Windows Malware: Traces in the Host

Gonzalo Ruiz De Angeli, Juan Ignacio Alberdi, Bruno Constanzo, Hugo Curti, Ana Haydée Di Iorio

***Abstract*—In the present world of information and interconnection, malware is a latent threat. Just speaking of "ill-intended software" gives a too broad definition: malware has evolved and taken different forms through the years. It is necessary to know and understand the traces that remain in a computer system after an infection. For years the information security community has focused on live analysis and response against these threats, so there is a huge opportunity to adapt and make post-mortem, host-based signatures. In this work, some features that may work as malware signatures for digital forensics experts are proposed.**

***Index Terms*— Host-based signatures, malware, memory forensics, DFIR, Windows registry, DLLs, Portable Executable.**

I. Introducción

En la actualidad el mundo se encuentra fuertemente conectado a través de Internet: las comunicaciones son instantáneas a escala global, y prácticamente toda la información se encuentra disponible a través de la interacción de múltiples sistemas de información distribuidos. El riesgo constante para estos sistemas, y por extensión, para la vida moderna, consiste en un ataque de *malware* que sea capaz de romper el frágil equilibrio en que coexisten y funcionan. No se plantea esto como una situación hipotética, es algo que ya ha sucedido y que aún continúa sucediendo[1].

Durante años los expertos en seguridad informática, antivirus e ingeniería inversa[2] han investigado *software* malicioso y desarrollado técnicas de análisis para detectar su presencia[4, 5], analizar su funcionamiento[6, 7] y descubrir información relacionada con el *malware*. Estas técnicas están enfocadas principalmente en el análisis en vivo de las amenazas, y cómo contrarrestar su accionar, por lo que en ocasiones no son directamente aplicables a una investigación *post-mortem* en un entorno forense. Es necesario entonces hacer el trabajo de trasladar este conocimiento a la Informática Forense, de manera que pueda ser aplicable y útil en la realización de pericias informáticas.

En el presente trabajo se presenta un marco teórico compuesto por conceptos de análisis de memoria, procesos en memoria, el formato de archivo de binarios ejecutables utilizado por Windows tanto para programas como para librerías dinámicas compartidas (DLLs), y conceptos del Registro de Windows. De cada tema se plantean posibles indicadores *host-based* (es decir, propios del equipo analizado) que pueden utilizarse en una investigación para determinar una infección por *malware* dentro de un equipo. Finalmente, se presenta un método, aplicado en un ejemplo de cómo analizar estos indicadores en un caso real y se cierra con las conclusiones y trabajo futuro.

Este trabajo fue enviado a revisión el día 15 de junio de 2017.

Gonzalo Ruiz De Angeli, Juan Ignacio Alberdi, Bruno Constanzo, Hugo Curti y Ana Haydée Di Iorio son investigadores del Grupo de Investigación en Sistemas Operativos e Informática Forense de la Facultad de Ingeniería de Universidad FASTA, Mar del Plata, Buenos Aires, Argentina (email: {ruizgon, ignacio_alberdi, bconstanzo, hcurti, diana}@ufasta.edu.ar).

.

[1] Se pueden mencionar Blaster y Slammer como dos *worms* históricos[1], o *WannaCry* como un ejemplo más reciente[2, 3].

[2] Entendiendo *reverse engineering* en el contexto de análisis de *malware* y sus mecanismos.

II. Análisis Forense de Memoria Principal

El análisis forense de memoria principal es base para la confección de indicadores en el equipo (*host-based*) que permitan detectar la presencia de *malware*, aunque no es el único análisis que se puede realizar para obtener métricas relevantes en este sentido.

Muchos indicadores pueden encontrarse en memoria, algunos de manera exclusiva y otros replicado desde otras fuentes, cargados por el sistema operativo en ella para agilizar su acceso. Por esta razón debería obtenerse un volcado de memoria del equipo que se va a analizar cuando aún se encuentra encendido luego de un ataque por *malware*, o eventualmente generar un volcado de memoria en base a la ejecución de la imagen de disco obtenida en la etapa de adquisición por medio de una máquina virtual[28].

También debe tenerse en cuenta que el *malware* moderno trata de interactuar lo menos posible con métodos de persistencia clásicos, por ejemplo, se auto-elimina del disco una vez que se ha ejecutado, o utiliza funciones de cifrado, *packing*, u otras, que le permiten pasar desapercibido en los medios de almacenamiento. Pese a todos esos intentos de ocultamiento, el *malware*, para ejecutarse tiene que ser un proceso cargado en memoria, debe descifrar su código, y como tal se puede obtener sus atributos de proceso de los que pueden derivarse indicadores útiles para su detección.

Otro aspecto de suma importancia es que de un volcado de memoria es posible extraer claves de cifrado, ya sean del sistema o utilizadas por el *malware*[3], archivos cargados en memoria por los procesos, o listados de conexiones activas entre otros datos de interés que pueden ser complementarios a una investigación.

El *malware* en general tiene como objetivo tomar control del equipo, y para eso busca, en primer lugar, ejecutarse con el mayor grado de privilegios posibles, y, en segundo lugar, mantenerse oculto del usuario. Por esta razón, ante la hipótesis

[3] Ver *WanaKiwi*[8, 9], una herramienta para recuperar información de un ataque de *WannaCry* que escanea la memoria y permite recuperar las claves de cifrado de los archivos afectados.

de una infección por *malware*, el análisis de memoria es una herramienta fundamental ya que permite al analista informático partir de una búsqueda de procesos ocultos, y analizar las funciones utilizadas por estos, los archivos que han cargado en memoria, las conexiones que establecieron, y las entradas de registro que se podrían haber modificado. De todos estos artefactos también es posible extraer indicadores para sostener o rechazar esta hipótesis.

Las principales estructuras de datos del sistema operativo[26, 27] Windows[4] presentes en memoria, que permiten llevar a cabo este tipo de análisis son:

- `_EPROCESS` e `_ETHREAD`, que representan procesos e hilos respectivamente.
- `_LDR_DATA_TABLE_ENTRY`, que almacena información de una DLL.
- `_FILE_OBJECT`, un archivo cargado en memoria.
- `_CMHIVE`, una entrada de Registro.
- `_TCP_CONNECTION`, `_UDP_CONNECTION` y, `_SOCKET` contienen información de conexiones abiertas.

El análisis de estas estructuras se verá en detalle en próximas secciones.

III. Formato Portable Executable (PE)

Portable Executable es el formato de archivos binarios ejecutables utilizado por Windows (por ejemplo, los archivos .exe o .dll), que se basa en el formato COFF de Unix[10, 11]. La implementación de PE permite que se pueda compartir código entre programas cargados en memoria, y así exportar funcionalidad común desde librerías del sistema, o desarrolladas por terceros. Por esta razón, el análisis de este tipo de archivos resulta útil: por un lado, porque el formato mismo brinda información sobre el sistema operativo[12, 13], y por otro lado, en el caso de las DLLs es posible realizar un análisis de las funciones exportadas que quedarán disponibles para que otros procesos utilicen.

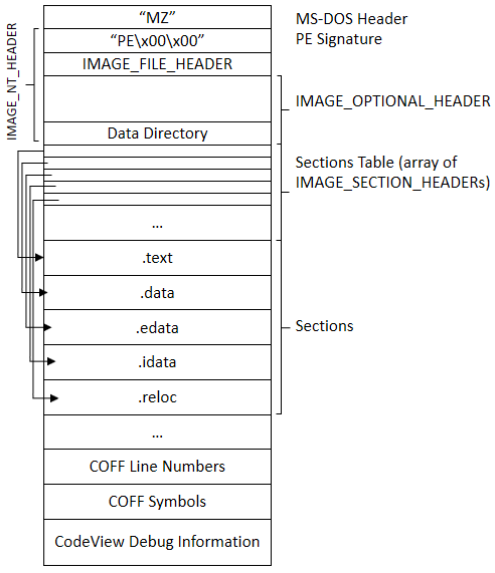

Fig. 1 – Estructura de un archivo PE.

[4] El objeto de este trabajo es el sistema operativo Windows 7, sin embargo el contenido desarrollado puede ser aplicable a otras versiones.

En la Fig. 1 puede verse la estructura de estos archivos, donde se pueden analizar sus campos, tanto en ejecutables como en librerías compartidas, para extraer información (por ejemplo, las funciones exportadas).

En la estructura del encabezado MS-DOS Header lo más importante es el puntero a `_IMAGE_NT_HEADER`, donde se aloja la firma `"PE\x00\x00"`, que confirma que se trata de un archivo *Portable Executable*. El campo más interesante en `_IMAGE_NT_HEADER` es el *Data Directory*, que contiene un arreglo de 16 elementos, del cual solo es relevante el primero: *Export Directory*, que contiene la dirección virtual relativa (*Relative Virtual Address*, o RVA) del `_IMAGE_EXPORT_DIRECTORY`. En esta última estructura hay tres elementos de interés con RVA:

- Los nombres de las funciones: *AddressOfNames*.
- Las direcciones de las funciones: *AddressOfFunctions*.
- Un índice para buscar obtener la dirección de la función propiamente dicha: *AddressOfNamesOrdinals*. Éste es un arreglo paralelo a *AddressOfNames* y provee el índice para buscar la dirección de la función en *AddressOfFunctions* con su nombre correspondiente en *AddressOfNames*.

Con esta información es posible construir un indicador de ***las funciones que potencialmente ejecutaría un proceso***. A partir de aquí, se puede profundizar el análisis en la búsqueda de *hooks* a las llamadas del sistema. Para ello es necesario en cierto punto realizar un análisis de más bajo nivel, sobre el código desensamblado. Ese tipo de análisis escapa a los alcances de este trabajo.

IV. Procesos en Memoria

En Windows los procesos activos se organizan dentro de una lista doblemente enlazada en memoria. A partir de la cabecera de la lista, se apunta a una estructura `_EPROCESS` (el primer proceso de la lista), y ésta tiene dos punteros: al siguiente proceso (`Flink`) y al anterior (`Blink`). En el caso del primer y último elemento de la lista, los punteros `Flink` y `Blink` apuntan a la cabecera de la lista, respectivamente. Estos punteros almacenan direcciones virtuales:

```
_LIST_ENTRY
{
  struct _LIST_ENTRY* Flink;
  struct _LIST_ENTRY* Blink;
}
```

Dado que el *malware* suele buscar ocultarse del usuario[14-16], es muy común que se des-referencie de la lista de procesos activos[17] (y garantice su ejecución contaminando funciones de sistema[18, 19]). La Fig. 2 ilustra como un *malware* en memoria se ocultaría de la lista de procesos activos.

Combinando algunas técnicas de búsqueda y análisis de estructuras, como el recorrido de la lista de procesos activos y su comparación con las estructuras `_EPROCESS` halladas a través de la técnica de *pool-tag scanning*[20, 21], es posible confeccionar un indicador de ***procesos ocultos en memoria principal***.

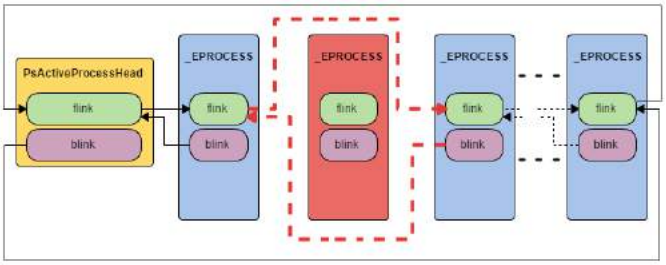

Fig. 2 – Proceso oculto en la lista de procesos activos.

V. Dynamic-Link Libraries (DLLs)

Las DLLs permiten que un proceso pueda hacer uso de funcionalidad común sin la necesidad de reescribir código. La utilización de algunas DLLs puede dar indicios de algún comportamiento particular por parte del proceso que las referencia, por ejemplo, aquellas funciones asociadas a conexiones, la creación de archivos, edición del Registro (*RegOpenKey*[6]), etc.

En memoria, las DLLs forman tres listas doblemente enlazadas, en las que todas las librerías cargadas se encuentran presentes, pero cada lista lleva un orden distinto: *InLoadOrderModuleList*, *InMemoryOrderLinks* e *InInitializationOrderLinks*. Estas listas son similares a la lista de procesos activos, ya que todas utilizan la estructura `_LIST_ENTRY`.

Si se realiza un análisis de las DLLs utilizadas por un proceso sospechoso, se puede saber cuáles son las funciones que podría llegar a ejecutar. A continuación, se muestra una estructura `_EPROCESS` resumida:

```
typedef struct _EPROCESS
{
  struct _KPROCESS Pcb;
  union _LARGE_INTEGER CreateTime;
  union _LARGE_INTEGER CreateTime;
  VOID* UniqueProcessId;
  struct _LIST_ENTRY ActiveProcessLinks;
  UINT8 ImageFileName[15];
  struct _PEB* Peb;
}
```

La estructura `_PEB` dentro de `_EPROCESS` tiene un puntero a `_PEB_LDR_DATA`[20], a través de la cual, se puede acceder a la lista de DLLs cargadas por el proceso, recorriendo la lista de estructuras `_LDR_DATA_TABLE_ENTRY`, como se muestra en la Fig. 3.

De esta manera se puede construir un indicador de ***DLLs cargadas por proceso***, que permitirá luego llevar a cabo un análisis más profundo sobre las funciones que un software podría requerir de las DLLs del sistema, o aquellas que carga por su cuenta.

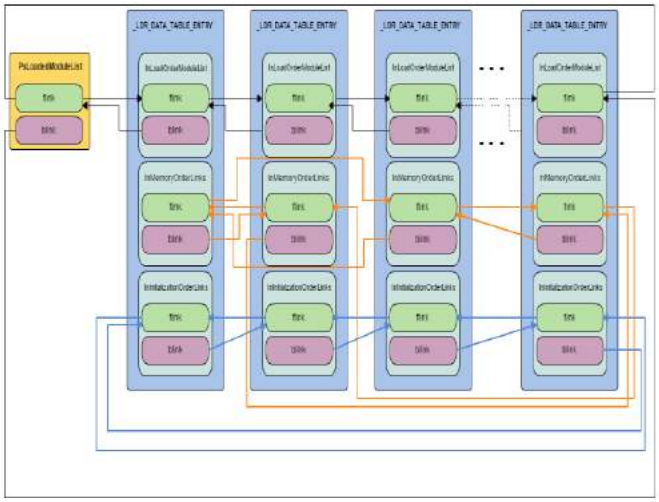

Fig. 3 – Listas doblemente enlazadas de los módulos cargados por un proceso.

VI. Registro de Windows

El Registro de Windows es el repositorio de configuración del sistema operativo. El hallazgo de determinadas entradas de registro puede dar indicios sobre la presencia de *malware* en el equipo.

Entender la estructura del Registro de Windows permite comprender cuál es el alcance de cada entrada en la configuración, es decir, si se trata de una configuración global, si afecta a todos los usuarios, o a un usuario en particular. A continuación, se da una breve explicación sobre cómo es la organización jerárquica de esta base de datos de configuraciones[22]:

- HKEY_LOCAL_MACHINE, abreviado como **HKLM**, almacena configuraciones específicas del equipo local. Algunas de sus sub-ramas más importantes son:
  - HKLM\SAM almacena información de todas las cuentas de usuario utilizadas en el equipo.
  - HKLM\SECURITY está vinculada a la base de datos de seguridad del dominio en el que ha iniciado sesión el usuario.
  - HKLM\SYSTEM contiene información sobre el programa de instalación del sistema de Windows, datos para el generador seguro de números aleatorios (RNG), la lista de los dispositivos montados actualmente que contienen un sistema de archivos, entre otros.
  - HKLM\SOFTWARE contiene información vinculada a todo el software instalado en el sistema actual.
- HKEY_USERS, abreviado como **HKU**, contiene sub-claves correspondientes a las claves HKEY_CURRENT_USER de cada perfil de usuario cargado activamente en el equipo.
- HKEY_CURRENT USER, abreviado como **HKCU**, almacena configuraciones específicas del usuario con sesión iniciada en ese momento.
- HKEY_CLASSES_ROOT, abreviado como **HKCR**, contiene información sobre aplicaciones registradas y el tratamiento de los distintos formatos de archivo.

- HKEY_CURRENT_CONFIG contiene información sobre el perfil de hardware actual del sistema informático local. La información en HKEY_CURRENT_CONFIG describe sólo las diferencias entre la configuración de hardware actual y la configuración estándar.

El análisis de las entradas de registro permite obtener información sobre la configuración del equipo. Por ejemplo:

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run es una entrada de registro que puede ser modificada por un proceso para lograr la autoejecución del mismo al inicio del sistema.
- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run es otra entrada que puede modificarse con el mismo propósito, pero desde el entorno del usuario.
- HKU\.DEFAULT\SOFTWARE\Microsoft\Windows\CurrentVersion\Run es otra entrada que abarca a los programas que por defecto deben cargarse en un entorno de usuario nuevo.

En otras ramas del registro también es posible encontrar claves de cifrado, la ubicación de archivos infectados, o un *malware* ubicar los últimos programas utilizados en el equipo con el objetivo de infectarlos. Todo esto es sumamente rico para la confección de indicadores de *malware*.

Además del análisis de los archivos de registro en el disco, también es posible analizar la memoria y en ella encontrar entradas de registro relacionadas a los hilos propietarios. La estructura `_CMHIVE` representa un archivo de entrada de registro en disco, y posee un atributo `_KTHREAD` que es un puntero a su hilo propietario. A continuación, se resumen la estructura y el atributo de interés:

```
typedef struct _CMHIVE
{
  struct _HHIVE Hive;
  struct _LIST_ENTRY HiveList;
  struct _UNICODE_STRING FileFullPath;
  struct _UNICODE_STRING FileUserName;
  struct _UNICODE_STRING HiveRootPath;
  struct _KTHREAD* CreatorOwner;
}
```

La estructura `_HHIVE` es un encabezado que brinda información sobre el contenido de la entrada de registro. Como se muestra en la Fig. 4, cada estructura `_CMHIVE` en memoria se relaciona con el resto de las entradas utilizadas por el hilo a través de una estructura `_LIST_ENTRY`, llamada *HiveList*, formando una lista doblemente enlazada. De esta manera, es posible encontrar ***entradas de registro cargadas por un determinado proceso***.

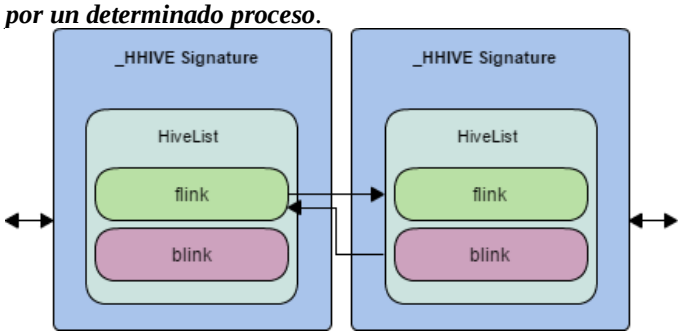

Fig. 4 – Elementos de la lista enlazada de entradas de registro.

Como las estructuras `_CMHIVE` se encuentran dentro de *pools* de memoria con la etiqueta `"CM10"`, es posible combinar la técnica de *pool-tag scanning* con el recorrido de la lista doblemente enlazada a través del atributo *HiveList* para obtener todas las entradas de registro en memoria y construir un mapa de configuración del sistema, del usuario actual (*Current User*) y de un posible software malintencionado en el equipo.

VII. Caso Ejemplo: WannaCry

A continuación se muestra un análisis de un equipo infectado con el *malware WannaCry*. En este ejemplo, se parte de un volcado de memoria que se obtuvo del equipo a poco de comenzar la ejecución del software malicioso. Para las tareas de análisis de memoria en este ejemplo se utilizará Volatility Framework[23, 24].

En primer lugar, se empieza por una búsqueda de procesos ocultos con el *plugin* `psxview`, que utiliza múltiples técnicas de búsqueda de procesos.

Comando: `vol.py --plugins=./contrib/plugins/ --profile=Win7SP1x86 -f M:\memdump.mem psxview`

Aclaración: por brevedad, y para mantener una cierta claridad en el ejemplo, se han recortado algunas filas y columnas del resultado obtenido.

Salida:

```
Name             PID pslist psscan ... ExitTime
-------------- ---- ------ ------ ... --------
mcshield.exe    3168 True   False  ...
svchost.exe     1180 True   False  ...
MsDtsSrvr.exe   1556 True   False  ...
svchost.exe     2228 True   False  ...
GoogleCrashHan 5200 True   False  ...
svchost.exe     1564 True   False  ...
chrome.exe      7680 True   False  ...
svchost.exe     2268 True   False  ...
...
tasksche.exe    1392 True   False  ...
...
tasksche.exe    2808 True   False  ...
svchost.exe     1032 True   False  ...
@WanaDecryptor 7676 True   False  ...
...
System             4 True   False  ...
csrss.exe       6916 True   False  ...
smss.exe         324 True   False  ...
csrss.exe        536 True   False  ...
RtHDVBg.exe     1432 True    False   ... 2017-05-16
14:48:50 UTC+0000
RtHDVBg.exe     4620 True    False   ... 2017-05-16
14:25:44 UTC+0000
N                  0 True   False  ...
dllhost.exe      972 True   False  ...
taskdl.exe      5752 True    False   ... 2017-05-16
14:57:50 UTC+0000
```

Como podemos observar, para los procesos más relevantes involucrados con el mencionado *malware*, la salida de Volatility no indica que estén intentado ocultarse, aunque si utilizan nombres de procesos poco llamativos. Esto es coherente con el *modus operandi* de los *ransomware*, que particularmente, luego de haber cifrado los archivos, buscan llamar la atención del usuario en lugar de ocultarse.

Se continúa el análisis buscando identificar las DLLs cargadas en memoria. Para esto, se obtiene el listado utilizando el *plugin* `dlllist`[5]:

Comando: `vol.py --plugins=./contrib/plugins/ --profile=Win7SP1x86 -f M:\memdump.mem dlllist`

Aclaración: sólo hemos dejado el campo "Path" de la salida para cada proceso.

Salida:

```
tasksche.exe pid:   2808
Command line :
 C:\ProgramData\zmlrzffkezkivj575\tasksche.exe
Service Pack 1

Path
----
C:\ProgramData\zmlrzffkezkivj575\tasksche.exe
C:\Windows\SYSTEM32\ntdll.dll
C:\Windows\system32\kernel32.dll
C:\Windows\system32\KERNELBASE.dll
C:\Windows\system32\USER32.dll
C:\Windows\system32\GDI32.dll
C:\Windows\system32\LPK.dll
C:\Windows\system32\USP10.dll
C:\Windows\system32\msvcrt.dll
C:\Windows\system32\ADVAPI32.dll
C:\Windows\SYSTEM32\sechost.dll
C:\Windows\system32\RPCRT4.dll
C:\Windows\system32\IMM32.DLL
C:\Windows\system32\MSCTF.dll
C:\Windows\system32\apphelp.dll
C:\Windows\system32\CRYPTSP.dll
C:\Windows\system32\rsaenh.dll
C:\Windows\system32\CRYPTBASE.dll
C:\Windows\system32\SHELL32.dll
C:\Windows\system32\SHLWAPI.dll
C:\Windows\system32\MSVCP60.dll
C:\Windows\system32\ntmarta.dll
C:\Windows\system32\WLDAP32.dll
C:\Windows\system32\ole32.dll
C:\Windows\system32\SspiCli.dll

...

tasksche.exe pid:   1392
Command line :
 "C:\ProgramData\zmlrzffkezkivj575\tasksche.exe"
Service Pack 1

Path
----
C:\ProgramData\zmlrzffkezkivj575\tasksche.exe
C:\Windows\SYSTEM32\ntdll.dll
C:\Windows\system32\kernel32.dll
C:\Windows\system32\KERNELBASE.dll
C:\Windows\system32\USER32.dll
C:\Windows\system32\GDI32.dll
C:\Windows\system32\LPK.dll
C:\Windows\system32\USP10.dll
C:\Windows\system32\msvcrt.dll
C:\Windows\system32\ADVAPI32.dll
C:\Windows\SYSTEM32\sechost.dll
C:\Windows\system32\RPCRT4.dll
C:\Windows\system32\IMM32.DLL
C:\Windows\system32\MSCTF.dll
C:\Windows\system32\apphelp.dll
C:\Windows\system32\CRYPTSP.dll
C:\Windows\system32\rsaenh.dll
C:\Windows\system32\CRYPTBASE.dll
C:\Windows\system32\SHELL32.dll
C:\Windows\system32\SHLWAPI.dll
C:\Windows\system32\MSVCP60.dll

...

@WanaDecryptor pid:   7676
Command line :
"C:\ProgramData\zmlrzffkezkivj575\@WanaDecryptor@.exe"
Service Pack 1

Path
----
C:\ProgramData\zmlrzffkezkivj575\@WanaDecryptor@.exe
C:\Windows\SYSTEM32\ntdll.dll
C:\Windows\system32\kernel32.dll
C:\Windows\system32\KERNELBASE.dll
C:\Windows\system32\MFC42.DLL
C:\Windows\system32\msvcrt.dll
C:\Windows\system32\USER32.dll
C:\Windows\system32\GDI32.dll
C:\Windows\system32\LPK.dll
C:\Windows\system32\USP10.dll
C:\Windows\system32\ole32.dll
C:\Windows\system32\RPCRT4.dll
C:\Windows\system32\OLEAUT32.dll
C:\Windows\system32\ODBC32.dll
C:\Windows\system32\ADVAPI32.dll
C:\Windows\SYSTEM32\sechost.dll
C:\Windows\system32\SHELL32.dll
C:\Windows\system32\SHLWAPI.dll
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2\COMCTL32.dll
C:\Windows\system32\urlmon.dll
C:\Windows\system32\api-ms-win-downlevel-ole32-l1-1-0.dll
C:\Windows\system32\api-ms-win-downlevel-shlwapi-l1-1-0.dll
C:\Windows\system32\api-ms-win-downlevel-advapi32-l1-1-0.dll
C:\Windows\system32\api-ms-win-downlevel-user32-l1-1-0.dll
C:\Windows\system32\api-ms-win-downlevel-version-l1-1-0.dll
C:\Windows\system32\version.DLL
C:\Windows\system32\api-ms-win-downlevel-normaliz-l1-1-0.dll
C:\Windows\system32\normaliz.DLL
C:\Windows\system32\iertutil.dll
C:\Windows\system32\WININET.dll
C:\Windows\system32\USERENV.dll
C:\Windows\system32\profapi.dll
C:\Windows\system32\MSVCP60.dll
C:\Windows\system32\WS2_32.dll
C:\Windows\system32\NSI.dll
C:\Windows\system32\IMM32.DLL
C:\Windows\system32\MSCTF.dll
C:\Windows\system32\odbcint.dll
C:\Windows\system32\RICHED32.DLL
C:\Windows\system32\RICHED20.dll
C:\Windows\system32\uxtheme.dll
C:\Windows\system32\dwmapi.dll
C:\Windows\system32\IconCodecService.dll
C:\Windows\system32\WindowsCodecs.dll
C:\Windows\system32\msls31.dll
C:\Windows\system32\CRYPTBASE.dll
C:\Program Files\Unlocker\UnlockerHook.dll
```

Se puede observar en los resultados obtenidos que los procesos involucrados están utilizando DLLs vinculadas a funciones de cifrado: CRYPTSP.dll y CRYPTBASE.dll.

[5] Nótese que no figura el proceso *taskdl.exe* porque al momento de realizar el volcado de memoria, éste ya había finalizado.

Profundizando en este análisis, se buscan las funciones que exportan estas DLLs. De esta manera, se busca obtener un indicador sobre la cantidad de funciones por librería y sus respectivos nombres. Se pretende así entender un poco más sobre la funcionalidad requerida por el software a las DLLs. Lo que se hace es enumerar las funciones con el *plugin* `enumfunc`, y luego procesar la salida para obtener los datos de manera resumida (ver Tabla 1).

Del listado de DLLs cargadas por estos procesos, resulta interesante analizar en particular CRYPTSP.dll. Antes de continuar, es importante notar una diferencia no menor que se observa entre los dos proceso @WannaDecryptor@.exe y tasksche.exe[25], y es que solo este último utiliza CRYPTSP.dll. Esto se debe a que es el proceso encargado de cifrar los datos en el equipo, mientras que @WannaDecryptor@.exe es la aplicación encargada de comunicarse con el usuario para pedir el rescate y brindar los datos de pago.

	Proceso	
	@Wanna…	**tasksche**
ADVAPI32.dll	806	806
api-ms-win-downlevel-advapi32-l1-1-0.dll	145	0
api-ms-win-downlevel-normaliz-l1-1-0.dll	2	0
api-ms-win-downlevel-ole32-l1-1-0.dll	48	0
api-ms-win-downlevel-shlwapi-l1-1-0.dll	155	0
api-ms-win-downlevel-user32-l1-1-0.dll	22	0
api-ms-win-downlevel-version-l1-1-0.dll	6	0
apphelp.dll		195
COMCTL32.dll	147	0
CRYPTBASE.dll	11	11
CRYPTSP.dll		41
dwmapi.dll	26	0
GDI32.dll	715	715
iertutil.dll	40	0
IMM32.DLL	134	134
kernel32.dll	1365	1365
KERNELBASE.dll	633	633
LPK.dll	11	11
MFC42.DLL	6	0
MSCTF.dll	71	71
msls31.dll	79	0
MSVCP60.dll	1307	1113
msvcrt.dll	1396	1396
normaliz.DLL	5	0
NSI.dll	26	0
ntdll.dll	1983	1884
ntmarta.dll		22
ODBC32.dll	102	0
ole32.dll	395	197,5
OLEAUT32.dll	405	0
profapi.dll		0
RICHED20.dll	9	0
RPCRT4.dll	541	540
rsaenh.dll		27
sechost.dll	60	60
SHELL32.dll	333	333
SHLWAPI.dll	369	369
SspiCli.dll		48,5
UnlockerHook.dll	2	0
urlmon.dll	126	0
USER32.dll	822	822
USERENV.dll	48	0
USP10.dll	44	44
uxtheme.dll	74	0
version.DLL	15	0
WindowsCodecs.dll	115	0
WININET.dll	292	0
WLDAP32.dll		34
WS2_32.dll	167	0

Tabla 1 – Cantidad de funciones por DLLs de los procesos maliciosos.

En la Tabla 2, se muestra un listado de las funciones exportadas por CRYPTSP.dll. De este listado se puede confirmar que ésta es la DLL que provee la funcionalidad de cifrado al *malware*. Particularmente, la función *CryptRandomGen* genera las claves AES que luego se utilizan para cifrar los archivos.

Orden	**Dirección**	**Función**
1	0x00000000754b4658	CheckSignatureInFile
2	0x00000000754b4a53	CryptAcquireContextA
3	0x00000000754b6b98	CryptAcquireContextW
4	0x00000000754b3629	CryptContextAddRef
5	0x00000000754b5d1b	CryptCreateHash
6	0x00000000754b5c1b	CryptDecrypt
7	0x00000000754b5324	CryptDeriveKey
8	0x00000000754b6135	CryptDestroyHash

9	0x00000000754b54a3	CryptDestroyKey
10	0x00000000754b5e3f	CryptDuplicateHash
11	0x00000000754b5217	CryptDuplicateKey
12	0x00000000754b5b18	CryptEncrypt
13	0x00000000754b385e	CryptEnumProviderTypesA
14	0x00000000754b68fa	CryptEnumProviderTypesW
15	0x00000000754b3a86	CryptEnumProvidersA
16	0x00000000754b6a51	CryptEnumProvidersW
17	0x00000000754b588d	CryptExportKey
18	0x00000000754b512b	CryptGenKey
19	**0x00000000754b5723**	**CryptGenRandom**
20	0x00000000754b6731	CryptGetDefaultProviderA
21	0x00000000754b70ea	CryptGetDefaultProviderW
22	0x00000000754b667c	CryptGetHashParam
23	0x00000000754b566e	CryptGetKeyParam
24	0x00000000754b651a	CryptGetProvParam
25	0x00000000754b57b3	CryptGetUserKey
26	0x00000000754b5f62	CryptHashData
27	0x00000000754b604a	CryptHashSessionKey
28	0x00000000754b598d	CryptImportKey
29	0x00000000754b36a0	CryptReleaseContext
30	0x00000000754b65b0	CryptSetHashParam
31	0x00000000754b55a2	CryptSetKeyParam
32	0x00000000754b376b	CryptSetProvParam
33	0x00000000754b6c83	CryptSetProviderA
34	0x00000000754b6d9f	CryptSetProviderExA
35	0x00000000754b72b4	CryptSetProviderExW
36	0x00000000754b722b	CryptSetProviderW
37	0x00000000754b62ca	CryptSignHashA
38	0x00000000754b62ba	CryptSignHashW
39	0x00000000754b6472	CryptVerifySignatureA
40	0x00000000754b6462	CryptVerifySignatureW
41	0x00000000754b4658	SystemFunction035

Tabla 2 – Funciones exportadas por CRYPTSP.dll.

Se ha mostrado entonces cómo diferentes indicadores de procesos, en este caso originarios de la memoria, pueden ir proveyendo información sobre un sistema, la presencia de *malware* (que en este caso era conocida), y cómo el acceso a determinadas DLLs y funciones puede hacer evidente su función o propósito.

VIII. Conclusiones y Trabajo Futuro

El análisis de memoria es clave para la confección de indiciadores para la detección de malware, pero también hay otros artefactos que se pueden encontrar en otros medios digitales que aportan información relevante. Estos últimos pueden ser simples y no necesariamente concluyentes, pero si aportar datos de utilidad para guiar el análisis y poder demostrar la presencia de *malware* en un equipo, y además entender su comportamiento.

En general, el análisis preponderante es siempre "en vivo", contra sistemas activos y en el momento en que el ataque sobre el sistema se está llevando a cabo, pero existe una amplia posibilidad (y una demanda) de realizar este tipo de estudios *post-mortem*.

En análisis de la memoria en base a un volcado realizado en el momento de del ataque (o luego, en base a una máquina virtual que cargue una imagen forense del disco), el análisis del Registro de Windows y de las DLLs son ejemplos de lo que es posible lograr, y de los indicadores que se pueden construir, para entender cuál es objetivo de un ataque y cómo se pretende lograrlo.

Se mostró cómo en base a las estructuras del sistema operativo Windows, y complementando ese conocimiento con desarrollos ya existentes (enfocados en otras problemáticas) se pueden confeccionar indicadores que están adaptados a las necesidades forenses. Es importante continuar con este trabajo, para ir formulando otros indicadores, enfocados en distintas fuentes de información forense.

También es importante seguir trabajando alrededor de los indicadores, tanto los ya existentes como nuevos que podrían formularse a futuro, para establecer una metodología probada de detección de *malware* en entornos forenses.

Agradecimientos

Los autores quieren agradecer al Decano de la Facultad de Ingeniería de la Universidad FASTA, Ing. Roberto Giordano, y a la Secretaria de Investigación de dicha Facultad, Lic. Mónica Pascual.

También se extiende el agradecimiento a Fernando Greco, Ariel Podestá, Martín Castellote, Juan Ignacio Iturriaga y Sebastián Lasia, integrantes del Grupo de Investigación en Sistemas Operativos e Informática Forense que colaboraron con la escritura, edición y revisión de este trabajo.

Referencias

[1] M. Hypponen, *The History and Evolution of Computer Viruses*. DEFCON 19, 2011.

[2] M. Hutchins, *How to Accidentally Stop a Global Cyber Attack*. Disponible en: https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html.

[3] M. Suiche, *WannaCry – The largest ransom-ware infection in History*. Disponible en: https://blog.comae.io/wannacry-the-largest-ransom-ware-infection-in-history-f37da8e30a58.

[4] MISP – Malware Information Sharing Platform and Threat Sharing http://www.misp-project.org/.

[5] OpenIOC Framework http://www.openioc.org/.

[6] M. Sikorski, A. Honig, *Practical Malware Analysis*. No Starch Press, San Francisco, California, EEUU, 2012. ISBN: 978-1-59327-290-6.

[7] radare/r2 http://www.radare.org/r/.

[8] B. Delpy, *wanakiwi*, software disponible en: https://github.com/gentilkiwi/wanakiwi. (2017).

[9] M. Suiche, *WannaCry – Decrypting files with WanaKiwi+Demos*, disponible en: https://blog.comae.io/wannacry-decrypting-files-with-wanakiwi-demo-86bafb81112d.
[10] Microsoft, *Microsoft Portable Executable and Common Object File Format Specification*. Documentación oficial, disponible en: https://www.microsoft.com/en-us/download/details.aspx?id=19509.
[11] Microsoft, *About COFF (Windows CE 5.0)*. Documentación oficial, disponible en: https://msdn.microsoft.com/en-us/library/aa448549.aspx.
[12] M. Pietrek, *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*. Microsoft Systems Journal, Marzo 1994.
[13] A. Albertini, *Exploring the Portable Executable format*, 44 CON, Londres, Inglaterra, Septiembre 2013. Disponible en: https://www.slideshare.net/ange4771/44con2013-workshop-exploring-the-portable-executable-format.
[14] A. H. Di Iorio, G. M. Ruiz de Angeli, J. I. Alberdi, B. Constanzo, A. Podestá, M. Castellote, *Quitando el Velo a la Memoria: Estructuras Ocultas y Malware*. CIBSI 2015, Quito, Ecuador, 2015.
[15] J. Kong, *Designing BSD Rootkits – An introduction to kernel hacking*. No Starch Press, San Francisco, California, EEUU, 2007. ISBN: 978-1-59327-142-8.
[16] K. Lee, H. Hwang, K. Kim, B. Noh, *Robust bootstrapping memory analysis against anti-forensics*. DFRWS USA 2016,
[17] J. Butler, *DKOM (Direct Kernel Object Manipulation)*. Black Hat Windows 2004, Seattle, EEUU, 2004.
[18] J. Butler, *VICE – Catch the Hookers!*. Black Hat USA 2004, Las Vegas, EEUU, 2004.
[19] J. Kornblum, *Windows Memory Forensics and Direct Kernel Object Manipulation*. DOD Cyber Crime Conference, 2011.
[20] M. H. Ligh, A. Case, J. Levy, A. Walters, *The Art of Memory Forensics*. Wiley, Inianapolis, Indiana, EEUU. 2014. ISBN: 978 -1-118 -82509-9.
[21] A. Schuster, *Searching for Processes and Threads in Microsoft Windows Memory Dumps*. DFRWS 2006, Lafayette, EEUU, Agosto 2006.
[22] Microsoft, *About the Registry: Predefined Keys*. Documentación official, disponible en: https://msdn.microsoft.com/en-us/library/windows/desktop/ms724836(v=vs.85).aspx.
[23] Volatility Framework, https://github.com/volatilityfoundation/volatility.
[24] Volatility Foundations, http://www.volatilityfoundation.org/.
[25] Panda Security, *Informe #WannaCry*. 2017, disponible en: http://www.pandasecurity.com/spain/mediacenter/src/uploads/2017/05/Informe_WannaCry-es.pdf.
[26] M. Russinovich, D. Solomon, A. Ionescu, *Windows Internals 5*, 2009.
[27] M. Russinovich, D. Solomon, A. Ionescu, *Windows Internals 6*, 2012.
[28] A. H. Di Iorio, et al., *El Rastro Digital del Delito*, Editorial Universidad FASTA, Mar del Plata, Argentina, 2017. ISBN: 978-987-1312-81-8.

Gonzalo Ruiz de Angeli es Ingeniero en Informática de la Universidad FASTA, docente e investigador graduado del Grupo de Investigación en Sistemas Operativos e Informática Forense, y docente en Sistemas Distribuídos en la carrera de Ingeniería en Informática. Co-autor de BIP-M Framework, un software para el análisis forense de memoria principal. Ha presentado trabajos en mútiples congresos nacionales e internacionales.

Juan Ignacio Alberdi es Ingeniero en Informática de la Universidad FASTA, docente e investigador graduado del Grupo de Investigación en Sistemas Operativos e Informática Forense, y docente en Sistemas Distribuídos en la carrera de Ingeniería en Informática. Co-autor de *BIP-M Framework*, un software para el análisis forense de memoria principal. Ha presentado trabajos en mútiples congresos nacionales e internacionales.

Bruno Constanzo es Ingeniero en Informática de la Universidad FASTA, Investigador del InFo-Lab y docente en materias de Ingeniería en Informática y Licenciatura en Criminalística. Ha sido tutor en varios proyectos finales y tésis vinculadas a la Informática Forense, y participó en múltiples proyectos de investigación relacionados con la temática. Ha presentado trabajos en numerosos congresos nacionales e internacionales.

Hugo Curti es Ingeniero de Sistemas y Magíster en Ingeniería de Sistemas recibido en la Universidad Nacional del Centro de la Provincia de Buenos Aires. Se desempeña como Profesor Adjunto del Departamento de Computación y Sistemas de la Facultad de Ciencias Exactas de dicha Universidad, y como Profesor Titular de la Cátedra de Sistemas Distribuidos en la Univerisdad FASTA. Es miembro de varios grupos de investigación, entre los que se cuenta el Grupo de Investigación en Sistemas Operativos e Informática Forenses. Ha presentado trabajos en numerosos congresos nacionales e internacionales.

Ana Haydée Di Iorio es ingeniera en Informática de la Universidad FASTA y Directora del InFo-Lab, Laboratorio de Investigación y Desarrollo de Tecnología en Informática Forense. Es miembro de la Comisión Asesora de la Red de Laboratorios Forenses de Ciencia y Tecnología del Ministerio de Ciencia Tecnología e Innovación Productiva de la República Argentina. Ha participado, presidido y dictado conferencias en numerosos congresos nacionales e internacionales.