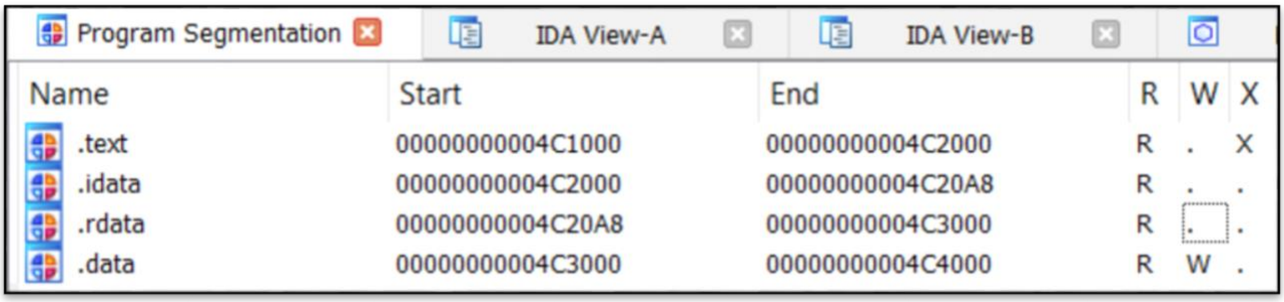
Lab 4 – Advance Topic in Malware Analysis

Reverse Engineering and IDA Pro

Yarden Curiel & Gal Azaria

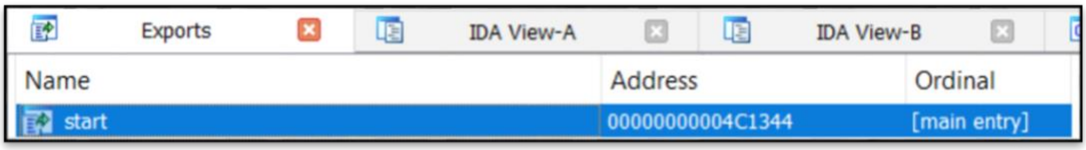
Exercise 1 – ex1.exe

1. List the section in the program:
   - Section: .**text**, Permission: Read and execute.
   - Section: .**idata**, Permission: Read.
   - Section: .**rdata**, Permission: Read.
   - Section: .**data**, Permission: Read and Write.



Name	Start	End	R	W	X
.text	00000000004C1000	00000000004C2000	R	.	X
.idata	00000000004C2000	00000000004C20A8	R	.	.
.rdata	00000000004C20A8	00000000004C3000	R	.	.
.data	00000000004C3000	00000000004C4000	R	W	.

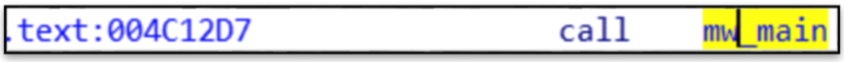
2. Entry Point locate at the ‘**004C1344**’ at the ‘.text’ section.



Name	Address	Ordinal
start	00000000004C1344	[main entry]

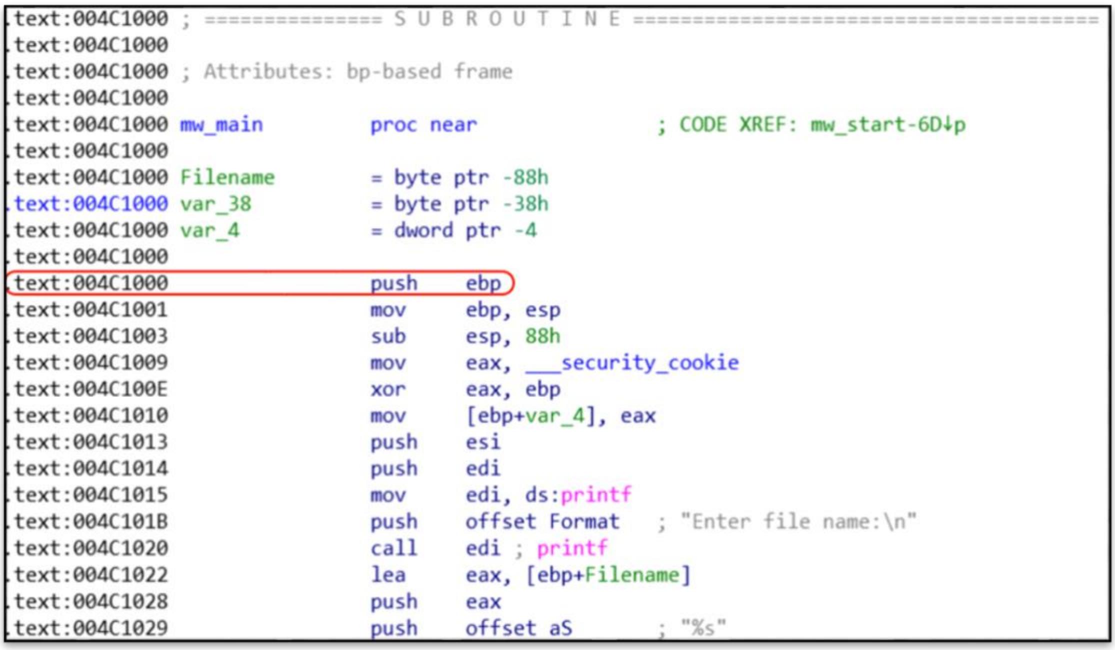
3. Entry Point locate at the ‘**004C1000**’ at the ‘.text’ section.

This is the call to ‘main’.



```
.text:004C12D7                 call    mw_main
```

This is the ‘main’.



```
.text:004C1000 ; =============== S U B R O U T I N E =======================================
.text:004C1000
.text:004C1000 ; Attributes: bp-based frame
.text:004C1000
.text:004C1000 mw_main         proc near               ; CODE XREF: mw_start-6D↓p
.text:004C1000
.text:004C1000 Filename        = byte ptr -88h
.text:004C1000 var_38          = byte ptr -38h
.text:004C1000 var_4           = dword ptr -4
.text:004C1000
.text:004C1000                 push    ebp
.text:004C1001                 mov     ebp, esp
.text:004C1003                 sub     esp, 88h
.text:004C1009                 mov     eax, ___security_cookie
.text:004C100E                 xor     eax, ebp
.text:004C1010                 mov     [ebp+var_4], eax
.text:004C1013                 push    esi
.text:004C1014                 push    edi
.text:004C1015                 mov     edi, ds:printf
.text:004C101B                 push    offset Format   ; "Enter file name:\n"
.text:004C1020                 call    edi ; printf
.text:004C1022                 lea     eax, [ebp+Filename]
.text:004C1028                 push    eax
.text:004C1029                 push    offset aS       ; "%s"
```

4. Detect and locate the bug:

   'main' parts explained:

- Start – '**mw_start**' - . Call the 'main' function.
- Main – '**mw_main**'. This section is the main function:
    i. It have some sort of canary (cookie create at the beginning and at the end compare to the origin one.
    ii. Print user "Enter file name:".
    iii. Wait for user's input.
    iv. Trying to load the given file with permission 'read' ('r').
    v. Check if the file exists or can be open with 'read' privilege (IF).
- Can't load – '**file_load_failed**'. This section is if the program couldn't load the file because it doesn't exists or it doesn't have the 'read' privilege. In this case the file will be close and "Can't open %s for reading.' Will be printed to the user the cookie will be check and the program will end.
- Empty file – '**Check_if_empty**'. This section will try to load text from file. If it seccuded it will keep to the next section ('**set_counter**'). Otherwise it will move to the '**close_file_and_exit_program**'.
- Start loop – '**set_counter**'. The section will set the ecx register (counter register) to the start of the file.
- Loop – '**loop_and_print**'. This section iterate each line and print it. Load next line and check if it's not empty, if so it will continue iterate again to the same section('**loop_and_print**'). If there is no more lines it will end by going to close section ('close_file_and_exit_program').
- Close – '**close_file_and_exit_program**'. In this case the file will be close and cookie will be check and the program will end.

   The bug is the locate:

   1. '**004C108B**' in '.text' section. This instruction is '**JNZ**'.
      This 'zero flag' should be zero if the compare one line above will tell if there is data inside the file, mean it's not empty.
   2. '**004C10AD**' in '.text' section. This instruction is '**JZ**'.
      This 'zero flag' should be one if the compare one line above will tell if there is still data inside the file, mean not all lines have been read.

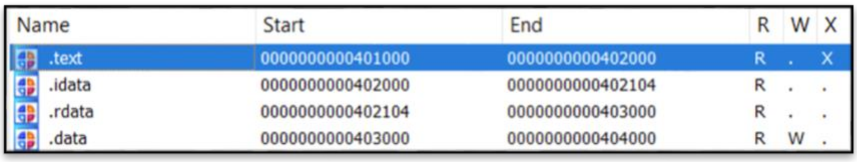
5. Fixing the bug will need to do the following:
- '**004C108B**' in '.text' section. Instruction is '**JNZ**' and should be '**JZ**'.
- '**004C10AD**' in '.text' section. Instruction is '**JZ**' and should be '**JNZ**'.

6. File Patched is attached.

7. Explanations:
- '**004C108B**' in '.text' section. This instruction is '**JNZ**'. This 'zero flag' should be zero if the compare one line above will tell if there is data inside the file, mean it's not empty.
- '**004C10AD**' in '.text' section. This instruction is '**JZ**'. This 'zero flag' should be one if the compare one line above will tell if there is still data inside the file, mean not all lines have been read.

Exercise 2 – ex2.exe

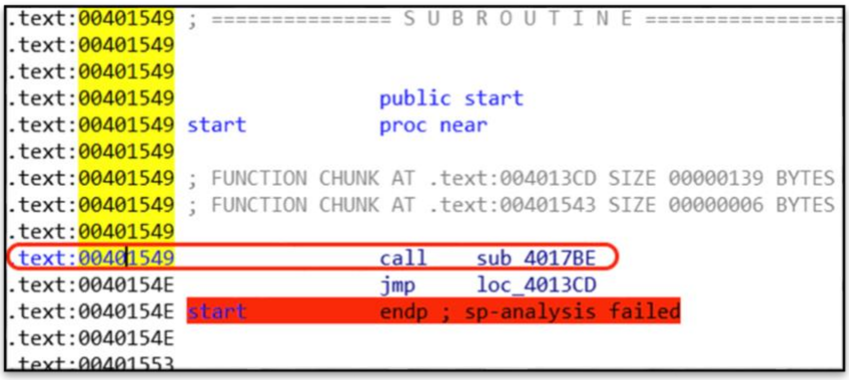
1. List the section in the program:
    a. Section: .**text**.
    b. Section: .**idata**.
    c. Section: .**rdata.**
    d. Section: .**data**.



Name	Start	End	R	W	X
.text	0000000000401000	0000000000402000	R	.	X
.idata	0000000000402000	0000000000402104	R	.	.
.rdata	0000000000402104	0000000000403000	R	.	.
.data	0000000000403000	0000000000404000	R	W	.

We can see that only the '**.text**' section have Execute Permission.

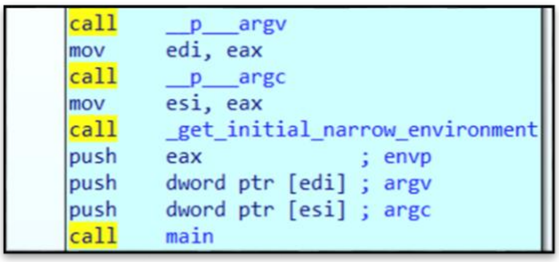
2. Entry Point locate at the '**00401549**' at the '.text' section (start).



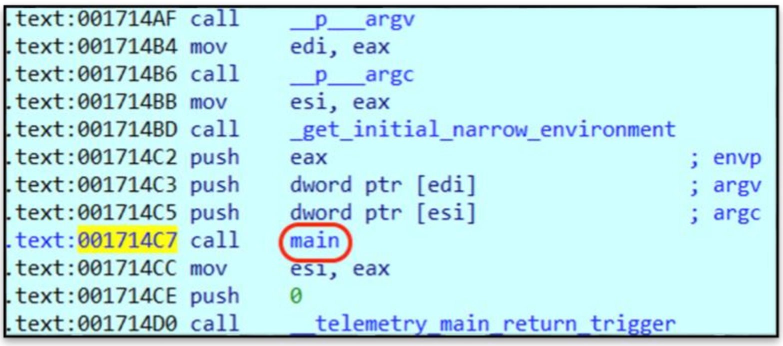
```
.text:00401549 ; =============== S U B R O U T I N E ================
.text:00401549
.text:00401549
.text:00401549                         public start
.text:00401549 start                   proc near
.text:00401549
.text:00401549 ; FUNCTION CHUNK AT .text:004013CD SIZE 00000139 BYTES
.text:00401549 ; FUNCTION CHUNK AT .text:00401543 SIZE 00000006 BYTES
.text:00401549
.text:00401549                         call    sub_4017BE
.text:0040154E                         jmp     loc_4013CD
.text:0040154E start                   endp ; sp-analysis failed
.text:0040154E
.text:00401553
```

3. Main locate at the '**001712B0**' at the '.text' section.

This is the call to 'main'.

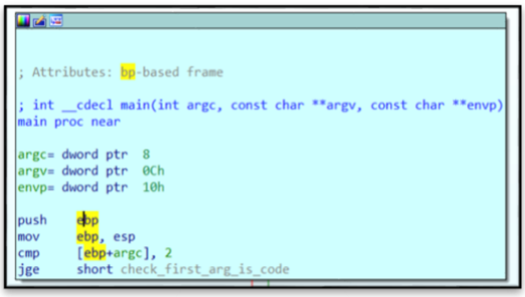


```
call    __p___argv
mov     edi, eax
call    __p___argc
mov     esi, eax
call    _get_initial_narrow_environment
push    eax             ; envp
push    dword ptr [edi] ; argv
push    dword ptr [esi] ; argc
call    main
```

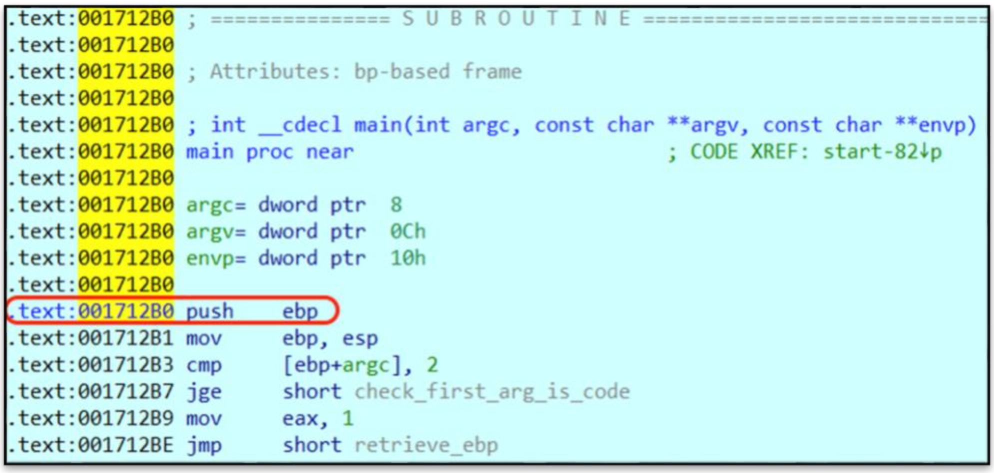


```
.text:001714AF call    __p___argv
.text:001714B4 mov     edi, eax
.text:001714B6 call    __p___argc
.text:001714BB mov     esi, eax
.text:001714BD call    _get_initial_narrow_environment
.text:001714C2 push    eax                                  ; envp
.text:001714C3 push    dword ptr [edi]                      ; argv
.text:001714C5 push    dword ptr [esi]                      ; argc
.text:001714C7 call    main
.text:001714CC mov     esi, eax
.text:001714CE push    0
.text:001714D0 call    __telemetry_main_return_trigger
```

This is the 'main'.

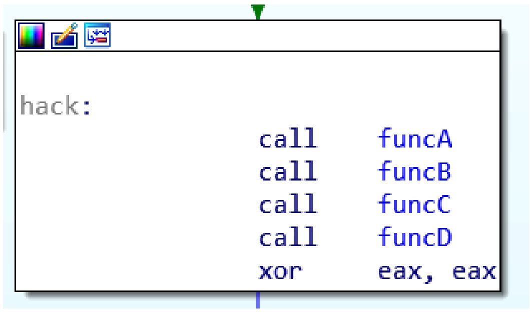


```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
main proc near

argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
cmp     [ebp+argc], 2
jge     short check_first_arg_is_code
```



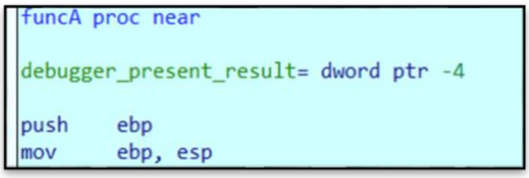
```
.text:001712B0 ; =============== S U B R O U T I N E ===========================
.text:001712B0
.text:001712B0 ; Attributes: bp-based frame
.text:001712B0
.text:001712B0 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:001712B0 main proc near                          ; CODE XREF: start-82↓p
.text:001712B0
.text:001712B0 argc= dword ptr  8
.text:001712B0 argv= dword ptr  0Ch
.text:001712B0 envp= dword ptr  10h
.text:001712B0
.text:001712B0 push    ebp
.text:001712B1 mov     ebp, esp
.text:001712B3 cmp     [ebp+argc], 2
.text:001712B7 jge     short check_first_arg_is_code
.text:001712B9 mov     eax, 1
.text:001712BE jmp     short retrieve_ebp
```

4. The program require one parameter in order to work. For the four function at the main to run the first parameter should be "-code".
5. Screen-shot.



```
hack:
                call    funcA
                call    funcB
                call    funcC
                call    funcD
                xor     eax, eax
```

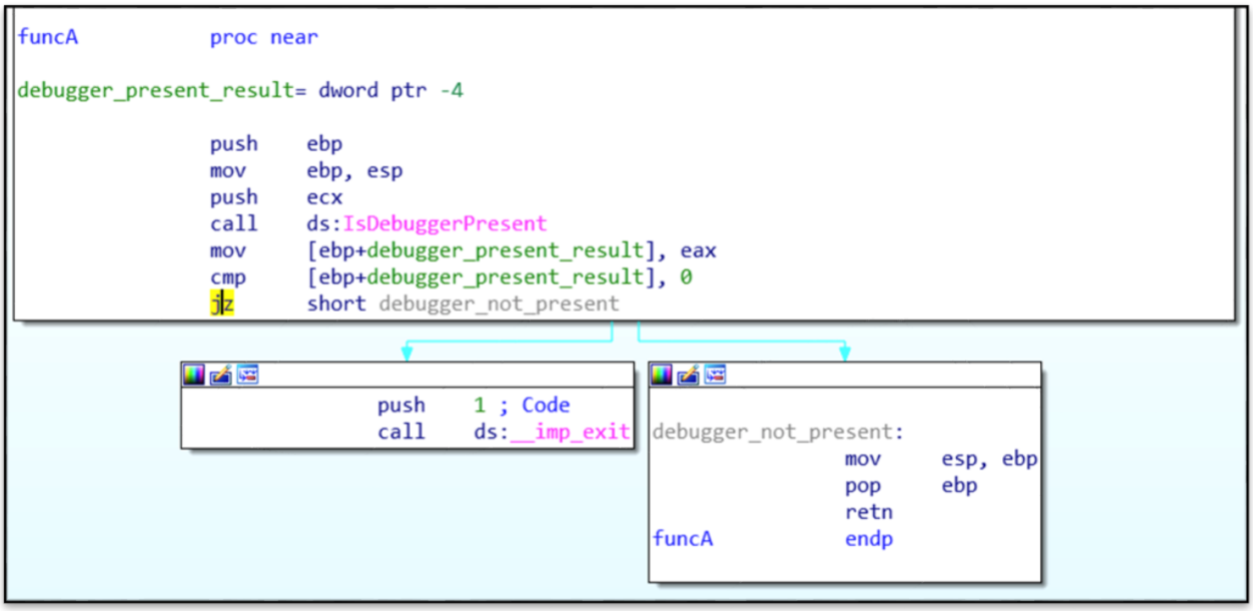
6. Analyze functions:
   a. funcA:
      i. Rename Parameters and Variables:



```
funcA proc near

debugger_present_result= dword ptr -4

push    ebp
mov     ebp, esp
```

      ii. List of system calls that use in the function:
         1. IsDebuggerPresent (kernel2.dll).

      iii. Function description:
      For our understanding, this function check if there is a debugger present by using the 'IsDebuggerPresent' function, if there is debugger running, the process will terminate.



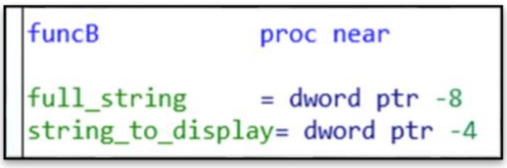
```
funcA           proc near

debugger_present_result= dword ptr -4

                push    ebp
                mov     ebp, esp
                push    ecx
                call    ds:IsDebuggerPresent
                mov     [ebp+debugger_present_result], eax
                cmp     [ebp+debugger_present_result], 0
                jz      short debugger_not_present
```

```
                push    1 ; Code
                call    ds:__imp_exit
```

```
debugger_not_present:
                mov     esp, ebp
                pop     ebp
                retn
funcA           endp
```

b. funcB:

i. Rename Parameters and Variables:



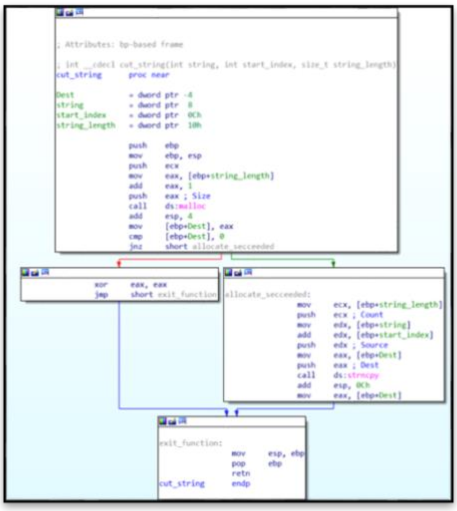
```
funcB           proc near

full_string     = dword ptr -8
string_to_display= dword ptr -4
```

Function:

1. cut_string(int string, int start_index, size_t string_length) – this function get string and return only sub-string. using the start_index is start in this index and cut the next 'string_length' chars.

ii. List of system calls that use in the function:

1. malloc (api-ms-win-crt-heap-l1-1-0.dll).
2. strncpy (api-ms-win-crt-string-l1-1-0.dll).
3. OutputDebugStringA (kernel32.dll).
4. free (api-ms-win-crt-heap-l1-1-0.dll).



```
; Attributes: bp-based frame

; int __cdecl cut_string(int string, int start_index, size_t string_length)
cut_string      proc near

Dest            = dword ptr -4
string          = dword ptr  8
start_index     = dword ptr  0Ch
string_length   = dword ptr  10h

                push    ebp
                mov     ebp, esp
                push    ecx
                mov     eax, [ebp+string_length]
                add     eax, 1
                push    eax ; Size
                call    ds:malloc
                add     esp, 4
                mov     [ebp+Dest], eax
                cmp     [ebp+Dest], 0
                jnz     short allocate_secceeded
```

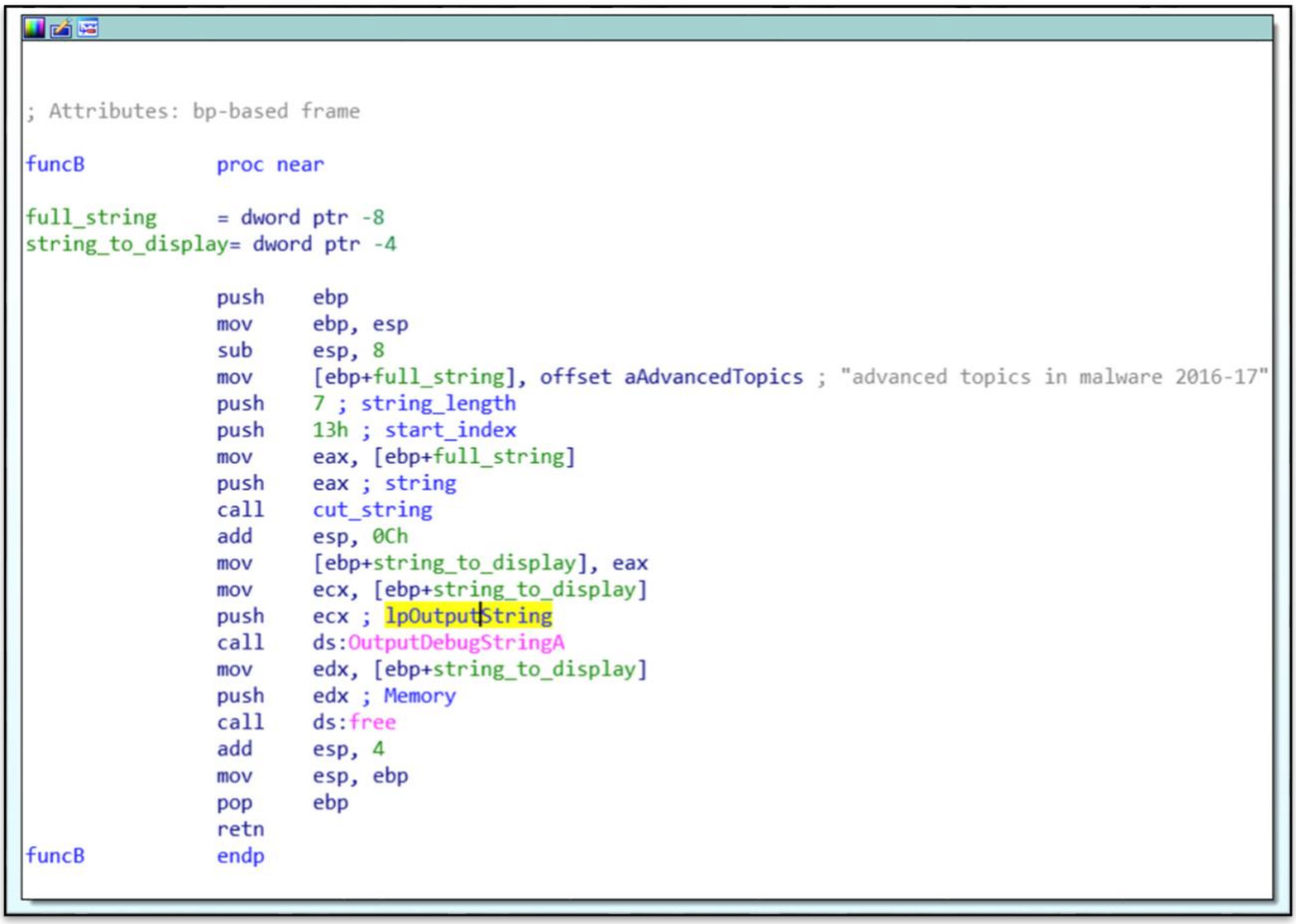
```
                xor     eax, eax
                jmp     short exit_function
```

```
allocate_secceeded:
                mov     ecx, [ebp+string_length]
                push    ecx ; Count
                mov     edx, [ebp+string]
                add     edx, [ebp+start_index]
                push    edx ; Source
                mov     eax, [ebp+Dest]
                push    eax ; Dest
                call    ds:strncpy
                add     esp, 0Ch
                mov     eax, [ebp+Dest]
```

```
exit_function:
                mov     esp, ebp
                pop     ebp
                retn
cut_string      endp
```

iii. Function description:

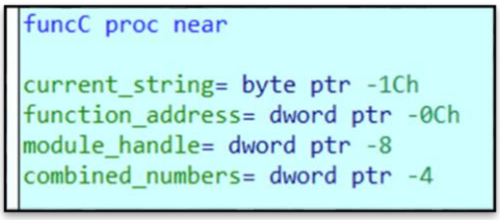
For our understanding, this function use existing string in the data section "advance topic in malware 2016-17", from this string it use "cut_string" function to sub string (with these three arguments: The string ("advance topic in malware 2016-17",), start_index (19, the index to start copy the given string) and string_length (7, length to take from the passed string)) and pass it as a parameter to "OutputDebugStringA" function which print the sub-string to the debug output window, and finally, it free the allocated space of the string.



```
; Attributes: bp-based frame

funcB           proc near

full_string     = dword ptr -8
string_to_display= dword ptr -4

                push    ebp
                mov     ebp, esp
                sub     esp, 8
                mov     [ebp+full_string], offset aAdvancedTopics ; "advanced topics in malware 2016-17"
                push    7 ; string_length
                push    13h ; start_index
                mov     eax, [ebp+full_string]
                push    eax ; string
                call    cut_string
                add     esp, 0Ch
                mov     [ebp+string_to_display], eax
                mov     ecx, [ebp+string_to_display]
                push    ecx ; lpOutputString
                call    ds:OutputDebugStringA
                mov     edx, [ebp+string_to_display]
                push    edx ; Memory
                call    ds:free
                add     esp, 4
                mov     esp, ebp
                pop     ebp
                retn
funcB           endp
```

c. funcC:

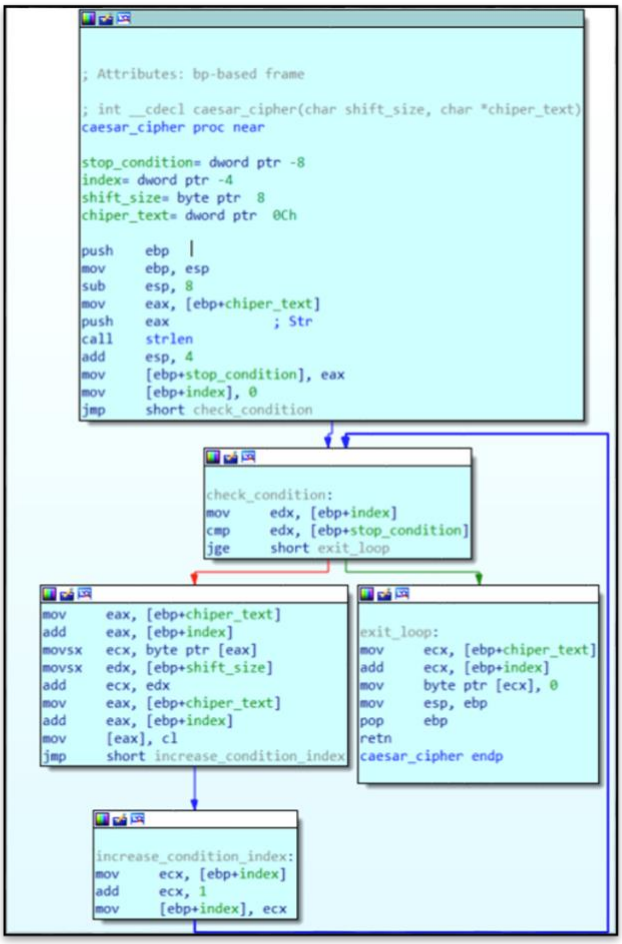
i. Rename Parameters and Variables:



```
funcC proc near

current_string= byte ptr -1Ch
function_address= dword ptr -0Ch
module_handle= dword ptr -8
combined_numbers= dword ptr -4
```

Function:

1. **ceaser_chiper**(char shift_size, char *chipper_text) – this function get string and 'shift_size', for each char in the given string it raise t value by 'shift_size', and return the new string.



```
; Attributes: bp-based frame

; int __cdecl caesar_cipher(char shift_size, char *chiper_text)
caesar_cipher proc near

stop_condition= dword ptr -8
index= dword ptr -4
shift_size= byte ptr  8
chiper_text= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+chiper_text]
push    eax             ; Str
call    strlen
add     esp, 4
mov     [ebp+stop_condition], eax
mov     [ebp+index], 0
jmp     short check_condition
```

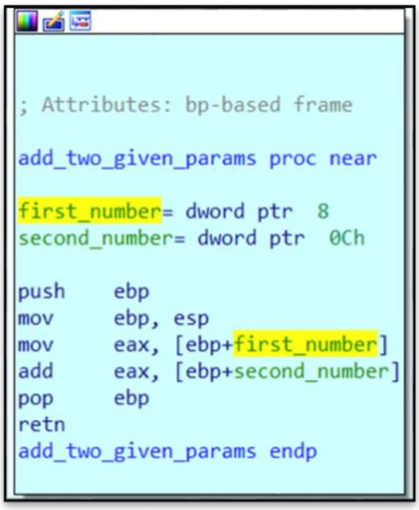
```
check_condition:
mov     edx, [ebp+index]
cmp     edx, [ebp+stop_condition]
jge     short exit_loop
```

```
mov     eax, [ebp+chiper_text]
add     eax, [ebp+index]
movsx   ecx, byte ptr [eax]
movsx   edx, [ebp+shift_size]
add     ecx, edx
mov     eax, [ebp+chiper_text]
add     eax, [ebp+index]
mov     [eax], cl
jmp     short increase_condition_index
```

```
exit_loop:
mov     ecx, [ebp+chiper_text]
add     ecx, [ebp+index]
mov     byte ptr [ecx], 0
mov     esp, ebp
pop     ebp
retn
caesar_cipher endp
```

```
increase_condition_index:
mov     ecx, [ebp+index]
add     ecx, 1
mov     [ebp+index], ecx
```

2. **Add_two_given_params**(int first_number, int second_number) – this function get two number as argument and return the sum of these numbers.



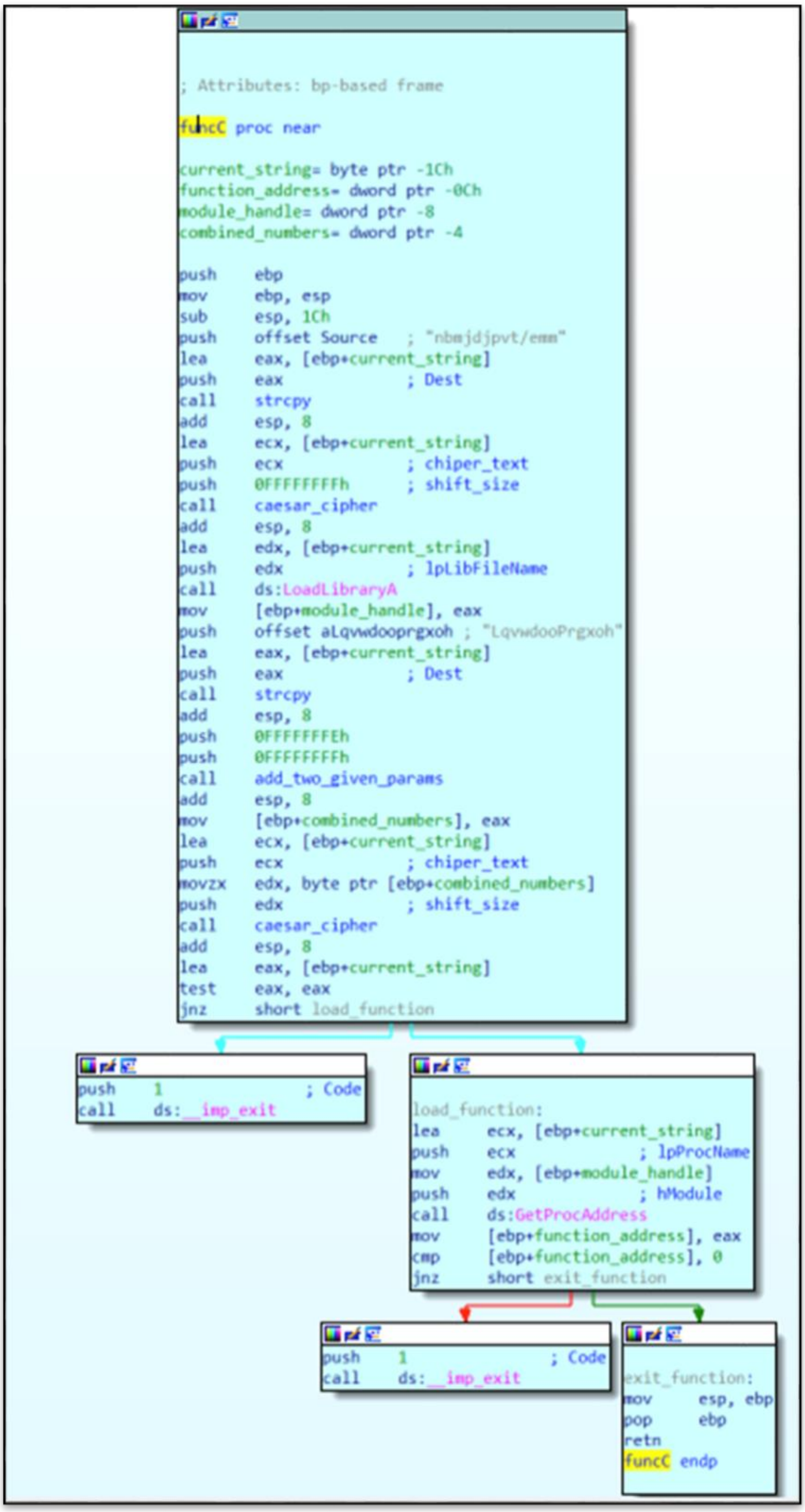
```
; Attributes: bp-based frame

add_two_given_params proc near

first_number= dword ptr  8
second_number= dword ptr  0Ch

push    ebp
mov     ebp, esp
mov     eax, [ebp+first_number]
add     eax, [ebp+second_number]
pop     ebp
retn
add_two_given_params endp
```

ii. List of system calls that use in the function:

1. strncpy (api-ms-win-crt-string-l1-1-0.dll).
2. strlen (api-ms-win-crt-string-l1-1-0.dll).
3. LoadLibraryA (kernel32.dll).
4. GetProcAddress (kernel32.dll).

iii. Function description:

For our understanding, this function load a existing string from the data section, the string is “nbmjdjpvt/emm”. Using ‘ceasar_chiper’ on this string and shift ‘-1’ we get the string: ‘malicious.dll’. using ‘LoadLibraryA’ function we load the ‘malicious.dll’ to the process. Next, the function load from data section the string “LqvwdooPrgxoh” and by using the ‘add_two_given_params with ‘-1’ and ‘-2’ it return ‘-3’ and the string and the number ‘-3’ using the ‘ceasar_chiper’ to return string: ‘InstallModule’. After checking the returned string do exists, the process trying to load the ‘InstallModule’ function from the “malicious.dll” handleusing ‘GetProcAddress’. If it secceed, it continue, otherwise it quit the system with error 1.



```
; Attributes: bp-based frame

funcC proc near

current_string= byte ptr -1Ch
function_address= dword ptr -0Ch
module_handle= dword ptr -8
combined_numbers= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 1Ch
push    offset Source   ; "nbmjdjpvt/emm"
lea     eax, [ebp+current_string]
push    eax             ; Dest
call    strcpy
add     esp, 8
lea     ecx, [ebp+current_string]
push    ecx             ; chiper_text
push    0FFFFFFFFh      ; shift_size
call    caesar_cipher
add     esp, 8
lea     edx, [ebp+current_string]
push    edx             ; lpLibFileName
call    ds:LoadLibraryA
mov     [ebp+module_handle], eax
push    offset aLqvwdooprgxoh ; "LqvwdooPrgxoh"
lea     eax, [ebp+current_string]
push    eax             ; Dest
call    strcpy
add     esp, 8
push    0FFFFFFFEh
push    0FFFFFFFFh
call    add_two_given_params
add     esp, 8
mov     [ebp+combined_numbers], eax
lea     ecx, [ebp+current_string]
push    ecx             ; chiper_text
movzx   edx, byte ptr [ebp+combined_numbers]
push    edx             ; shift_size
call    caesar_cipher
add     esp, 8
lea     eax, [ebp+current_string]
test    eax, eax
jnz     short load_function
```

```
push    1               ; Code
call    ds:__imp_exit
```

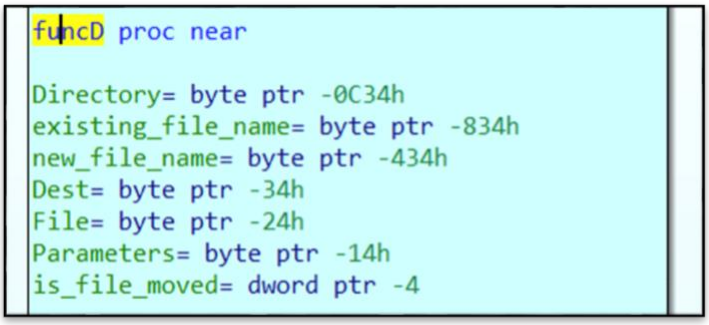
```
load_function:
lea     ecx, [ebp+current_string]
push    ecx             ; lpProcName
mov     edx, [ebp+module_handle]
push    edx             ; hModule
call    ds:GetProcAddress
mov     [ebp+function_address], eax
cmp     [ebp+function_address], 0
jnz     short exit_function
```

```
push    1               ; Code
call    ds:__imp_exit
```

```
exit_function:
mov     esp, ebp
pop     ebp
retn
funcC endp
```

d. funcD:

i. Rename Parameters and Variables:



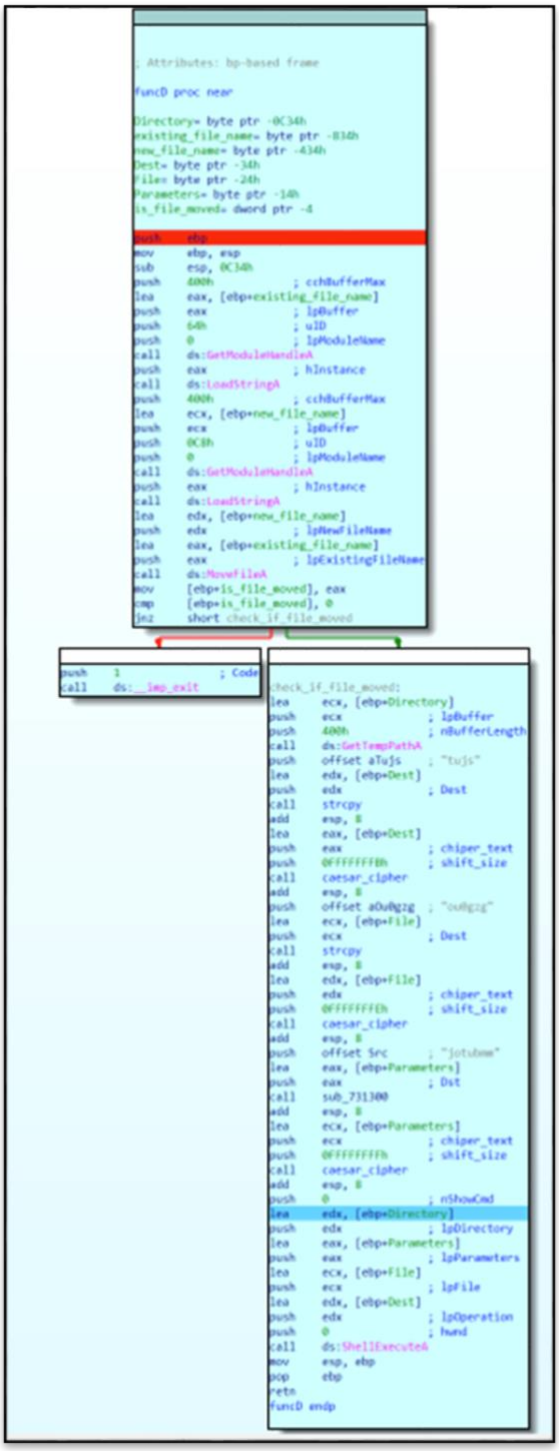
```
funcD proc near

Directory= byte ptr -0C34h
existing_file_name= byte ptr -834h
new_file_name= byte ptr -434h
Dest= byte ptr -34h
File= byte ptr -24h
Parameters= byte ptr -14h
is_file_moved= dword ptr -4
```

ii. List of system calls that use in the function:

1. **GetModuleHandleA** ().
2. **LoadLibrary** ().
3. **LoadStringA** ().
4. **MoveFileA** ().
5. **GetTempPathA** ().
6. **ShellExecuteA** ().

iii. Function description:

For our understanding, this function first trying to load module from the buffer using the 'GetModuleHandeA' function, them loading existing string using 'LoadStringA' function and load 'icon1.jpg' string. This opperatin preform again by this process to load different string '"ms.exe" as well in the same method and move these file using 'MoveFileA' function, if operation didn't secceeded exit with code 1, if it did succeeded, the function continue and load the temporary computer folder (C:\Users\ISE\AppData\Local\Temp\) using 'GetTempPathA', then it load the string: "tujs" from the data section, using 'ceasar_chiper' and the shift '-5' it create the word 'open', the same is with the string: 'ou0gzg' and the shift '-2' and then the word 'ms.exe' created, and again with the string 'jotubmm' with the shift '-1' to reavel the word 'install'. Than the function 'ShellExecuteA' called with the parameters: showcmd=0, LpDirectory= 'C:\Users\ISE\AppData\Local\Temp\', LpParameters='install', LpFile='ms'exe',mean it run the 'ms.exe' in Temp folder without printing it to the console and exit the function.



```
; Attributes: bp-based frame

funcD proc near

Directory= byte ptr -0C34h
existing_file_name= byte ptr -834h
new_file_name= byte ptr -434h
Dest= byte ptr -34h
File= byte ptr -24h
Parameters= byte ptr -14h
is_file_moved= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 0C34h
push    400h                ; cchBufferMax
lea     eax, [ebp+existing_file_name]
push    eax                 ; lpBuffer
push    64h                 ; uID
push    0                   ; lpModuleName
call    ds:GetModuleHandleA
push    eax                 ; hInstance
call    ds:LoadStringA
push    400h                ; cchBufferMax
lea     ecx, [ebp+new_file_name]
push    ecx                 ; lpBuffer
push    0C8h                ; uID
push    0                   ; lpModuleName
call    ds:GetModuleHandleA
push    eax                 ; hInstance
call    ds:LoadStringA
lea     edx, [ebp+new_file_name]
push    edx                 ; lpNewFileName
lea     eax, [ebp+existing_file_name]
push    eax                 ; lpExistingFileName
call    ds:MoveFileA
mov     [ebp+is_file_moved], eax
cmp     [ebp+is_file_moved], 0
jnz     short check_if_file_moved
```

```
push    1                   ; Code
call    ds:__imp_exit
```

```
check_if_file_moved:
lea     ecx, [ebp+Directory]
push    ecx                 ; lpBuffer
push    400h                ; nBufferLength
call    ds:GetTempPathA
push    offset aTujs        ; "tujs"
lea     edx, [ebp+Dest]
push    edx                 ; Dest
call    strcpy
add     esp, 8
lea     eax, [ebp+Dest]
push    eax                 ; chiper_text
push    0FFFFFFFBh          ; shift_size
call    caesar_cipher
add     esp, 8
push    offset aOu0gzg      ; "ou0gzg"
lea     ecx, [ebp+File]
push    ecx                 ; Dest
call    strcpy
add     esp, 8
lea     edx, [ebp+File]
push    edx                 ; chiper_text
push    0FFFFFFFEh          ; shift_size
call    caesar_cipher
add     esp, 8
push    offset Src          ; "jotubmm"
lea     eax, [ebp+Parameters]
push    eax                 ; Dst
call    sub_731300
add     esp, 8
lea     ecx, [ebp+Parameters]
push    ecx                 ; chiper_text
push    0FFFFFFFFh          ; shift_size
call    caesar_cipher
add     esp, 8
push    0                   ; nShowCmd
lea     edx, [ebp+Directory]
push    edx                 ; lpDirectory
lea     eax, [ebp+Parameters]
push    eax                 ; lpParameters
lea     ecx, [ebp+File]
push    ecx                 ; lpFile
lea     edx, [ebp+Dest]
push    edx                 ; lpOperation
push    0                   ; hwnd
call    ds:ShellExecuteA
mov     esp, ebp
pop     ebp
retn
funcD endp
```

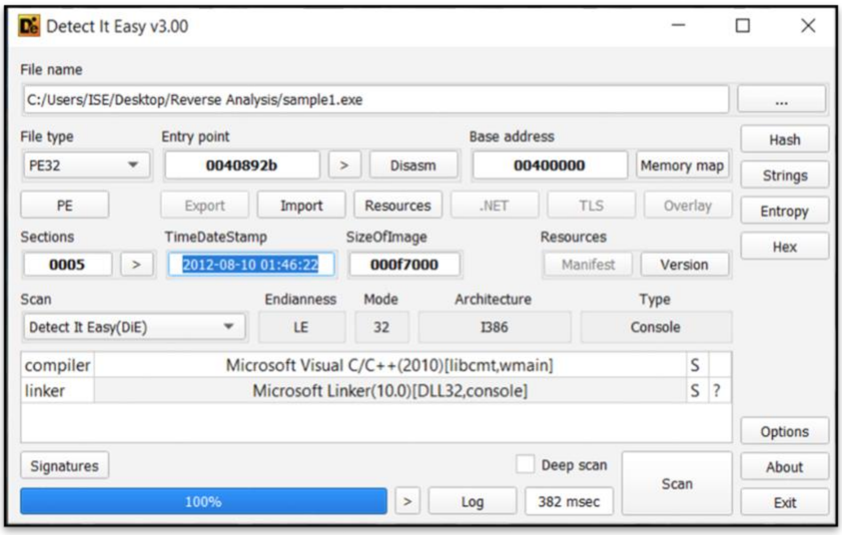
Exercise 3:

Sample 1:

We started our investigation by using static tools analysis.

Detect It Easy:

We can see that the complier is “Microsoft Visual C/C++(2010)” and the linker is “Microsoft Linker(10.0)”. The File time stamp is: “2012-08-10 01:46:22”.



File name: C:/Users/ISE/Desktop/Reverse Analysis/sample1.exe

File type	Entry point	Base address
PE32	0040892b	00400000

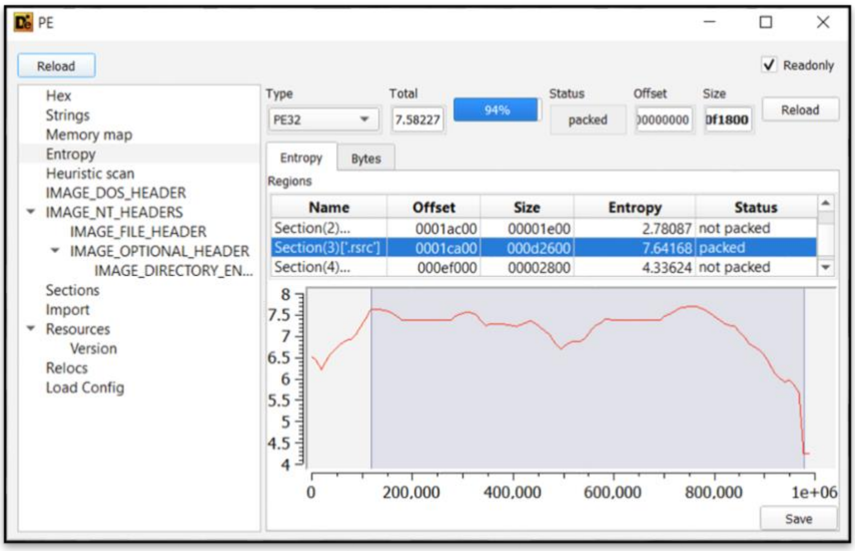
Sections	TimeDateStamp	SizeOfImage
0005	2012-08-10 01:46:22	000f7000

Scan	Endianness	Mode	Architecture	Type
Detect It Easy(DiE)	LE	32	I386	Console

compiler	Microsoft Visual C/C++(2010)[libcmt,wmain]	S	
linker	Microsoft Linker(10.0)[DLL32,console]	S	?

100% 382 msec

We can see here that the ‘.rsrc’ section is packet by the high entropy.

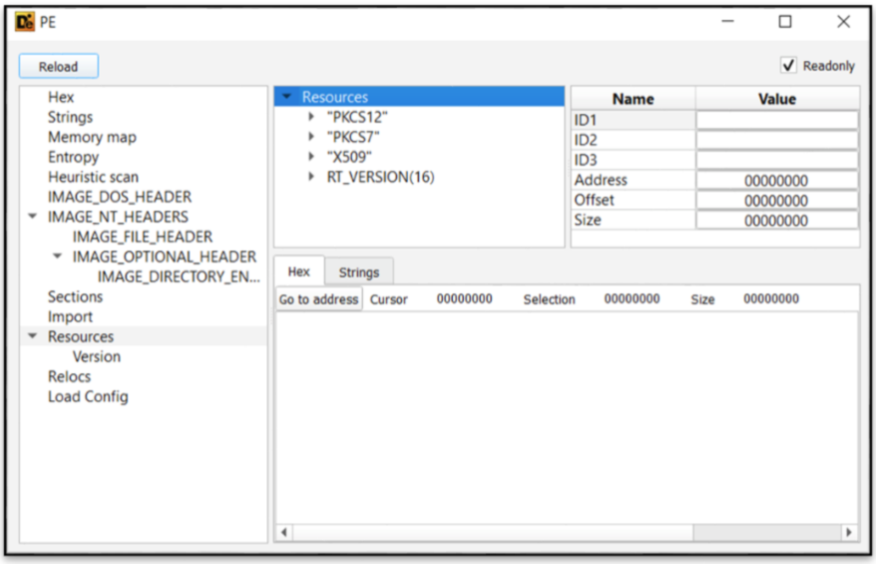


Type	Total		Status	Offset	Size
PE32	7.58227	94%	packed	)0000000	0f1800

Regions

Name	Offset	Size	Entropy	Status
Section(2)...	0001ac00	00001e00	2.78087	not packed
Section(3)['.rsrc']	0001ca00	000d2600	7.64168	packed
Section(4)...	000ef000	00002800	4.33624	not packed

By looking on the file resources we can see that it have 4 resources:

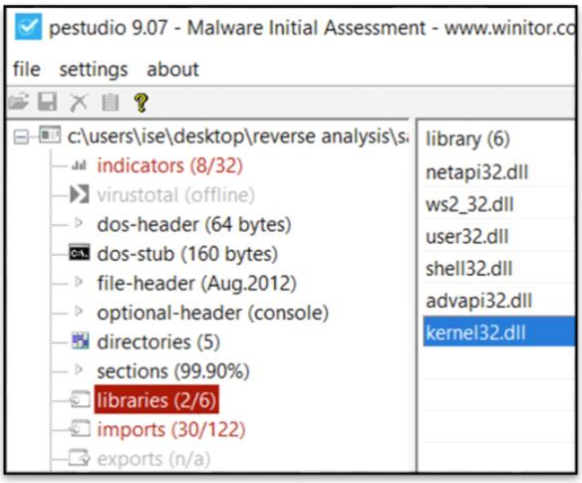


Resources
- "PKCS12"
- "PKCS7"
- "X509"
- RT_VERSION(16)

Name	Value
ID1	
ID2	
ID3	
Address	00000000
Offset	00000000
Size	00000000

Cursor 00000000 Selection 00000000 Size 00000000

PEstudio:



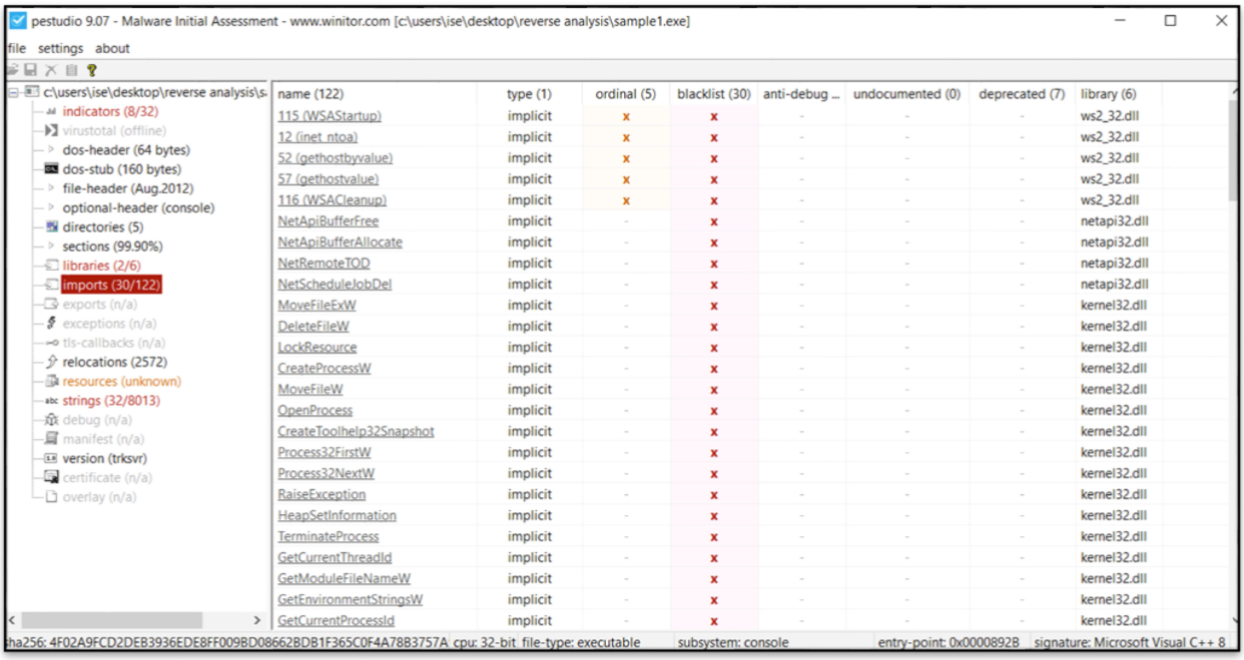
library (6)
netapi32.dll
ws2_32.dll
user32.dll
shell32.dll
advapi32.dll
kernel32.dll

By looking the imports we can see that it use:

a. **ws2_32.dll** – use for create socket, so that can be indicate that this PE may create internet connection.

After investigating in which DLL the PE file declare to use, we investigate the function it use. We can see some odds function import also here:
All the 'ws2_32.dll' that the file is using it use ordinals and not function names, this can be suspicious. Couple of suspicious function for 'kernel32.dll" are:

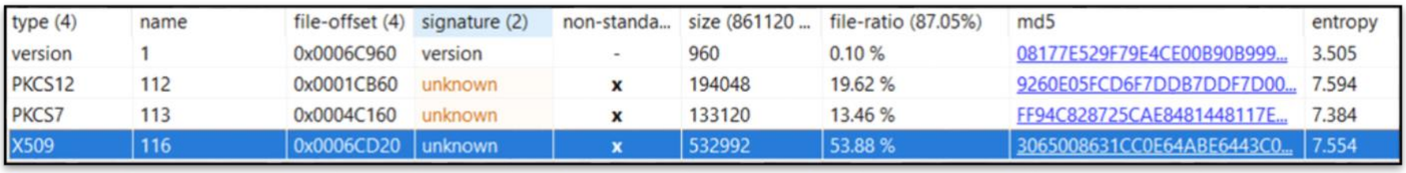
a. **MoveFileExW and DeleteFileW, MoveFileW** – This can be used to manipulate files.
b. **CreateProcess, OpenProcess, Process32FirstW, Process32NextW, TerminateProcess, GetCurrentThreadId, GetCurrentProcessId and CreateService** – These function can be used to create process, iterate process and Terminate others.
c. **RegDeleteValueW** – can be used to delete keys from registry.



name (122)	type (1)	ordinal (5)	blacklist (30)	anti-debug ...	undocumented (0)	deprecated (7)	library (6)
115 (WSAStartup)	implicit	x	x	-	-	-	ws2_32.dll
12 (inet_ntoa)	implicit	x	x	-	-	-	ws2_32.dll
52 (gethostbyvalue)	implicit	x	x	-	-	-	ws2_32.dll
57 (gethostvalue)	implicit	x	x	-	-	-	ws2_32.dll
116 (WSACleanup)	implicit	x	x	-	-	-	ws2_32.dll
NetApiBufferFree	implicit	-	x	-	-	-	netapi32.dll
NetApiBufferAllocate	implicit	-	x	-	-	-	netapi32.dll
NetRemoteTOD	implicit	-	x	-	-	-	netapi32.dll
NetScheduleJobDel	implicit	-	x	-	-	-	netapi32.dll
MoveFileExW	implicit	-	x	-	-	-	kernel32.dll
DeleteFileW	implicit	-	x	-	-	-	kernel32.dll
LockResource	implicit	-	x	-	-	-	kernel32.dll
CreateProcessW	implicit	-	x	-	-	-	kernel32.dll
MoveFileW	implicit	-	x	-	-	-	kernel32.dll
OpenProcess	implicit	-	x	-	-	-	kernel32.dll
CreateToolhelp32Snapshot	implicit	-	x	-	-	-	kernel32.dll
Process32FirstW	implicit	-	x	-	-	-	kernel32.dll
Process32NextW	implicit	-	x	-	-	-	kernel32.dll
RaiseException	implicit	-	x	-	-	-	kernel32.dll
HeapSetInformation	implicit	-	x	-	-	-	kernel32.dll
TerminateProcess	implicit	-	x	-	-	-	kernel32.dll
GetCurrentThreadId	implicit	-	x	-	-	-	kernel32.dll
GetModuleFileNameW	implicit	-	x	-	-	-	kernel32.dll
GetEnvironmentStringsW	implicit	-	x	-	-	-	kernel32.dll
GetCurrentProcessId	implicit	-	x	-	-	-	kernel32.dll

ha256: 4F02A9FCD2DEB3936EDE8FF009BD08662BDB1F365C0F4A78B3757A cpu: 32-bit file-type: executable subsystem: console entry-point: 0x0000892B signature: Microsoft Visual C++ 8

By looking at the resource entropy we can see clearly that they are obfuscated or packet.



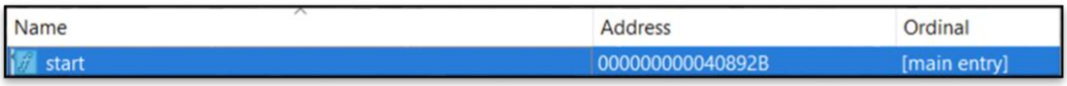
type (4)	name	file-offset (4)	signature (2)	non-standa...	size (861120 ...	file-ratio (87.05%)	md5	entropy
version	1	0x0006C960	version	-	960	0.10 %	08177E529F79E4CE00B90B999...	3.505
PKCS12	112	0x0001CB60	unknown	x	194048	19.62 %	9260E05FCD6F7DDB7DDF7D00...	7.594
PKCS7	113	0x0004C160	unknown	x	133120	13.46 %	FF94C828725CAE8481448117E...	7.384
X509	116	0x0006CD20	unknown	x	532992	53.88 %	3065008631CC0E64ABE6443C0...	7.554

Strings:

Couple of string that we found that seems strange to us:

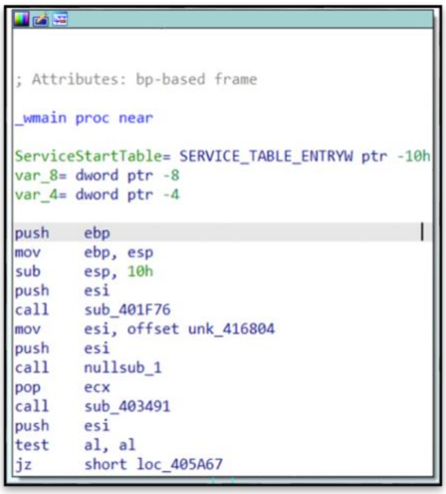
- “C:\Windows\system32\svchost.exe -k netsvcs” – can indicate of usage of naïve process to create internet connection.
- “c:\windows\temp\out17626867.txt” – can indicate of the PE file way of action, that it will create this file.
- “Copyright (c) 1992-2004 by P.J. Plauger, licensed by Dinkumware, Ltd. ALL RIGHTS RESERVED.” – P.J Plauger was author that wrote a C manual book, maybe it have some clue or misdirection.
- “AKERNEL32.DLL” – this look like kernel32.dll with ‘a’ at the beginning, could be misleading.
- “C$\WINDOWS” and “D$\WINDOWS” and “E$\WINDOWS “ – we can see clearly that the windows directory is use in this PE file.
- “Distributed Link Tracking Server” – This PE file have something with Tracking server.
- “\System32\cmd.exe /c "ping -n 30 127.0.0.1 >nul && sc config TrkSvr binpath= system32\trksrv.exe && ping -n 10 127.0.0.1 >nul && sc start TrkSvr "” – this command found without the attempt to hide it. Seems that there is attempt to ping a localhost machine in some sort.
- “trksrv.exe” – here we can see the above file that use in the ping command.
- “LoadLibrary” – may indicate of dynamically load libraries.
- “LoadResource” and “LockResource” – may indicate the attempt of the PE file to lock it’s resources to avoid external access.
- “RegCloseKey”, “RegDeleteValue”, “RefOpenKeyEx” and “RegQueryValueEx” – are not appeare in the function that the PE file imported, this may also indicate of a dynamically loading of a functions.

- Entry Point locate at the ‘**0040892B**’ at the ‘.text’ section.



Name	Address	Ordinal
start	000000000040892B	[main entry]

- Main locate at ‘**00405A38**’:

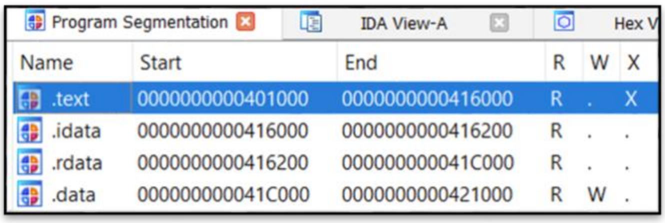


```
; Attributes: bp-based frame

_wmain proc near

ServiceStartTable= SERVICE_TABLE_ENTRYW ptr -10h
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 10h
push    esi
call    sub_401F76
mov     esi, offset unk_416804
push    esi
call    nullsub_1
pop     ecx
call    sub_403491
push    esi
test    al, al
jz      short loc_405A67
```

- As we saw earlier at the static analysis stage, the compiler is: “Microsoft Visual C/C++(2010)” and the linker is “Microsoft Linker(10.0)”.

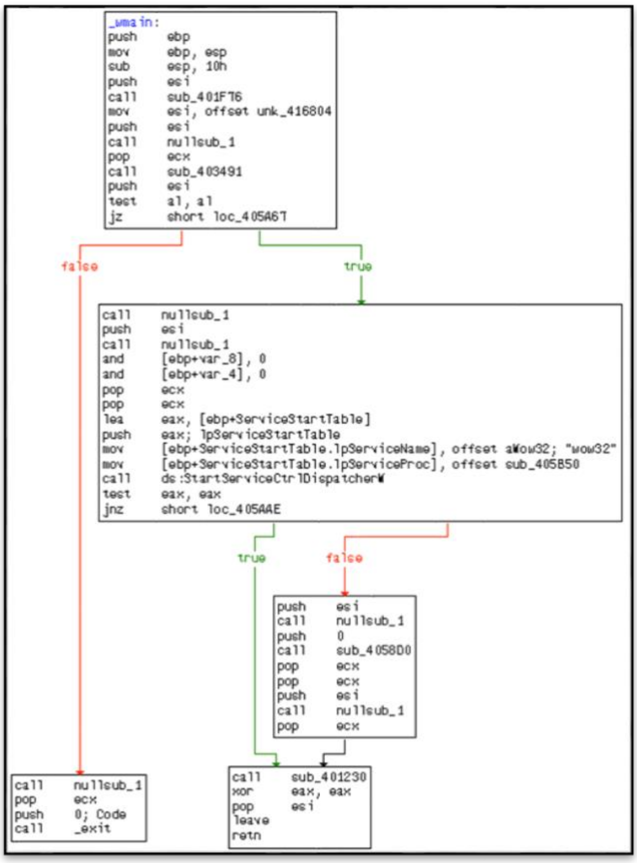
8. List the section in the program:

- Section: .**text**, Permission: Read and execute.
- Section: .**idata**, Permission: Read.
- Section: .**rdata**, Permission: Read.
- Section: .**data**, Permission: Read and Write.



Name	Start	End	R	W	X
.text	0000000000401000	0000000000416000	R	.	X
.idata	0000000000416000	0000000000416200	R	.	.
.rdata	0000000000416200	000000000041C000	R	.	.
.data	000000000041C000	0000000000421000	R	W	.

- Call flow graph (main):



```
_wmain:
push    ebp
mov     ebp, esp
sub     esp, 10h
push    esi
call    sub_401F76
mov     esi, offset unk_416804
push    esi
call    nullsub_1
pop     ecx
call    sub_403491
push    esi
test    al, al
jz      short loc_405A67
```

```
call    nullsub_1
push    esi
call    nullsub_1
and     [ebp+var_8], 0
and     [ebp+var_4], 0
pop     ecx
pop     ecx
lea     eax, [ebp+ServiceStartTable]
push    eax; lpServiceStartTable
mov     [ebp+ServiceStartTable.lpServiceName], offset aWow32; "wow32"
mov     [ebp+ServiceStartTable.lpServiceProc], offset sub_405B50
call    ds:StartServiceCtrlDispatcherW
test    eax, eax
jnz     short loc_405AAE
```

```
push    esi
call    nullsub_1
push    0
call    sub_4058D0
pop     ecx
pop     ecx
push    esi
call    nullsub_1
pop     ecx
```

```
call    nullsub_1
pop     ecx
push    0; Code
call    _exit
```

```
call    sub_401230
xor     eax, eax
pop     esi
leave
retn
```

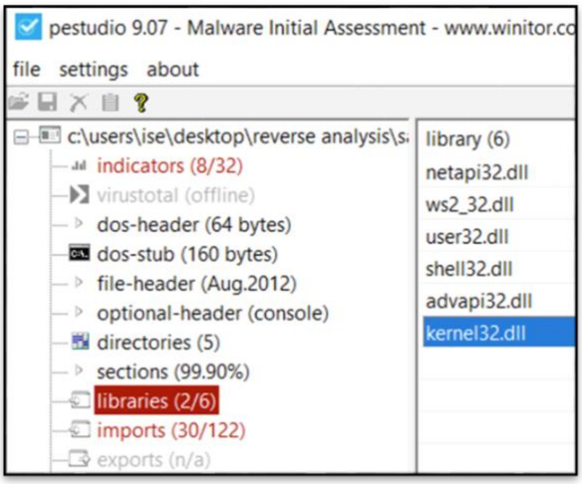
- Analyze ‘sample_1’ using IDA:

The First approach we thought it smart to understand the sample it to try debug the main, and try to get hint of function and sample flow, to understand what the sample trying to do.

After the initial run we already notice that the import that the sample loaded is different than the one we saw it declared earlier at the static analysis stage.

PEstudio – Static Analysis

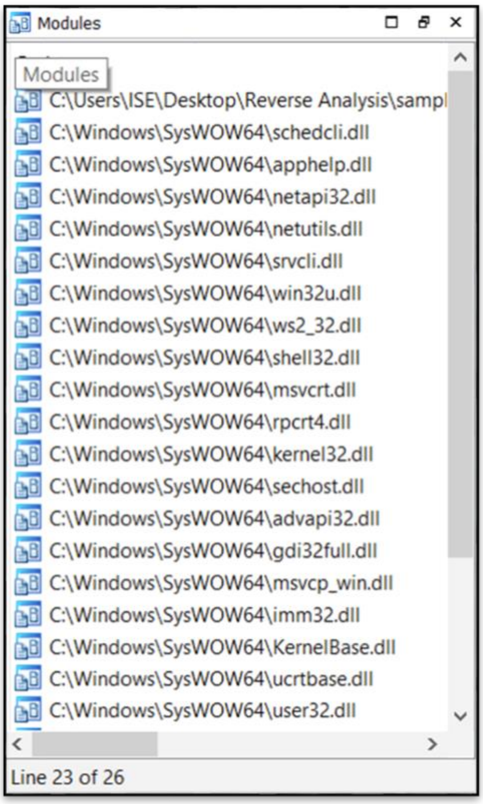
IDA – Reverse Analysis



pestudio 9.07 - Malware Initial Assessment - www.winitor.co

```
c:\users\ise\desktop\reverse analysis\s
    indicators (8/32)
    virustotal (offline)
    dos-header (64 bytes)
    dos-stub (160 bytes)
    file-header (Aug.2012)
    optional-header (console)
    directories (5)
    sections (99.90%)
    libraries (2/6)
    imports (30/122)
    exports (n/a)
```

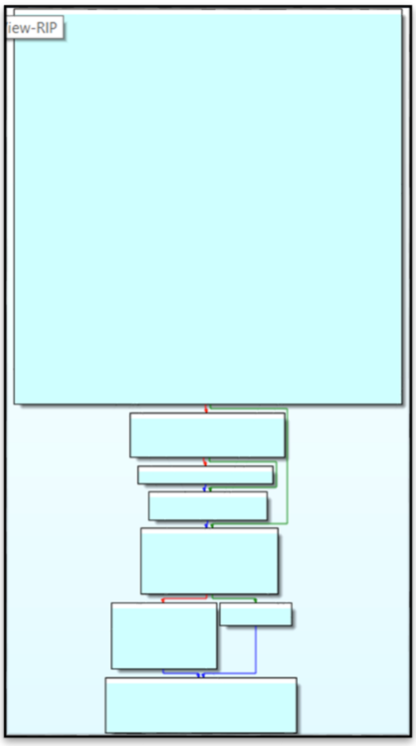
library (6)
netapi32.dll
ws2_32.dll
user32.dll
shell32.dll
advapi32.dll
kernel32.dll



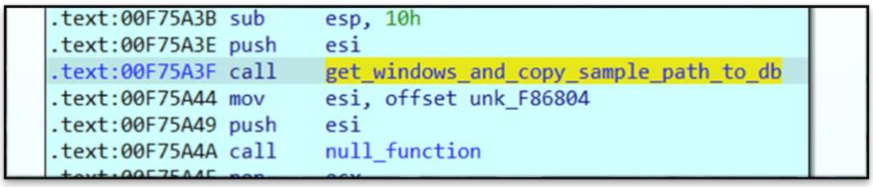
Modules

```
Modules
C:\Users\ISE\Desktop\Reverse Analysis\sampl
C:\Windows\SysWOW64\schedcli.dll
C:\Windows\SysWOW64\apphelp.dll
C:\Windows\SysWOW64\netapi32.dll
C:\Windows\SysWOW64\netutils.dll
C:\Windows\SysWOW64\srvcli.dll
C:\Windows\SysWOW64\win32u.dll
C:\Windows\SysWOW64\ws2_32.dll
C:\Windows\SysWOW64\shell32.dll
C:\Windows\SysWOW64\msvcrt.dll
C:\Windows\SysWOW64\rpcrt4.dll
C:\Windows\SysWOW64\kernel32.dll
C:\Windows\SysWOW64\sechost.dll
C:\Windows\SysWOW64\advapi32.dll
C:\Windows\SysWOW64\gdi32full.dll
C:\Windows\SysWOW64\msvcp_win.dll
C:\Windows\SysWOW64\imm32.dll
C:\Windows\SysWOW64\KernelBase.dll
C:\Windows\SysWOW64\ucrtbase.dll
C:\Windows\SysWOW64\user32.dll
```

Line 23 of 26

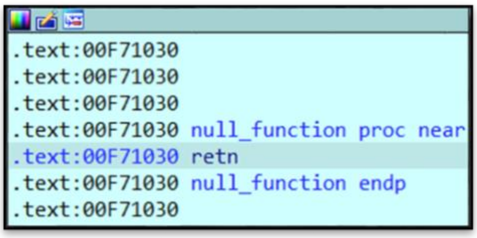


By start investigation this malware we started to follow the main function we found and found that the malware firstly trying to disable what called “shadow volume” by using the “Wow64DisableWow64FsRedirection” function under “kernel32.dll” this prevent from the user to attempt to restore his system back in case of ransomwares. Than it create a “kernel32.dll” file under “system32”, and retrieve the usage in “shadow volume” by calling again the “Wow64DisableWow64FsRedirection”.
The is gather information about the kernel32.dll time and date and the current location of the ‘sample’ by looking at the cmd path using ‘GetCommandLineW’ and ‘CommandLineToArgvW’. Strangely, every time the sample load existing string from the db it check it size, sometimes to allocate space but sometimes it seems without any reason.
All the mention above, can be found under function catalog under name: **‘get_windows_and_copy_sample_path_to_db’** call from the **main** function.



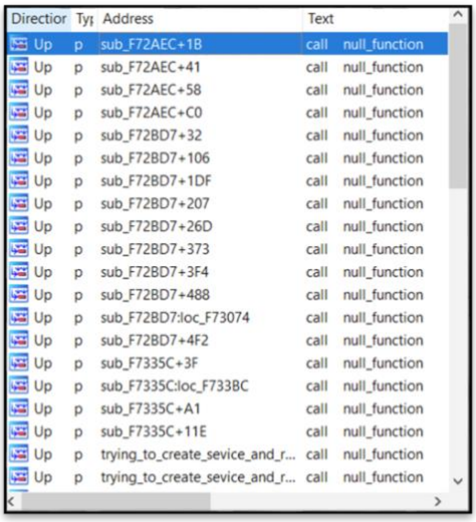
```
.text:00F75A3B sub     esp, 10h
.text:00F75A3E push    esi
.text:00F75A3F call    get_windows_and_copy_sample_path_to_db
.text:00F75A44 mov     esi, offset unk_F86804
.text:00F75A49 push    esi
.text:00F75A4A call    null_function
```

We also notice that there are a lot of calls to useless function we called: ‘null_function’.



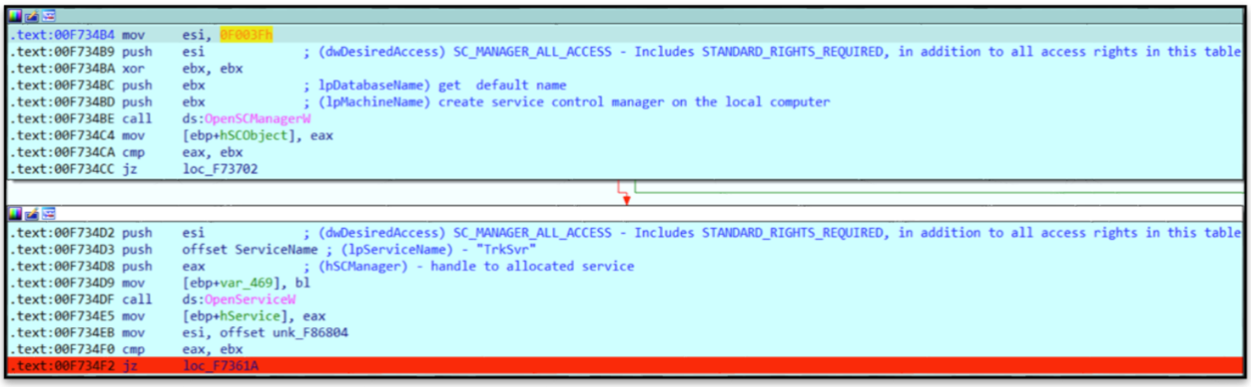
```
.text:00F71030
.text:00F71030
.text:00F71030
.text:00F71030 null_function proc near
.text:00F71030 retn
.text:00F71030 null_function endp
.text:00F71030
```

We thought that this function appear just to confuse if someone will try to analyze like disassemble the code or the instruction can be change in runtime dynamically and then the code will change its behavior by adding calls or jumps.



Directior	Ty	Address	Text
Up	p	sub_F72AEC+1B	call null_function
Up	p	sub_F72AEC+41	call null_function
Up	p	sub_F72AEC+58	call null_function
Up	p	sub_F72AEC+C0	call null_function
Up	p	sub_F72BD7+32	call null_function
Up	p	sub_F72BD7+106	call null_function
Up	p	sub_F72BD7+1DF	call null_function
Up	p	sub_F72BD7+207	call null_function
Up	p	sub_F72BD7+26D	call null_function
Up	p	sub_F72BD7+373	call null_function
Up	p	sub_F72BD7+3F4	call null_function
Up	p	sub_F72BD7+488	call null_function
Up	p	sub_F72BD7:loc_F73074	call null_function
Up	p	sub_F72BD7+4F2	call null_function
Up	p	sub_F7335C+3F	call null_function
Up	p	sub_F7335C:loc_F733BC	call null_function
Up	p	sub_F7335C+A1	call null_function
Up	p	sub_F7335C+11E	call null_function
Up	p	trying_to_create_sevice_and_r...	call null_function
Up	p	trying_to_create_sevice_and_r...	call null_function

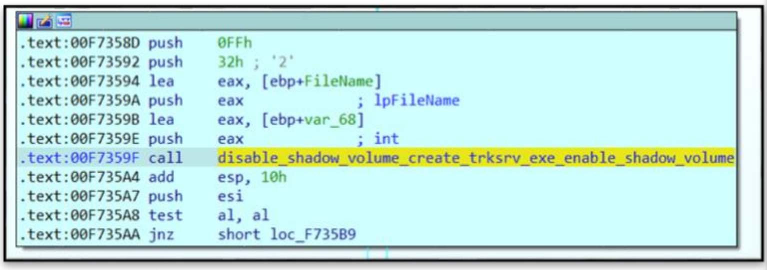
At Next stage the sample tried to check the architecture of the running machine, we saw a comparation between the ‘AMD64’ and the a result of query the registry using ‘RegQueryValueExW’, both string compare using ‘_wcscmp’, if so the sample trying to open a new service using ‘OpenSCManagerW’ and ‘OpenServiceW’ but we can see here that the access this service get is ‘0F003F’ which is the include all the privilege.



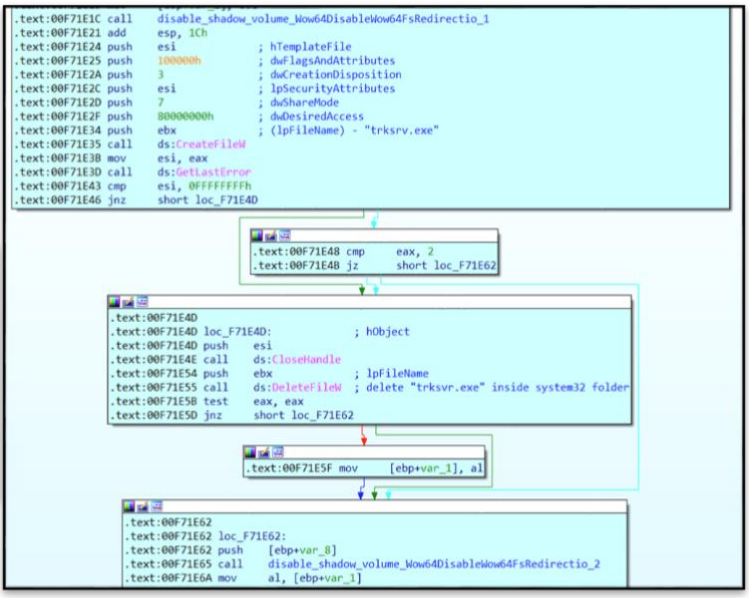
```
.text:00F734B4 mov     esi, 0F003Fh
.text:00F734B9 push    esi             ; (dwDesiredAccess) SC_MANAGER_ALL_ACCESS - Includes STANDARD_RIGHTS_REQUIRED, in addition to all access rights in this table
.text:00F734BA xor     ebx, ebx
.text:00F734BC push    ebx             ; lpDatabaseName) get  default name
.text:00F734BD push    ebx             ; (lpMachineName) create service control manager on the local computer
.text:00F734BE call    ds:OpenSCManagerW
.text:00F734C4 mov     [ebp+hSCObject], eax
.text:00F734CA cmp     eax, ebx
.text:00F734CC jz      loc_F73702
```

```
.text:00F734D2 push    esi             ; (dwDesiredAccess) SC_MANAGER_ALL_ACCESS - Includes STANDARD_RIGHTS_REQUIRED, in addition to all access rights in this table
.text:00F734D3 push    offset ServiceName ; (lpServiceName) - "TrkSvr"
.text:00F734D8 push    eax             ; (hSCManager) - handle to allocated service
.text:00F734D9 mov     [ebp+var_469], bl
.text:00F734DF call    ds:OpenServiceW
.text:00F734E5 mov     [ebp+hService], eax
.text:00F734EB mov     esi, offset unk_F86804
.text:00F734F0 cmp     eax, ebx
.text:00F734F2 jz      loc_F7361A
```

Next the sample again disable ‘shadow volume’ by call ‘Wow64DisableWow64FsRedirection’ and create ‘trksrv.exe’ (which we found very odd and confusing because there is already file under ‘system32’ that called ‘trksvr.exe’) delete "trksvr.exe" inside system32 folder and then enable ‘shadow volume’ by call ‘Wow64DisableWow64FsRedirection’.



```
.text:00F7358D push    0FFh
.text:00F73592 push    32h ; '2'
.text:00F73594 lea     eax, [ebp+FileName]
.text:00F7359A push    eax             ; lpFileName
.text:00F7359B lea     eax, [ebp+var_68]
.text:00F7359E push    eax             ; int
.text:00F7359F call    disable_shadow_volume_create_trksrv_exe_enable_shadow_volume
.text:00F735A4 add     esp, 10h
.text:00F735A7 push    esi
.text:00F735A8 test    al, al
.text:00F735AA jnz     short loc_F735B9
```



```
.text:00F71E1C call    disable_shadow_volume_Wow64DisableWow64FsRedirectio_1
.text:00F71E21 add     esp, 1Ch
.text:00F71E24 push    esi             ; hTemplateFile
.text:00F71E25 push    100000h         ; dwFlagsAndAttributes
.text:00F71E2A push    3               ; dwCreationDisposition
.text:00F71E2C push    esi             ; lpSecurityAttributes
.text:00F71E2D push    7               ; dwShareMode
.text:00F71E2F push    80000000h       ; dwDesiredAccess
.text:00F71E34 push    ebx             ; (lpFileName) - "trksrv.exe"
.text:00F71E35 call    ds:CreateFileW
.text:00F71E3B mov     esi, eax
.text:00F71E3D call    ds:GetLastError
.text:00F71E43 cmp     esi, 0FFFFFFFFh
.text:00F71E46 jnz     short loc_F71E4D
```

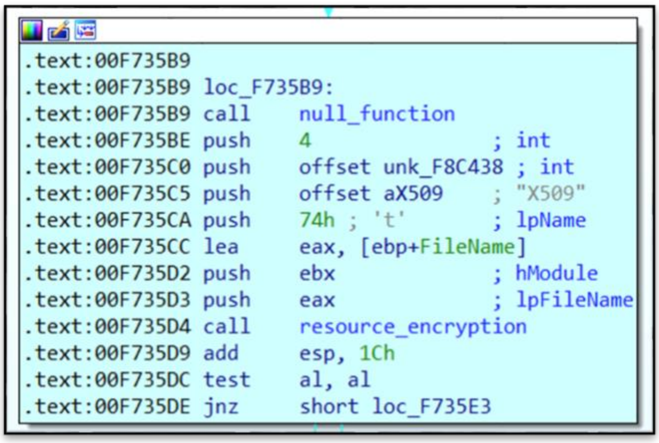
```
.text:00F71E48 cmp     eax, 2
.text:00F71E4B jz      short loc_F71E62
```

```
.text:00F71E4D
.text:00F71E4D loc_F71E4D:                     ; hObject
.text:00F71E4D push    esi
.text:00F71E4E call    ds:CloseHandle
.text:00F71E54 push    ebx             ; lpFileName
.text:00F71E55 call    ds:DeleteFileW  ; delete "trksvr.exe" inside system32 folder
.text:00F71E5B test    eax, eax
.text:00F71E5D jnz     short loc_F71E62
```

```
.text:00F71E5F mov     [ebp+var_1], al
```

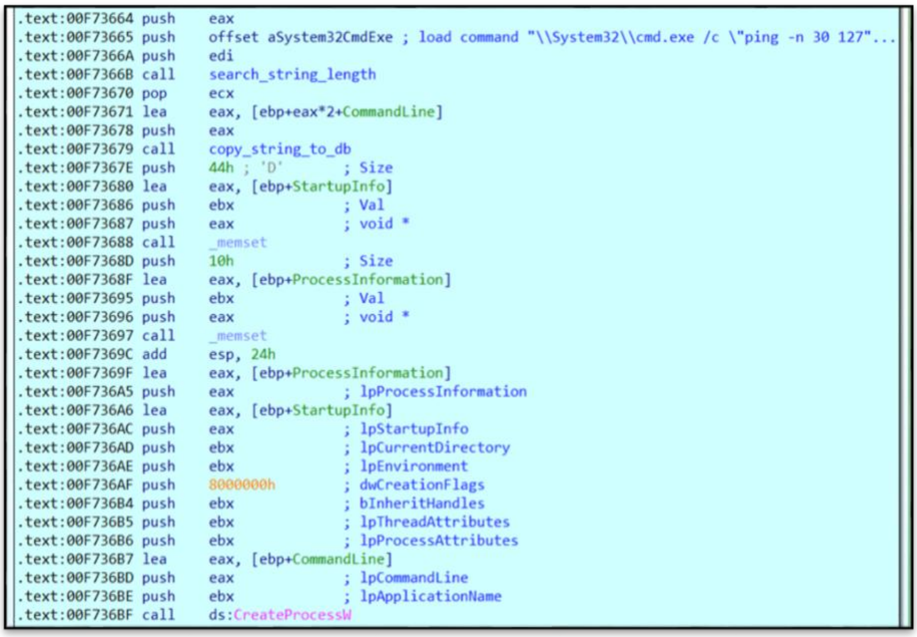
```
.text:00F71E62
.text:00F71E62 loc_F71E62:
.text:00F71E62 push    [ebp+var_8]
.text:00F71E65 call    disable_shadow_volume_Wow64DisableWow64FsRedirectio_2
.text:00F71E6A mov     al, [ebp+var_1]
```

We can see here that the sample is trying to use “X509” resource and pass it to ‘resource_encryption’ function



```
.text:00F735B9
.text:00F735B9 loc_F735B9:
.text:00F735B9 call    null_function
.text:00F735BE push    4               ; int
.text:00F735C0 push    offset unk_F8C438 ; int
.text:00F735C5 push    offset aX509    ; "X509"
.text:00F735CA push    74h ; 't'       ; lpName
.text:00F735CC lea     eax, [ebp+FileName]
.text:00F735D2 push    ebx             ; hModule
.text:00F735D3 push    eax             ; lpFileName
.text:00F735D4 call    resource_encryption
.text:00F735D9 add     esp, 1Ch
.text:00F735DC test    al, al
.text:00F735DE jnz     short loc_F735E3
```

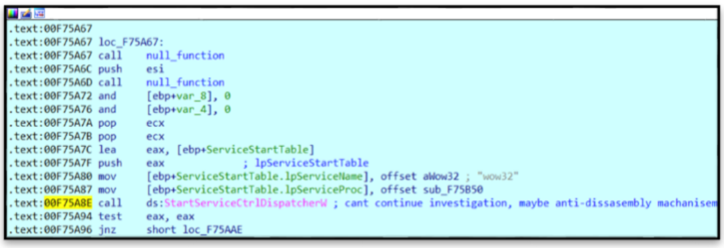
Than the sample trying to create process that get argument that we already sew at the static string section “\System32\cmd.exe /c "ping -n 30 127.0.0.1 >nul && sc config TrkSvr binpath= system32\trksrv.exe && ping -n 10 127.0.0.1 >nul && sc start TrkSvr "”, this string is insert to ‘CreateProcessW’ in ‘lpApplicationName’ and ‘lpStartupInfo’. For our understanding, this command will open command lins and run the file that we have just created with the following parameter (the commands above). We still don’t know what the file TrkSrv.exe do, but seem like it connect to a localhost machine and send ping which can be interpreted as a signal to start a action or passing information.



```
.text:00F73664 push    eax
.text:00F73665 push    offset aSystem32CmdExe ; load command "\\System32\\cmd.exe /c \"ping -n 30 127"...
.text:00F7366A push    edi
.text:00F7366B call    search_string_length
.text:00F73670 pop     ecx
.text:00F73671 lea     eax, [ebp+eax*2+CommandLine]
.text:00F73678 push    eax
.text:00F73679 call    copy_string_to_db
.text:00F7367E push    44h ; 'D'       ; Size
.text:00F73680 lea     eax, [ebp+StartupInfo]
.text:00F73686 push    ebx             ; Val
.text:00F73687 push    eax             ; void *
.text:00F73688 call    _memset
.text:00F7368D push    10h             ; Size
.text:00F7368F lea     eax, [ebp+ProcessInformation]
.text:00F73695 push    ebx             ; Val
.text:00F73696 push    eax             ; void *
.text:00F73697 call    _memset
.text:00F7369C add     esp, 24h
.text:00F7369F lea     eax, [ebp+ProcessInformation]
.text:00F736A5 push    eax             ; lpProcessInformation
.text:00F736A6 lea     eax, [ebp+StartupInfo]
.text:00F736AC push    eax             ; lpStartupInfo
.text:00F736AD push    ebx             ; lpCurrentDirectory
.text:00F736AE push    ebx             ; lpEnvironment
.text:00F736AF push    8000000h        ; dwCreationFlags
.text:00F736B4 push    ebx             ; bInheritHandles
.text:00F736B5 push    ebx             ; lpThreadAttributes
.text:00F736B6 push    ebx             ; lpProcessAttributes
.text:00F736B7 lea     eax, [ebp+CommandLine]
.text:00F736BD push    eax             ; lpCommandLine
.text:00F736BE push    ebx             ; lpApplicationName
.text:00F736BF call    ds:CreateProcessW
```

Next the sample close all it handles, seems that he doesn’t need them anymore. Maybe the call above continue the operation.

Here is the sample trying to start the current thread as a service and pass this ‘StartServiceCtrlDispatchW’ function eax which contain ‘wow32’ and some function.
We couldn’t go further than this, IDA debugger crush each time we tried to execute this like, but we thought that this is some way to move away analyzers like dis-assemblers.



```
.text:00F75A67
.text:00F75A67 loc_F75A67:
.text:00F75A67 call    null_function
.text:00F75A6C push    esi
.text:00F75A6D call    null_function
.text:00F75A72 and     [ebp+var_8], 0
.text:00F75A76 and     [ebp+var_4], 0
.text:00F75A7A pop     ecx
.text:00F75A7B pop     ecx
.text:00F75A7C lea     eax, [ebp+ServiceStartTable]
.text:00F75A7F push    eax             ; lpServiceStartTable
.text:00F75A80 mov     [ebp+ServiceStartTable.lpServiceName], offset aWow32 ; "wow32"
.text:00F75A87 mov     [ebp+ServiceStartTable.lpServiceProc], offset sub_F75B50
.text:00F75A8E call    ds:StartServiceCtrlDispatcherW ; cant continue investigation, maybe anti-dissasembly machanisem
.text:00F75A94 test    eax, eax
.text:00F75A96 jnz     short loc_F75AAE
```

- Anti-disassembly and anti-debugging and anti-vm tricks: