Malware extraction and analysis with Volatility 3

In this lab you will practice with the analysis of malware from memory dumps with the Volatility environment. Specifically, we are going to analyze with Volatility version 3 the memory dump available at `https://webdiis.unizar.es/~ricardo/sbc-2022/malware-memory-forensics/additional/dumps/wannacry.elf.tar.gz`, which is a dump from a Windows 7 machine infected with WannaCry (sample MD5 hash: `84c82835a5d21bbcf75a61706d8ab549`).

1 Process identification

Once the memory dump is downloaded, the first thing is to identify the processes that appear in it. To do this, we are going to use the `PsList` plugin:

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

Executing this command shows us the processes captured in the 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 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
```

Of this list, there are two that powerfully attract attention. Specifically, the processes with identifier 816 and 1252, called `@WanaDecryptor`. We can now run the `PsTree` plugin to look at the process hierarchy in more detail:

```
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
```

Remember that this plugin allows us to see the hierarchy of the processes captured in the dump. In this case, we see that the two `@WanaDecryptor` processes highlighted above are child processes of a process named `ed01ebfbc9eb5b`. A fourth process also appears, which is a child of the process with identifier 816, with name `taskhsvc.exe`.

From the name of the processes, everything seems to indicate that it is the WannaCry ransomware. This ransomware communicated with its command and control server through the Tor network, distributing along with the ransomware a local Tor server that was renamed and executed as `taskhsvc.exe`.

We can now take a look with the `DllList` plugin at the dynamic libraries that are associated with `taskhsvc.exe` (process id 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-
```

As can be seen, this process effectively has certain libraries used by the Tor server loaded. Also, these libraries are stored in a folder with the same name. Other libraries related to the management of sockets and Windows Internet connectivity are also observed, such as «`WS_2.32.dll`», «`mswsock.dll`», or «`dhcpcsvc.dll`», among others.

If we look at the libraries related to the first process `@WanaDecryptor` (identifier 816), there

are libraries related to the Microsoft Visual 6.0 development environment, with networking and socket creation issues, with handling of the Windows Registry (≪`ADVAPI32.dll`≫), with cryptography (≪`CRYPT32.dll`≫), among others.

```
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 Identification of objects of interest

If we now analyze the file handlers associated with the files, especially those of mutex objects, we will find a Mutex type object associated with the process with identifier 3812 with name `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
```

This mutex is what the WannaCry sample creates so as not to reinfect the machine. At the first moment of infection, it creates this mutex so that subsequent executions of the malware will not infect the machine upon finding this mutex already created. This way of avoiding

reinfections is very common with malware, as well as a defense mechanism to prevent infections from uncompromised machines.

3 Persistence identification

Remember that one of the phases of the malicious code is to perform persistence on infected systems to ensure that they continue to carry out their malicious activity after a system reboot. You can find a classification of all the possible persistence methods that a malware can exploit in "*Characteristics and Detectability of Windows Auto-Start Extensibility Points in Memory Forensics*" (`https://doi.org/10.1016/j.diin.2019.01.026`).

To locate the registry keys that are accessible in the memory dump, you can make use of the `PrintKey` plugin, whose operation is similar to the Volatility 2 plugin of the same name:

```
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 Dumping processes and DLLs

Once you have detected the processes and/or libraries that you want to extract to perform a more detailed analysis of their code, you can use the `PsList` plugin itself with the `--dump` parameter to extract a process determined from the dump. Let's extract the process with identifier 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
```

In the same way, if you want to extract the libraries associated with a certain process, you have to use the `DllList` plugin with the parameter `--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 Malware analysis methodology

Once extracted, these dump files can be analyzed in the usual way for malware analysis. Remember that the questions to answer when performing a malware analysis are the following:

- *What malicious activity is it carrying out on the system?*
- *How does this activity start?* (i.e., locate the persistence method it uses)
- *When did it infect the system, and how did it get in?*
- *How can I remove the threat and clean the infected system, and how can I prevent it from happening again?*

To answer these questions, the malware analysis methodology to follow is as follows:

1. **Static analysis phase**. This phase, also called *dead code* or *cold code analysis phase*, includes the analysis of the binary file without executing it. That is, in this phase the binary content (the bytes) of the file will be read. In this phase it will help to know in depth the assembler of the architecture where the binary is executed (this is where the reverse engineering skills that we saw in the previous section come in). This phase is divided into three basic aspects:
    - *MD5/SHA1/SHA256 signature calculation.* In this subphase, the signature of the binary will be calculated using some cryptographic algorithm such as MD5, SHA-1, or SHA-256. These algorithms allow, given an input, to calculate a number that

"exclusively" identifies the binary. Once the signature has been calculated, it can be searched on the Internet in order to locate other analyzes of the malware sample and thus facilitate the analysis work itself. A good tool on Windows for this task is `HashTab` (`http://implbits.com/products/hashtab/`). On macOS or GNU/Linux systems, you can use the native commands `md5sum` or `sha1sum`, to name a few.

- *Strings within the file.* This second subphase consists of locating all the text strings that are contained within the binary. The text strings, as variables defined within the original source code of the program and that it uses during its execution, are found within the executable file. In fact, if the executable does not have any protection mechanism (e.g., code obfuscation or encryption), these strings will be found in the file completely visible. That is, if we viewed the executable file with a word processor, the text strings would be clearly visible. The presence of certain strings is an indication of the activity that the malicious sample will be carrying out. For example, if we see an Internet IP address, or a web domain and parts of strings relating to an HTTP POST or GET request, we can infer that the malicious sample is surely connecting to that IP/web address via HTTP POST or GET requests. For this phase, you can use a tool like `strings` (native tool on UNIX-based systems, and available for Windows at `https://docs.microsoft.com/en-us/sysinternals/downloads/strings`).
- *Import functions.* Finally, the last subphase of the static analysis consists of analyzing the Windows API functions that the executable file is using. An API function is a function provided by the operating system that makes it easier for other programs to interact with it. That is, it allows the application developer to use system resources such as files, network sockets, registry keys, create processes, etc. For example, the presence of file-related functions (for example, *FindFirstFileA*) will lead to the conclusion that the malware supposedly performs some action with files. To learn more about the functions used by the binary, it is recommended that you consult the MSDN website (`https://msdn.microsoft.com/en-us/library/ms310241`) for a specific function. Specifically, the MSDN provides information about what a system function does, what parameters it uses, and what it returns. On Windows, tools like `CFF Explorer` (`http://www.ntcore.com/exsuite.php`) can be used to check the import functions.

All these described steps correspond to a basic static analysis of the sample. A more advanced analysis would use disassembly tools or other techniques based on flow control graphs or symbolic execution, among others. In any case, one of the biggest limitations of static analysis is that the code of the binary to be analyzed does not have to be obfuscated or protected. Furthermore, a static analysis shows us **all possible execution paths** of that program. That is, the state space of all these paths can be very large and difficult to manage.

2. **Dynamic analysis phase**. This phase, also called *live* or *hot code analysis phase*, includes the study of the program during its execution. Specifically, in this phase the interaction of the malware sample with its environment is studied. This phase can be divided into two sub-phases, depending on what is meant by environment:
   - *Interaction with the Operating System.* When we understand by environment the interaction made with the operating system, we must attend to three particular extremes: files (*what files is the program creating, modifying, or deleting?*, Registry keys (*what registry keys is the program creating, modifying, or deleting?*), and processes

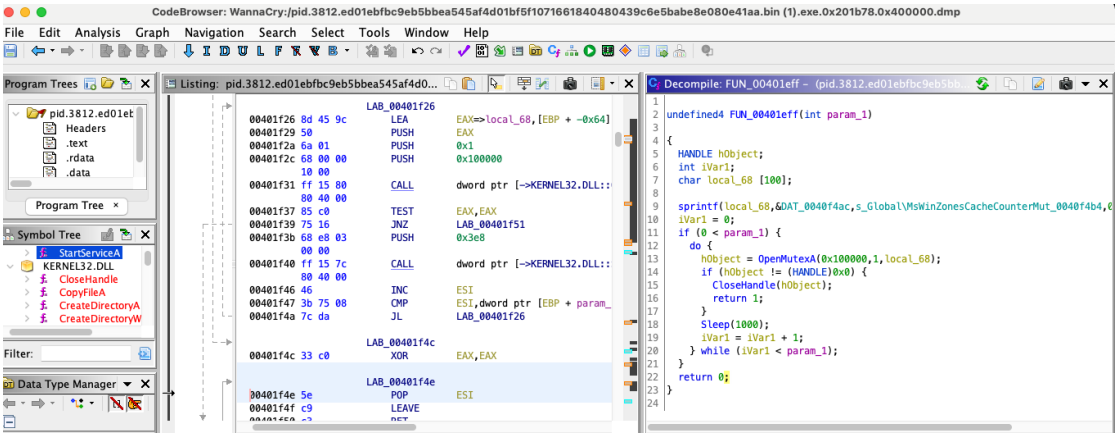

Figure 1: Analysis of the malicious sample extracted from the dump with `Ghidra` version 10.01.

(*what processes is the program creating, modifying, or deleting?*). If the malware sample is observed to be creating new files, it will be necessary to analyze what type of files are created and in the case of executable files, carry out a new malware analysis with them. In the case of Registry keys, it is necessary to check which registry keys are affected (configuration changes, persistence, etc.).

- *Interaction with the outside (Internet).* Finally, we must look at the interaction that the malware performs with the Internet. Specifically, it will be necessary to locate if the malware sample makes any connection to the Internet (*to which IP addresses or Internet domains is it trying to connect?*), and if it does make any connection, it is necessary to look at what type of information is sending, how it receives data and what type of information it receives. These servers to which you connect to send information and possibly receive orders are called *command and control servers* (*C&C servers*).

This dynamic analysis phase that we have just described will help us to answer (almost) all the questions related to malware that were discussed at the beginning of this section. More advanced dynamic analysis would require using a debugger to analyze at runtime how CPU registers are evolving, the values returned by Windows functions called by the malware, and so on.

Finally, it should be noted that, **unlike the static analysis, in the dynamic analysis only one of the possible execution paths is analyzed**, which may also depend on the current execution conditions.

Once extracted, the process can be taken to analysis tools such as disassemblers or debuggers for analysis, as is the normal malware analysis process. As an example, Figure 1 shows the analysis of the process dumped in the tool `Ghidra`. In particular, the part of the code related to the creation of the mutex is shown.

