Team Precuzzo - EPICODE

S10 - L5

TEAM

Iosif Castrucci
Donato Tralli
Gianpaolo Miliccia Mendoza

14/06/2024

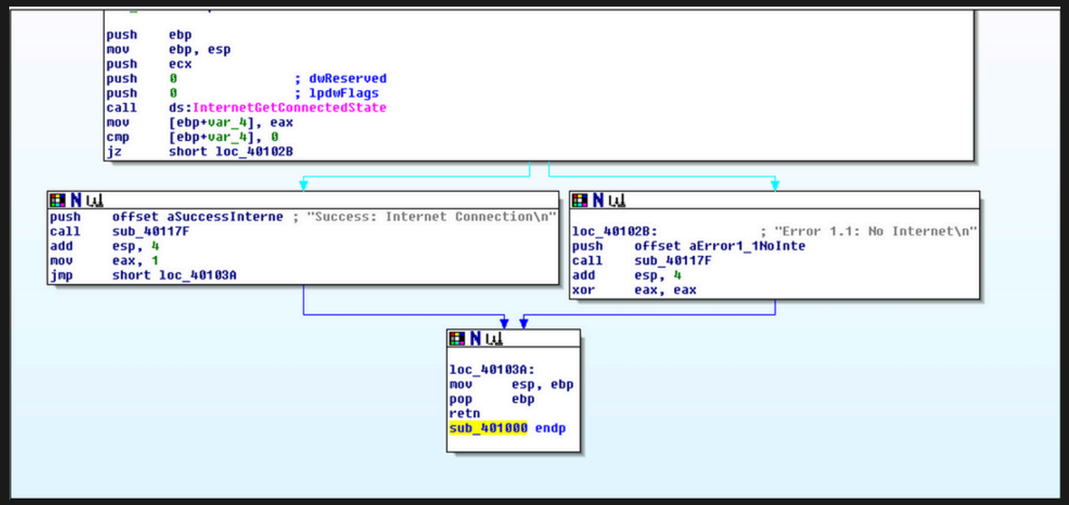
Instructions:

With reference to the file Malware_U3_W2_L5 located in the folder "Esercizio_Pratico_U3_W2_L5" on the desktop of the virtual machine dedicated to malware analysis, answer the following questions:

1. Which libraries are imported by the executable file?
2. What are the sections that make up the executable file of the malware?

With reference to the figure in slide 3, answer the following questions:

3. Identify the known constructs (stack creation, possible loops).
4. Hypothesize the behavior of the functionality implemented in assembly.
5. BONUS: Create a table with the meaning of each line of code.



1. Libraries used by the malware:

With the aid of the CFF Explorer tool, we find that the malware calls the following **libraries:**

KERNEL32.dll: contains numerous essential functions for the operation of the Windows operating system and applications. These functions include memory management, process and thread management, input/output operations, file management, time and date management, resource management, and many more.

WININET.dll: contains functions for implementing some network protocols like HTTP, FTP, NTP. Essentially, it provides an API to allow Windows applications to make requests and manage network operations, such as downloading files from the Internet, sending HTTP requests to web servers, and accessing FTP services.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	00006084

Observing the image above, we can see that the KERNEL32 library imports 44 functions, including:

Sleep: This is a Windows system function used to pause the execution of a thread for a specified period of time to allow other threads to be executed in the system during that time.

CloseHandle: This is an API used to close a handle (a type of reference or pointer) to a kernel object. Essentially, this function is used to release the resources used by a handle after it has been used.

GetProcAddress: This dynamic library function allows programmers to obtain a pointer to a function within a DLL loaded in memory. This is often used when one wants to call a function exported by a DLL dynamically during program execution, rather than linking it statically during compilation.

VirtualAlloc: This is a programming function primarily used on the Windows platform to reserve or allocate virtual memory for a process. It allows programmers to reserve a portion of virtual memory without necessarily allocating corresponding physical memory in physical storage space (RAM or hard disk).

VirtualFree: This is a Windows API function used to free memory dynamically allocated by a process.

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep
00006940	00006940	027C	SetStdHandle
0000692E	0000692E	0156	GetStringTypeW
0000691C	0000691C	0153	GetStringTypeA
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA

The **WININET** library imports 5 functions, including:

InternetOpenUrl: This function is used to open an Internet connection and retrieve the content of a resource identified by a specified URL. It is widely used in Windows applications to download data from online resources, such as web pages, files, and more. Essentially, it allows applications to make HTTP or FTP requests to obtain content from the Internet.

InternetCloseHandle: This function is used to close an Internet resource handle previously opened or created by other functions of the WinINet library. Using this function is important for properly releasing allocated resources and freeing associated memory, ensuring correct program operation and preventing potential memory leaks.

InternetReadFile: This function is used to read data from an Internet resource identified by a handle. It allows applications to retrieve data from online resources, such as web pages or files on FTP servers.

InternetGetConnectedState: This function is used to determine the network connection state of the system. It allows applications to check if the system is connected to the Internet or a local network.

InternetOpen: This function creates a handle that represents a communication session, providing an entry point for network access. Through this session, the application can perform network operations such as downloading resources from the Internet, sending HTTP requests, and other communication activities.

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

2. Composition of the malware

- **.text**: This section contains the instructions (the lines of code) that the CPU will execute once the software is started. Generally, this is the only section of an executable file that is executed by the CPU, as all other sections contain data or supporting information.

- **.rdata**: This section typically includes information about the libraries and functions imported and exported by the executable. This information, as we have seen, can be obtained with CFF Explorer.

- **.data**: This section typically contains the data/global variables of the executable program, which need to be accessible from any part of the program. A variable is considered global when it is not defined within the context of a function but is instead declared globally and is therefore accessible from any function within the executable.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

3. Known constructs

```
push    ebp
mov     ebp, esp
push    ecx
push    0                       ; dwReserved
push    0                       ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

Creation of a stack of unspecified size. It is dynamically allocated.

Function call with parameter passing. The 3 parameters are passed via the stack.

If statement. Compares the local variable with 0; if the variable is equal to 0, it skips.

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

Function call with parameter passing. A variable of type string is loaded.

```
loc_40102B:                     ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
add     esp, 4
xor     eax, eax
```

Function call with parameter passing. A string variable is loaded.

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

Stack deallocation

4. Hypotheses about behavior

The code in question uses the function `**InternetGetConnectedState**` to check the status of the network connection. If the returned value is nonzero, the machine is connected to the Internet. The execution flow skips the conditional jump instruction and proceeds to load a string variable onto the stack. This string presumably contains the message "Success, Internet Connection\n". Subsequently, the code calls a subroutine located at address 0x40117F.

If the returned value is zero, the machine is not connected to the Internet. In this case, the execution flow follows the jump instruction to move to the specified address and load another string variable onto the stack, presumably containing the message "Error 1.1: No Internet\n". Even in this scenario, the subroutine located at address 0x40117F is called.

It can be inferred that at address 0x40117F there is a function responsible for printing the string currently loaded onto the stack.

5. Code Analysis

1	push	ebp	The contents of the register ebp are pushed onto the stack.
2	mov	ebp, esp	I move the contents of esp into ebp.
3	push	ecx	The contents of the register ecx are pushed onto the stack.
4	push	0 ;dwReserved	I push the value 0 onto the stack.
5	push	0 ;lpdwFlags	I push the value 0 onto the stack.
6	call	ds:InternetGetCo nnectState	It calls the function InternetGetConnectedState.
7	mov	[ebp+var_4], eax	I move the contents of register eax into the local variable [ebp+var_4].
8	cmp	[ebp+var_4],0	The local variable is compared with 0.
9	jz	short loc_40102B	If the previous instruction has set the Zero Flag (ZF) to 1, then a jump to the specified location is performed; otherwise, execution continues.

10	push	offset asuccessInterne	The aSuccessInterne variable of type string is loaded onto the stack.
11	call	sub_40117F	The function at address 40117F is called.
12	add	esp, 4	The content of the ESP register is incremented by 4, and the result is stored back in the same register. This frees the stack from the push performed at line 10.
13	mov	eax, 1	The value 1 is stored in the EAX register.
14	jmp	short loc_40103A	An unconditional jump is made to memory location 40103A to skip the portion of code that is executed in the case of the jump at line 9.
15	loc_40102B		It is a flag indicating that I am at location 40102B.
16	push	offset aError1_1NoInte	The string variable “aError1_1NoInte” is loaded onto the stack.
17	call	sub_40117F	The function at address 40117F is called.
18	add	esp, 4	The content of the esp register is incremented by 4, and the result is stored back in the same register. This operation frees the stack from the push performed at line 16.

19	xor	eax, eax	The logical XOR operation is performed on the content of EAX to initialize it to 0.
20	loc_40103A		It's a flag indicating that I am at location 40103A.
21	mov	esp, ebp	The content of the ebp register is moved into the esp register.
22	pop	ebp	To remove the content located at the address indexed by ebp from the stack.
23	retn		Terminating a subroutine or function and returning to the calling point
24	sub_401000 endp		It is used to delimit the body of a subroutine and signal its end to the compiler or assembler ****

*dwReserved: In the function "InternetGetConnectedState," the parameter dwReserved is reserved for future use and is currently not utilized.

**lpdwFlags: The prefix "lp" stands for "long pointer," indicating that it is a pointer to a DWORD value. In the previously mentioned function "InternetGetConnectedState," the lpdwFlags parameter is used to return the Internet connection status. Specifically, the function writes the connection state data to the memory address pointed to by lpdwFlags, allowing the caller to obtain this information after calling the function.

***InternetGetConnectedState: This function is used to check the Internet connection status on a Windows operating system. When the function returns TRUE, the Internet connection is active; if it returns FALSE, there is no connection.

****endp: The ENDP command is a directive used in assembly language to indicate the end of a procedure or subroutine. When this directive is encountered, the compiler or assembler understands that the code following it is part of the procedure or subroutine previously defined with the PROC directive. In the specific case of the code under examination, the command "sub_401000 endp" indicates the end of the function that began at the address.