG 2

Para sacar el segundo flag empecé mirando el proceso mspaint.exe, porque según el enunciado la niña estaba dibujando justo antes de que el ordenador se colgara. Lo primero que probé fue usar dumpfiles sobre ese proceso para intentar recuperar directamente la imagen:

```
>python vol.py -f MemoryDump_Lab1.raw windows.dumpfiles --pid 2424 --dump
```

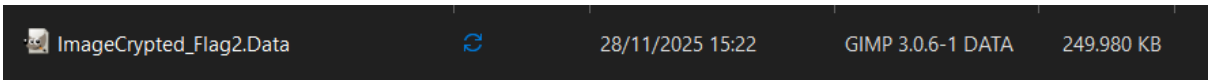
El problema es que dumpfiles generaba muchísimos archivos pequeños, cada uno perteneciente a distintas partes del proceso, y no había manera rápida de saber cuál contenía realmente la imagen sin guardar. Para evitar perder tiempo revisando uno a uno, preferí usar otra técnica: volcar toda la memoria del proceso en un único archivo bruto e ir reconstruyendo la imagen manualmente.

Así que opté por usar memmap con --dump

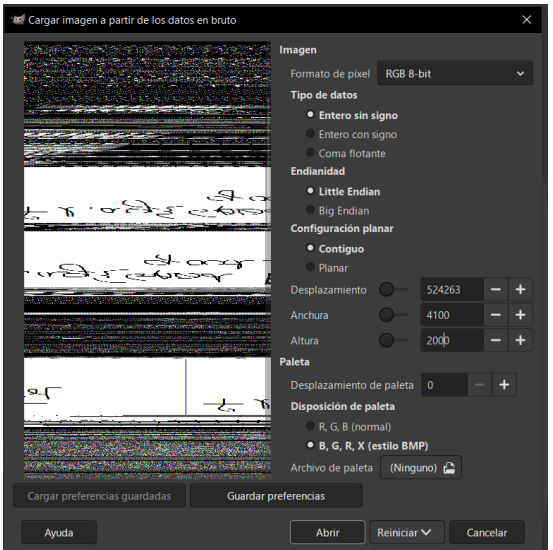
```
>python vol.py -f MemoryDump_Lab1.raw windows.memmap --pid 2424 --dump
```

Este comando generó varios archivos, pero uno de ellos era enorme (unos 250 MB). Ese claramente tenía toda la memoria del Paint, incluyendo la imagen sin guardar.

El archivo venía con extensión .dat, pero para poder abrirlo como imagen RAW en GIMP tuve que renombrarlo a .data.



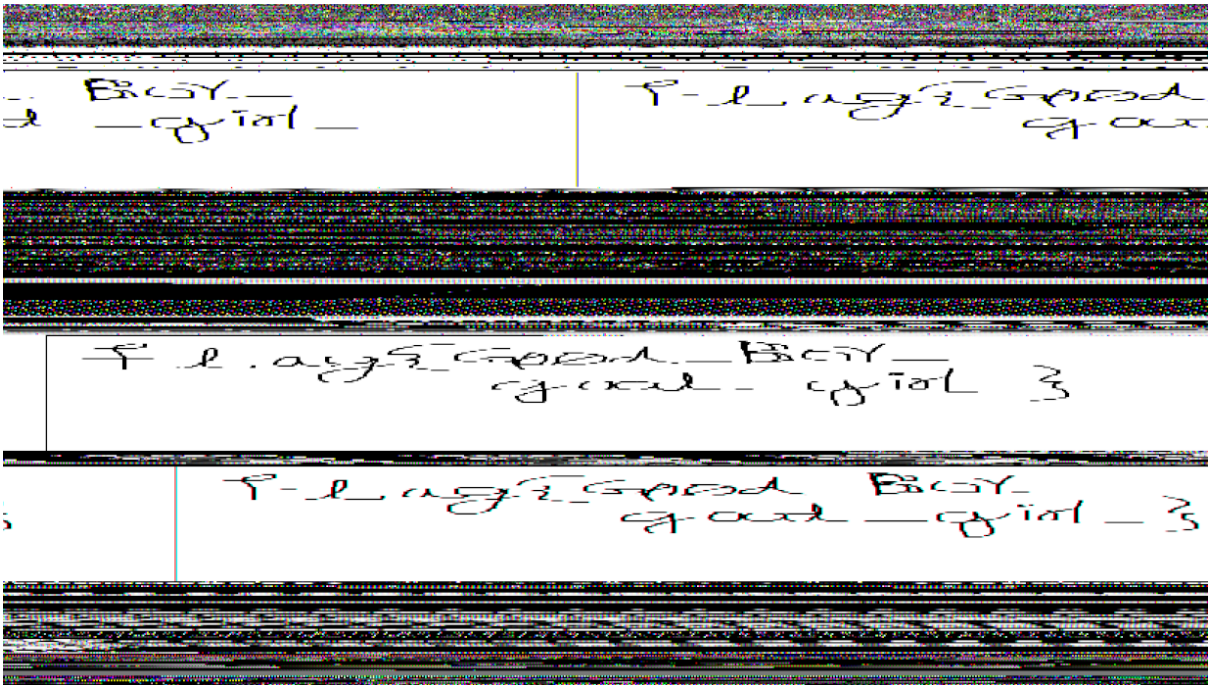
Después lo abrí en GIMP.
Al ser un volcado en bruto, no tenía cabecera ni metadatos, así que al principio no se veía nada. Tuve que ir probando resoluciones manualmente. Fui variando anchos y alturas hasta que finalmente encontré una combinación que empezaba a dar forma a algo: 4100 px de ancho y unos 2000 px de altura.



A partir de ahí ajusté también el offset, porque si no la imagen aparece desplazada o corrupta. Tras varios intentos, las zonas blancas y trazos del dibujo empezaron a encajar y finalmente pude distinguir texto dentro de la imagen.

Una vez la tenía, tuve que rotar la imagen verticalmente para que fuese legible

Y así apareció claramente el segundo flag:



Con esto conseguí la Flag 2.

FLAG 3

La Flag 3 fue, sin duda, la que más tiempo me llevó. El problema no era la dificultad en sí, sino que estaba usando mal el comando clave desde el principio. El comando que realmente hacía falta era filescan, pero tardé en darme cuenta.

Al principio intenté recuperar los archivos relacionados con WinRAR.exe usando dumpfiles directamente:

```
>python vol.py -f MemoryDump_Lab1.raw windows.dumpfiles --pid 1512 --dump
```

Esto me devolvió un montón de archivos, pero no era lo que esperaba. Al cambiar la extensión de algunos a .rar, las cabeceras estaban totalmente corruptas. Además, dumpfiles del proceso mezcla archivos de caché, temporales, fragmentos de memoria… básicamente un caos.

Eso sí, esta prueba me dejó claro algo importante:
Si quería obtener el RAR original, necesitaba la dirección física exacta del archivo dentro de la memoria.
Y ahí es donde entra filescan.

Ejecuté:

```
python vol.py -f MemoryDump_Lab1.raw windows.filescan
```

Aquí sí apareció lo que buscaba: varias entradas del archivo Important.rar con sus direcciones físicas. En concreto, me salieron tres direcciones asociadas al mismo archivo dentro de WinRAR.

```
0x3fa3d9b0  \ProgramData\Microsoft\Search\Data\Applications\Wi
0x3fa3dd10  \Windows\System32\en-US\consent.exe.mui
0x3fa3ebc0  \Users\Alissa Simpson\Documents\Important.rar
0x3fa3ef20  \Windows\System32\eisiau.dll
```

Con esas direcciones, ya pude usar el comando adecuado:

```
D:\volatility3>python vol.py -f "...\Case01\Evidence\Dangerous_File" -o "...\WINRAR MEMORY FORENSICS\
Case01\Evidence\Dangerous_File" windows.dumpfiles --physaddr 0x3fa3ebc0
Volatility 3 Framework 2.27.0
Progress:  100.00               PDB scanning finished
Cache   FileObject      FileName        Result

DataSectionObject       0x3fa3ebc0      Important.rar   file.0x3fa3ebc0.0xfa8001034450.DataSectionObject.Important.rar.dat
```

Esto sí generó un archivo mucho más limpio. Nada que ver con el desastre de

dumpfiles aplicado al PID entero. Le cambié la extensión a .rar y, esta vez, el archivo quedó perfecto.

Pero al intentar abrirlo… pedía contraseña.

Aquí empezó el segundo problema, no tenía claro cuál podría ser. Probé varias contraseñas, incluso frases relacionadas con la Flag 2, pero nada.

Buscando información encontré que podía extraer contraseñas usando hashdump, pero solamente estaba disponible en Volatility 2. Así que abrí Volatility 2 y ejecuté:

```
D:\volatility2>vol2.exe -f "[redacted]" --profile=Win7SP1x64 hashdump
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SmartNet:1001:aad3b435b51404eeaad3b435b51404ee:4943abb39473a6f32c11301f4987e7e0:::
HomeGroupUser$:1002:aad3b435b51404eeaad3b435b51404ee:f0fc3d257814e08fea06e63c5762ebd5:::
Alissa Simpson:1003:aad3b435b51404eeaad3b435b51404ee:f4ff64c8baac57d22f22edc681055ba6:::
```

Esto me mostró los hashes de las contraseñas del sistema. Me fijé en la del perfil de Alissa Simpson, que era claramente la usuaria principal.

Intenté crackear ese hash con hashcat. Me descargué un wordlist estilo “rockyou.txt” y probé:

```
D:\hashcat>hashcat.exe -m 1000 -a 0" [redacted]" Rockyou.txt
hashcat (v6.2.6) starting
```

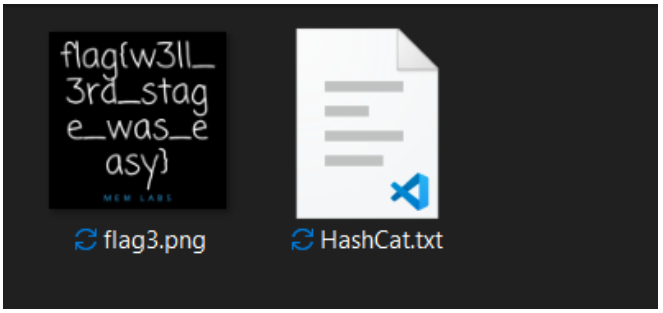
Crackeó una contraseña… pero no funcionaba para el RAR.

Probé más herramientas, distintos modos, reglas… pero ninguna de las contraseñas que me devolvieron funcionaba.

Después de varios intentos fallidos usando diccionarios y distintas herramientas para romper el hash, decidí parar un momento y buscar información sobre por qué ninguna de las contraseñas generadas funcionaba.

Revisando foros y documentación, encontré que en algunos laboratorios forenses los hashes que devuelve *hashdump* no siempre están pensados para ser crackeados, sino que a veces se usan directamente como contraseña dentro del propio ejercicio.

Con esa idea en mente, probé a introducir el hash tal cual aparecía en la salida de *hashdump*, sin modificaciones. Como pensaba, esa resultó ser la contraseña real del archivo RAR y dentro de él estaba la Flag 3.



CONCLUSIÓN

En general, este caso me sirvió para practicar prácticamente todas las técnicas básicas de análisis de memoria. Aunque al principio parecía un caso sencillo, al final fue necesario combinar varios métodos porque no todo funcionaba a la primera. Para encontrar las flags tuve que ir saltando entre Volatility 3 y Volatility 2, pelearme con plugins incompatibles, abrir imágenes RAW a mano y hasta probar contraseñas sin mucha esperanza.

La Flag 1 apareció gracias al plugin de consoles en Volatility 2, que mostró un texto en Base64 que pude decodificar fácilmente. La Flag 2 la encontré en la memoria del proceso de Paint reconstruyendo una imagen sin guardar a partir de un volcado bruto. Y la Flag 3 fue la más complicada: necesitó filescan, dumpfiles con dirección física exacta, y finalmente usar el propio hash extraído del sistema como contraseña del RAR.

Al final, las tres flags se pudieron recuperar completamente desde el volcado de memoria, confirmando que la mayoría de la información del sistema seguía disponible pese al crash del ordenador.