Extracción y análisis de malware con Volatility 3

En este laboratorio vas a introducirte en el análisis de malware a partir de volcados de memoria con el entorno Volatility. En concreto, vamos a analizar el volcado de memoria disponible en `https://webdiis.unizar.es/~ricardo/sbc-2022/malware-forense-memoria/adicional/volcados/wannacry.elf.tar.gz`, que es un volcado de una máquina de Windows 7 infectada con WannaCry (hash MD5 de la muestra: `84c82835a5d21bbcf75a61706d8ab549`), con Volatility versión 3.

1. Identificación de procesos

Una vez descargado el volcado de memoria, lo primero es identificar los procesos que aparecen en él. Para ello, vamos a hacer uso del plugin `PsList`:

```
python3 vol.py -f ~/volcados/wannacry.elf windows.pslist.PsList
```

La ejecución de este comando nos muestra los procesos capturados en el volcado:

```
Volatility 3 Framework 1.0.1
Progress:  100.00       PDB scanning finished
PID PPID    ImageFileName   Offset(V)   Threads Handles SessionId   Wow64   CreateTime  ExitTime    File output

4   0   System  0x841389c8  85  496 N/A False   2021-07-15 04:13:26.000000  N/A Disabled
240 4   smss.exe    0x8553dd20  2   29  N/A False   2021-07-15 04:13:26.000000  N/A Disabled
[... omitido ...]
2288    2176    SearchFilterHo  0x8429ac08  6   111 0   False   2021-07-14 19:21:01.000000  N/A Disabled
3812    832 ed01ebfbc9eb5b  0x858282a8  10  83  1   False   2021-07-14 19:21:30.000000  N/A Disabled
816 3812    @WanaDecryptor  0x8502d030  2   60  1   False   2021-07-14 19:22:02.000000  N/A Disabled
3920    816 taskhsvc.exe    0x84bfa3a8  6   119 1   False   2021-07-14 19:22:05.000000  N/A Disabled
172 364 conhost.exe 0x84bc8030  1   33  1   False   2021-07-14 19:22:05.000000  N/A Disabled
1252    3812    @WanaDecryptor  0x84adf030  1   63  1   False   2021-07-14 19:22:18.000000  N/A Disabled
2880    568 dllhost.exe 0x84a0f030  6   83  1   False   2021-07-14 19:22:31.000000  N/A Disabled
3048    568 dllhost.exe 0x857d1a60  6   78  0   False   2021-07-14 19:22:31.000000  N/A Disabled
2932    448 VSSVC.exe   0x84e41a48  7   125 0   False   2021-07-14 19:22:31.000000  N/A Disabled
3424    448 svchost.exe 0x84f63030  6   70  0   False   2021-07-14 19:22:32.000000  N/A Disabled
520 568 WmiPrvSE.exe    0x84e0e030  8   112 0   False   2021-07-14 19:22:32.000000  N/A Disabled
2936    448 wbengine.exe    0x845e5030  7   113 0   False   2021-07-14 19:22:33.000000  N/A Disabled
3400    568 vdsldr.exe  0x84b50030  6   74  0   False   2021-07-14 19:22:33.000000  N/A Disabled
2300    448 vds.exe 0x84e1b030  14  148 0   False   2021-07-14 19:22:33.000000  N/A Disabled
```

De este listado, existen dos que llaman poderosamente la atención. Concretamente, los procesos con identificador 816 y 1252, llamados `@WanaDecryptor`. Podemos ejecutar ahora el plugin `PsTree` para observar la jerarquía de los procesos en más detalle:

```
python3 vol.py -f ~/volcados/wannacry.elf windows.pstree.PsTree
Volatility 3 Framework 1.0.1
Progress:  100.00       PDB scanning finished
PID PPID    ImageFileName   Offset(V)   Threads Handles SessionId   Wow64   CreateTime  ExitTime

4   0   System  0x84e1b030  85  496 N/A False   2021-07-15 04:13:26.000000  N/A
* 240   4   smss.exe    0x84e1b030  2   29  N/A False   2021-07-15 04:13:26.000000  N/A
320 312 csrss.exe   0x84e1b030  8   657 0   False   2021-07-15 04:13:28.000000  N/A
356 312 wininit.exe 0x84e1b030  3   76  0   False   2021-07-15 04:13:29.000000  N/A
[... omitido ...]
* 172   364 conhost.exe 0x84e1b030  1   33  1   False   2021-07-14 19:22:05.000000  N/A
404 348 winlogon.exe    0x84e1b030  5   121 1   False   2021-07-15 04:13:29.000000  N/A
832 840 explorer.exe    0x84e1b030  52  1262    1   False   2021-07-14 19:13:40.000000  N/A
* 3812  832 ed01ebfbc9eb5b  0x84e1b030  10  83  1   False   2021-07-14 19:21:30.000000  N/A
** 816  3812    @WanaDecryptor  0x84e1b030  2   60  1   False   2021-07-14 19:22:02.000000  N/A
*** 3920    816 taskhsvc.exe    0x84e1b030  6   119 1   False   2021-07-14 19:22:05.000000  N/A
** 1252 3812    @WanaDecryptor  0x84e1b030  1   63  1   False   2021-07-14 19:22:18.000000  N/A
* 1604  832 VBoxTray.exe    0x84e1b030  13  162 1   False   2021-07-14 19:13:40.000000  N/A
```

Recuerda que este plugin nos permite ver la jerarquía de los procesos capturados en el volcado. En este caso, observamos que los dos procesos `@WanaDecryptor` destacados anteriormente son procesos hijos de un proceso denominado `ed01ebfbc9eb5b`. Aparece también un cuarto proceso, que es hijo del proceso con identificador 816, con nombre `taskhsvc.exe` .

Por el nombre de los procesos, todo parece indicar que se trata del ransomware WannaCry. Este ransomware se comunicaba con su servidor de mando y control a través de la red Tor, distribuyendo junto con el ransomware un servidor de Tor local que se renombraba y ejecutaba como `taskhsvc.exe`.

Podemos ahora echarle un vistazo con el plugin `DllList` a las bibliotecas dinámicas que están asociadas a `taskhsvc.exe` (identificador de proceso 3920):

```
python3 vol.py -f ~/volcados/wannacry.elf windows.dlllist.DllList --pid 3920
```

```
Volatility 3 Framework 1.0.1
Progress:  100.00       PDB scanning finished
PID Process Base    Size    Name    Path    LoadTime    File output

3920    taskhsvc.exe    0x1090000   0x2fe000    taskhsvc.exe    C:\Path\TaskData\Tor\taskhsvc.exe   N/A
3920    taskhsvc.exe    0x770c0000  0x142000    ntdll.dll   C:\Windows\SYSTEM32\ntdll.dll   N/A Disable
3920    taskhsvc.exe    0x756b0000  0xd5000 kernel32.dll    C:\Windows\system32\kernel32.dll    2021-07-
3920    taskhsvc.exe    0x75250000  0x4b000 KERNELBASE.dll  C:\Windows\system32\KERNELBASE.dll  2021-07-
3920    taskhsvc.exe    0x62d20000  0x82000 libevent-2-0-5.dll  C:\Path\TaskData\Tor\libevent-2-0-5.dll
3920    taskhsvc.exe    0x6b5d0000  0x1c000 libssp-0.dll    C:\Path\TaskData\Tor\libssp-0.dll    2021-07-
[... omitido ...]
3920    taskhsvc.exe    0x772d0000  0x35000 WS2_32.dll  C:\Windows\system32\WS2_32.dll  2021-07-14 19:2
3920    taskhsvc.exe    0x76f80000  0x6000  NSI.dll C:\Windows\system32\NSI.dll 2021-07-14 19:22:05.000
3920    taskhsvc.exe    0x622c0000  0x21c000    LIBEAY32.dll    C:\Path\TaskData\Tor\LIBEAY32.dll   202
3920    taskhsvc.exe    0x62230000  0x82000 SSLEAY32.dll    C:\Users\IEUser\Desktop\ed01ebfbc9eb5bbea54
[... omitido ...]
3920    taskhsvc.exe    0x74b90000  0x3c000 mswsock.dll C:\Windows\system32\mswsock.dll 2021-07-14 19:2
3920    taskhsvc.exe    0x746e0000  0x5000  wshtcpip.dll    C:\Windows\System32\wshtcpip.dll    2021-07-
3920    taskhsvc.exe    0x74750000  0x1c000 iphlpapi.dll    C:\Windows\system32\iphlpapi.dll    2021-07-
3920    taskhsvc.exe    0x74740000  0x7000  WINNSI.DLL  C:\Windows\system32\WINNSI.DLL  2021-07-14 19:2
3920    taskhsvc.exe    0x73310000  0xd000  dhcpcsvc6.DLL   C:\Windows\system32\dhcpcsvc6.DLL   2021-07-
3920    taskhsvc.exe    0x746f0000  0x12000 dhcpcsvc.DLL    C:\Windows\system32\dhcpcsvc.DLL    2021-07-
```

Como se observa, efectivamente este proceso tiene cargadas ciertas bibliotecas usadas por el servidor Tor. Además, estas bibliotecas están guardadas en una carpeta que tiene el mismo nombre. Se observan además otras bibliotecas relacionadas con el manejo de sockets y conectividad a Internet de Windows, como «`WS_2.32.dll`», «`mswsock.dll`», o «`dhcpcsvc.dll`», entre otras.

Si observamos las bibliotecas relacionadas con el primer proceso `@WanaDecryptor` (identificador

816), se encuentran bibliotecas relacionadas con el entorno de desarrollo de Microsoft Visual 6.0, con temas de redes y creación de sockets, con manejo del Registro de Windows («`ADVAPI32.dll`»), con criptografía («`CRYPT32.dll`»), entre otras.

```
python3 vol.py -f ~/volcados/wannacry.elf -r json windows.dlllist.DllList --pid 816 | grep Path
Volatility 3 Framework 1.0.1
    "Path": "C:\\Users\\IEUser\\Desktop\\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41a
    "Path": "C:\\Windows\\SYSTEM32\\ntdll.dll",
    "Path": "C:\\Windows\\system32\\kernel32.dll",
    "Path": "C:\\Windows\\system32\\KERNELBASE.dll",
    "Path": "C:\\Windows\\system32\\MFC42.DLL",
    "Path": "C:\\Windows\\system32\\msvcrt.dll",
    "Path": "C:\\Windows\\system32\\USER32.dll",
    "Path": "C:\\Windows\\system32\\GDI32.dll",
    "Path": "C:\\Windows\\system32\\LPK.dll",
    "Path": "C:\\Windows\\system32\\USP10.dll",
    "Path": "C:\\Windows\\system32\\ole32.dll",
    "Path": "C:\\Windows\\system32\\RPCRT4.dll",
    "Path": "C:\\Windows\\system32\\OLEAUT32.dll",
    "Path": "C:\\Windows\\system32\\ODBC32.dll",
    "Path": "C:\\Windows\\system32\\ADVAPI32.dll",
    "Path": "C:\\Windows\\SYSTEM32\\sechost.dll",
    "Path": "C:\\Windows\\system32\\SHELL32.dll",
    "Path": "C:\\Windows\\system32\\SHLWAPI.dll",
    "Path": "C:\\Windows\\WinSxS\\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.18837
    "Path": "C:\\Windows\\system32\\urlmon.dll",
    "Path": "C:\\Windows\\system32\\XmlLite.dll",
    "Path": "C:\\Windows\\system32\\WININET.dll",
    "Path": "C:\\Windows\\system32\\iertutil.dll",
    "Path": "C:\\Windows\\system32\\CRYPT32.dll",
    "Path": "C:\\Windows\\system32\\MSASN1.dll",
    "Path": "C:\\Windows\\system32\\MSVCP60.dll",
    "Path": "C:\\Windows\\system32\\WS2_32.dll",
    "Path": "C:\\Windows\\system32\\NSI.dll",
    "Path": "C:\\Windows\\system32\\IMM32.DLL",
    "Path": "C:\\Windows\\system32\\MSCTF.dll",
    "Path": "C:\\Windows\\system32\\odbcint.dll",
    "Path": "C:\\Windows\\system32\\RICHED32.DLL",
    "Path": "C:\\Windows\\system32\\RICHED20.dll",
    "Path": "C:\\Windows\\system32\\uxtheme.dll",
    "Path": "C:\\Windows\\system32\\dwmapi.dll",
    "Path": "C:\\Windows\\system32\\mswsock.dll",
    "Path": "C:\\Windows\\System32\\wshtcpip.dll",
    "Path": "C:\\Windows\\system32\\apphelp.dll",
```

2. Identificación de objetos de interés

Si analizamos ahora los manejadores de archivos asociados a los ficheros, en especial los de objetos mutex, encontraremos un objeto de tipo Mutex asociado al proceso con identificador 3812 con nombre `MsWinZonesCacheCounterMutexA`.

```
python3 vol.py -f ~/volcados/wannacry.elf windows.handles.Handles --pid 3812 | grep Mutant
3812ressed01ebfbc9eb5b  0x85d22650B scan0x30 finMutant  0x1f0001
3812    ed01ebfbc9eb5b  0x855e59b0  0x34    Mutant  0x1f0001
3812    ed01ebfbc9eb5b  0x8561e9e8  0x6c    Mutant  0x1f0001    MsWinZonesCacheCounterMutexA
3812    ed01ebfbc9eb5b  0x85562278  0x74    Mutant  0x1f0001    MsWinZonesCacheCounterMutexA0
3812    ed01ebfbc9eb5b  0x852ab768  0x150   Mutant  0x1f0001
```

Este mutex es el que crea la muestra de WannaCry para no reinfectar la máquina. En el primer momento de infección, crea este mutex de manera que posteriores ejecuciones del malware no infectarán la máquina al encontrar este mutex ya creado. Esta forma de evitar reinfecciones es

muy habitual del malware, así como mecanismo de defensa para evitar infecciones de máquinas no comprometidas.

3. Identificación de persistencia

Recuerda que una de las fases del código dañino es realizar persistencia en los sistemas infectados para garantizar que siguen realizando su actividad dañina tras el reinicio del sistema. Puedes encontrar una clasificación de todos los posibles métodos de persistencia que puede explotar un malware en "*Characteristics and Detectability of Windows Auto-Start Extensibility Points in Memory Forensics*" (`https://doi.org/10.1016/j.diin.2019.01.026`).

Para localizar las claves de registro que están accesibles en el volcado de memoria, puedes hacer uso del plugin `PrintKey`, cuyo funcionamiento es similar al plugin con el mismo nombre de Volatility 2:

```
python3 vol.py -f ~/volcados/wannacry.elf windows.registry.printkey.PrintKey
Volatility 3 Framework 1.0.1
Progress:  100.00       PDB scanning finished
Last Write Time Hive Offset Type    Key Name    Data    Volatile

2021-07-15 04:13:21.000000  0x87c0e140  Key [NONAME]    A       False
2021-07-15 04:13:21.000000  0x87c0e140  Key [NONAME]    MACHINE     False
2021-07-15 04:13:21.000000  0x87c0e140  Key [NONAME]    USER        False
2021-07-15 04:13:21.000000  0x87c1a008  Key \REGISTRY\MACHINE\SYSTEM    ControlSet001       False
2021-07-15 04:13:21.000000  0x87c1a008  Key \REGISTRY\MACHINE\SYSTEM    ControlSet002       False
2021-07-15 04:13:21.000000  0x87c1a008  Key \REGISTRY\MACHINE\SYSTEM    MountedDevices      False
2021-07-15 04:13:21.000000  0x87c1a008  Key \REGISTRY\MACHINE\SYSTEM    RNG     False
2021-07-15 04:13:21.000000  0x87c1a008  Key \REGISTRY\MACHINE\SYSTEM    Select      False
[... omitido ...]
```

4. Volcado de procesos y DLLs

Una vez se tenga detectado los procesos y/o librerías que se quieran extraer para realizar un análisis más detallado de su código, se puede usar el propio plugin `PsList` con el parámetro `--dump` para extraer un proceso determinado del volcado. Vamos a extraer el proceso con identificador 3812:

```
python3 vol.py -f ~/volcados/wannacry.elf windows.pslist.PsList --pid 3812 --dump
Volatility 3 Framework 1.0.1
Progress:  100.00       PDB scanning finished
PID PPID    ImageFileName   Offset(V)   Threads Handles SessionId   Wow64   CreateTime  ExitTime    File

3812    832 ed01ebfbc9eb5b  0x858282a8  10  83  1   False   2021-07-14 19:21:30.000000  N/A pid.3812.0x
```

Del mismo modo, si se quisieran extraer las bibliotecas asociadas a un determinado proceso, hay que hacer uso del plugin `DllList` con el parámetro `--dump`:


```
python3 vol.py -f ~/volcados/wannacry.elf windows.dlllist.DllList --pid
    3812  --dump
Volatility 3 Framework 1.0.1
Progress:  100.00       PDB scanning finished
PID Process Base    Size    Name    Path    LoadTime    File output

3812    ed01ebfbc9eb5b  0x400000    0x35a000
    ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.bin
    (1).exe    C:\Users\IEUser\Desktop\
    ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.bin\
    ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.bin
    (1).exe   N/A pid.3812.
    ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.bin
    (1).exe.0x201b78.0x400000.dmp
3812    ed01ebfbc9eb5b  0x770c0000  0x142000    ntdll.dll   C:\Windows\
    SYSTEM32\ntdll.dll   N/A pid.3812.ntdll.dll.0x201bf8.0x770c0000.dmp
3812    ed01ebfbc9eb5b  0x756b0000  0xd5000 kernel32.dll    C:\Windows\
    system32\kernel32.dll    2021-07-14 19:21:30.000000  pid.3812.kernel32
    .dll.0x201ef0.0x756b0000.dmp
3812    ed01ebfbc9eb5b  0x75250000  0x4b000 KERNELBASE.dll  C:\Windows\
    system32\KERNELBASE.dll  2021-07-14 19:21:30.000000  pid.3812.
    KERNELBASE.dll.0x201fd8.0x75250000.dmp
3812    ed01ebfbc9eb5b  0x755e0000  0xc9000 USER32.dll  C:\Windows\
    system32\USER32.dll  2021-07-14 19:21:30.000000  pid.
[... omitido ...]
3812    ed01ebfbc9eb5b  0x74220000  0x6000  IconCodecService.dll    C:\
    Windows\system32\IconCodecService.dll    2021-07-14 19:22:02.000000
    pid.3812.IconCodecService.dll.0x23a170.0x74220000.dmp
3812    ed01ebfbc9eb5b  0x73ac0000  0xfb000 WindowsCodecs.dll   C:\
    Windows\system32\WindowsCodecs.dll   2021-07-14 19:22:02.000000  pid
    .3812.WindowsCodecs.dll.0x23a1f0.0x73ac0000.dmp
```

5. Metodología de análisis de malware

Una vez extraído, estos ficheros volcados pueden ser analizados de la manera habitual en análisis de malware. Recuerda que las preguntas que hay que responder cuando uno realiza un análisis de malware son las siguientes:

- *¿Qué actividad maliciosa está llevando a cabo en el sistema?*
- *¿Cómo se inicia esta actividad?* (es decir, localizar el método de persistencia que utiliza)
- *¿Cuándo infectó el sistema, y cómo consiguió entrar?*
- *¿Cómo se puede eliminar la amenaza y limpiar el sistema infectado, y cómo puedo prevenir que vuelva a ocurrir?*

Para responder a estas preguntas, la metodología del análisis de malware a seguir es la siguiente:

1. **Fase de análisis estático**. Esta fase, también llamada fase de análisis de código muerto, comprende el análisis del fichero binario sin llegar a ejecutarlo. Es decir, en esta fase se leerá el contenido binario (los bytes) del fichero. En esta fase ayudará conocer en profundidad el ensamblador de la arquitectura donde se ejecuta el binario (es aquí donde entran las habilidades de ingeniería inversa que veíamos en la sección anterior). Esta fase se divide en tres aspectos básicos:
    - *Cálculo de firmas MD5/SHA1/SHA256.* En esta subfase se calculará la firma del binario usando algún algoritmo criptográfico como MD5, SHA-1, o SHA-256. Estos

algoritmos permiten, dada una entrada, calcular un número que identifica de manera "exclusiva" al binario. Una vez calculada la firma, se puede proceder a buscarla en Internet con el objetivo de localizar otros análisis de la muestra de malware y facilitar así el trabajo propio de análisis. Una buena herramienta en Windows para esta tarea es `HashTab` (`http://implbits.com/products/hashtab/`). En sistemas macOS o GNU/Linux, se pueden usar los comandos nativos `md5sum` o `sha1sum`, por nombrar algunos.

- *Cadenas del fichero binario.* Esta segunda subfase consiste en localizar todas las cadenas de texto que estén contenidas dentro del binario. Las cadenas de texto, como variables definidas dentro del código fuente original del programa y que éste usa durante su ejecución, se encuentran dentro del fichero ejecutable. De hecho, si el ejecutable no tiene ningún mecanismo de protección (por ejemplo., ofuscación de código o cifrado), estas cadenas se encontrarán en el fichero completamente visibles. Es decir, si viéramos el fichero ejecutable con un procesador de textos, se verían las cadenas de texto claramente. La presencia de ciertas cadenas es un indicio de la actividad que estará realizando la muestra maliciosa. Por ejemplo, si vemos una dirección IP de Internet, o un dominio web y partes de cadenas relativas a una petición HTTP POST o GET, podemos intuir que la muestra maliciosa seguramente se está conectando a dicha IP/dirección web mediante peticiones HTTP POST o GET. Para esta fase, se puede usar una herramienta como `strings` (herramienta nativa en sistemas basados en UNIX, y disponible para Windows en `https://docs.microsoft.com/en-us/sysinternals/downloads/strings`).
- *Funciones de importación.* Por último, la última subfase del análisis estático consiste en analizar las funciones APIs de Windows que está utilizando el fichero ejecutable. Una función API es una función proporcionada por el sistema operativo que facilita la interacción con el mismo por parte de otros programas. Es decir, permite al desarrollador de aplicaciones utilizar los recursos del sistema como ficheros, sockets de red, claves de registro, crear procesos, etc. Por ejemplo, la presencia de funciones relativas a ficheros (por ejemplo, *FindFirstFileA*) llevará a concluir que supuestamente el malware realiza alguna acción con ficheros. Para conocer más sobre las funciones que usa el binario, se recomienda consultar la web del MSDN (`https://msdn.microsoft.com/en-us/library/ms310241`) para consultar una función específica. En concreto, el MSDN proporciona información acerca de qué hace, qué parámetros usa, y qué devuelve cada una de las funciones del sistema. En Windows, para consultar las funciones de importación se pueden usar herramientas como `CFF Explorer` (`http://www.ntcore.com/exsuite.php`).

Todos estos pasos descritos corresponden a un análisis estático básico de la muestra. Un análisis más avanzado usaría herramientas desensambladoras u otras técnicas basadas en grafos de control de flujo o ejecución simbólica, entre otras. En cualquier caso, una de las mayores limitaciones del análisis estático es que el código del binario a analizar no tiene que estar ofuscado o protegido. Además, un análisis estático nos muestra **todos los posibles caminos de ejecución** de ese fichero ejecutable. Es decir, el espacio de estados de todos estos caminos puede ser muy grande y difícil de gestionar.

2. **Fase de análisis dinámico**. Esta fase, también llamada fase de análisis de código vivo o en caliente, comprende el estudio del fichero binario durante su ejecución. En concreto, en esta fase se estudia la interacción de la muestra de malware con su entorno. Esta fase se puede dividir en dos subfases, según lo que se entiende por entorno:

- *Interacción con el Sistema Operativo.* Cuando entendemos por entorno la interacción realizada con el sistema operativo, hay que atender a tres extremos particulares: ficheros (*¿qué archivos está creando, modificando, o eliminando?*, claves de registro (*¿qué claves de registro está creando, modificando, o eliminando?*) y procesos (*¿qué procesos está creando, modificando, o eliminando?*). Si se observa que la muestra de malware está creando nuevos ficheros, habrá que analizar de qué tipo de ficheros se trata y en caso de ser ficheros ejecutables, realizar con ellos un nuevo análisis de malware. En el caso de las claves de registro, hay que comprobar qué claves de registro se ven afectadas (cambios de configuración, persistencia, etc.).
- *Interacción con el exterior (Internet).* Por último, hay que fijarse en la interacción que realiza el malware con Internet. En concreto, habrá que localizar si la muestra de malware realiza alguna conexión a Internet (*¿a qué direcciones IP o dominios de Internet trata de conectarse?*), y en caso de que haga alguna conexión, hay que mirar qué tipo de información le está mandando, cómo la recibe y qué tipo de información recibe. Estos servidores a los que se conecta para mandarles información y posiblemente recibir órdenes se denominan *servidores de mando y control* (*C&C servers*, del inglés).

Esta fase de análisis dinámico que acabamos de describir nos servirá para responder a (casi) todas las cuestiones relativas al malware que se comentaban al inicio de esta sección. Un análisis dinámico más avanzado requeriría usar un depurador para analizar durante tiempo de ejecución cómo van evolucionando los registros de la CPU, los valores que devuelven las funciones de Windows que llama el malware, y demás.

Por último, cabe destacar que **a diferencia del análisis estático, en el análisis dinámico únicamente se analiza uno de los posibles caminos de ejecución**, el cual además puede depender de las condiciones actuales de ejecución.

Una vez extraído, el proceso se puede llevar a herramientas de análisis como desensambladores o depuradores para analizarlos, como es el proceso habitual de análisis de malware. A modo de ejemplo, en la Figura 1 se muestra el análisis del proceso volcado en la herramienta `Ghidra`. En particular, se muestra la parte del código relativa a la creación de mutex.

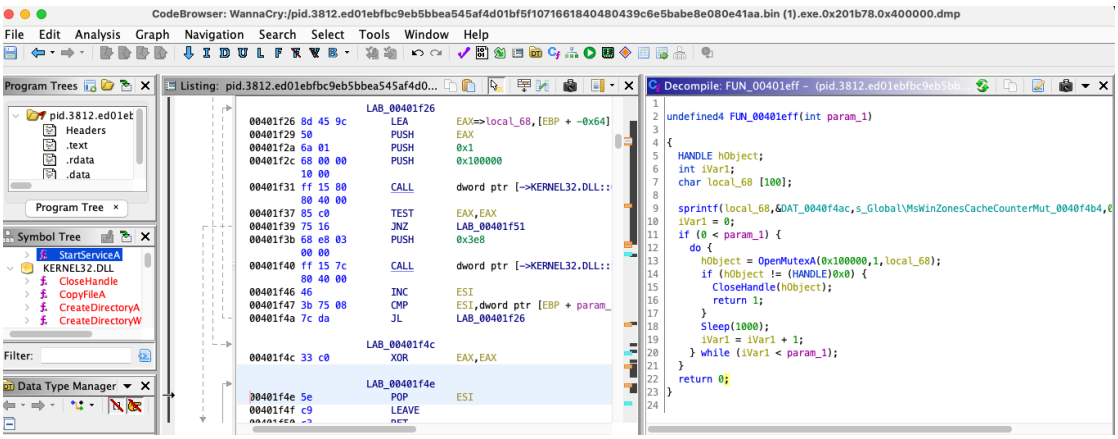

Figura 1: Análisis de la muestra maliciosa extraída del volcado con `Ghidra` versión 10.01.