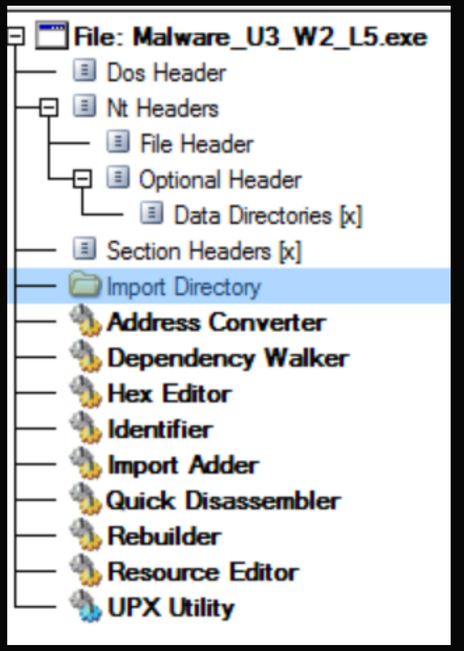
PROGETTO s10/L5

Traccia:
Con riferimento al file Malware_U3_W2_L5 presente all’interno della cartella «Esercizio_Pratico_U3_W2_L5 » sul desktop della macchina virtuale dedicata per l’analisi dei malware, rispondere ai seguenti quesiti:

Sara Spaccialbelli

1. Quali librerie vengono importate dal file eseguibile?

Per prima cosa importiamo il nostro file eseguibile su CFF Explorer, un tool utile che ci permette di per esaminare, modificare e analizzare file eseguibili.



Per analizzare le librerie e le funzioni importate, andiamo su import directory :

Malware_U3_W2_L5.exe

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	000060B4

Spiegazione librerie

Kernell32.dll

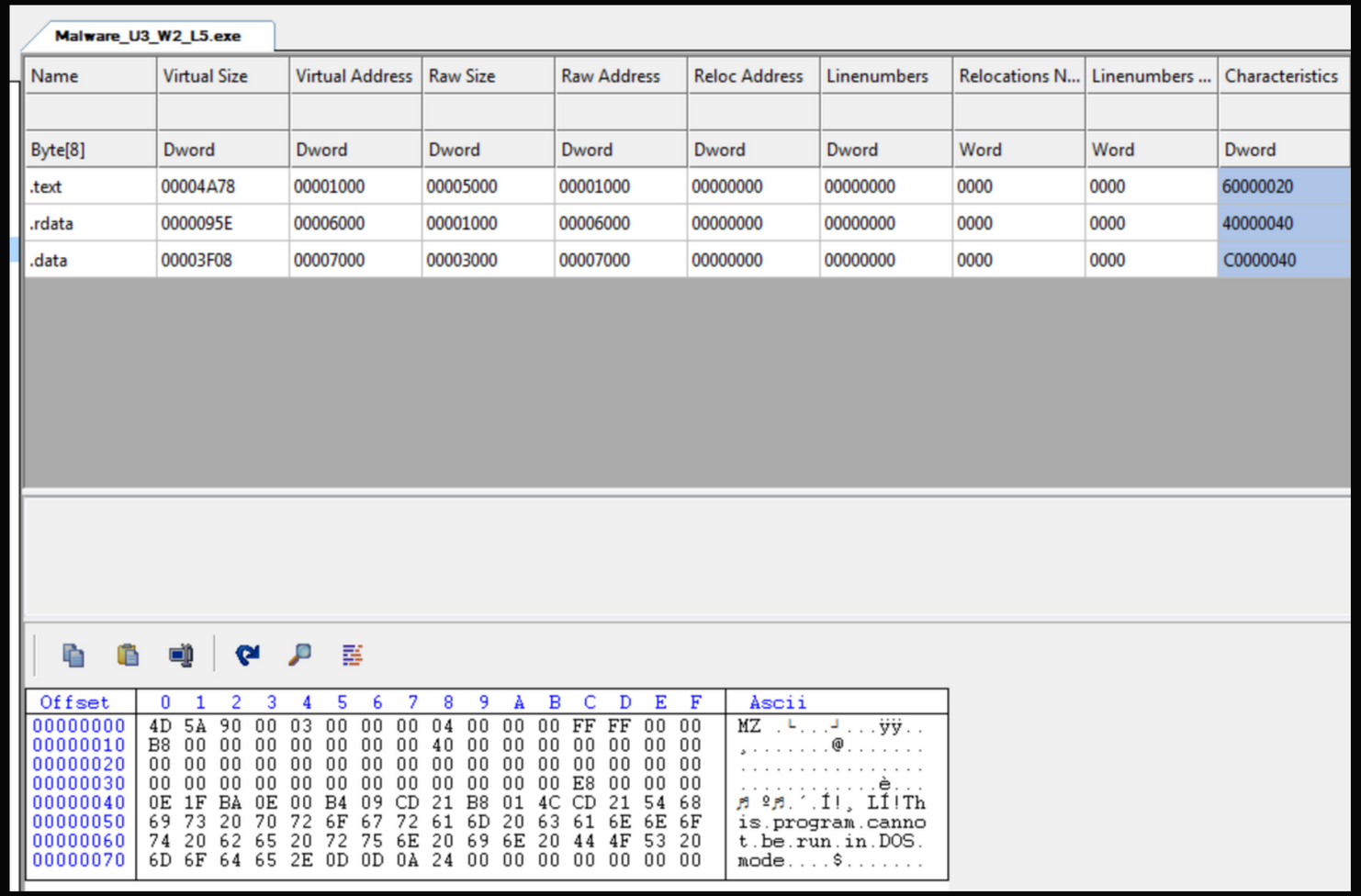
Il suo nome deriva dal fatto che contiene funzioni (DLL, Dynamic Link Library) utilizzate dal kernel del sistema operativo Windows. Questa libreria fornisce una vasta gamma di funzionalità, tra cui gestione dei file, gestione della memoria, gestione dei processi...ed è dunque essenziale per il corretto funzionamento di molti programmi e componenti del sistema operativo Windows

Wininet.dll

Libreria di sistema di Microsoft utilizzata per fornire funzionalità di connettività di rete e accesso a Internet nei sistemi operativi Windows. Questa DLL offre un'ampia gamma di funzioni per l'accesso e la gestione di risorse su Internet, è ampiamente utilizzata da applicazioni Windows che richiedono connettività di rete, come browser web, client di posta elettronica, applicazioni di download e molti altri programmi che interagiscono con risorse su Internet. È una parte fondamentale del supporto di rete nel sistema operativo Windows

2. Quali sono le sezioni di cui si compone il file eseguibile del malware?

Vediamo ora le sezioni di un file eseguibile, spesso in formato PE, (Portable Executable), le quali sono aree di memoria organizzate all'interno del file che contengono diverse tipologie di dati e istruzioni necessarie per il funzionamento del programma.
Vediamo quali vi sono nella sezione “section headers”:



Malware_U3_W2_L5.exe

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

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ..└...┘...ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.......@.......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	............è...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	♫ º♫.´.Í!¸.LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......

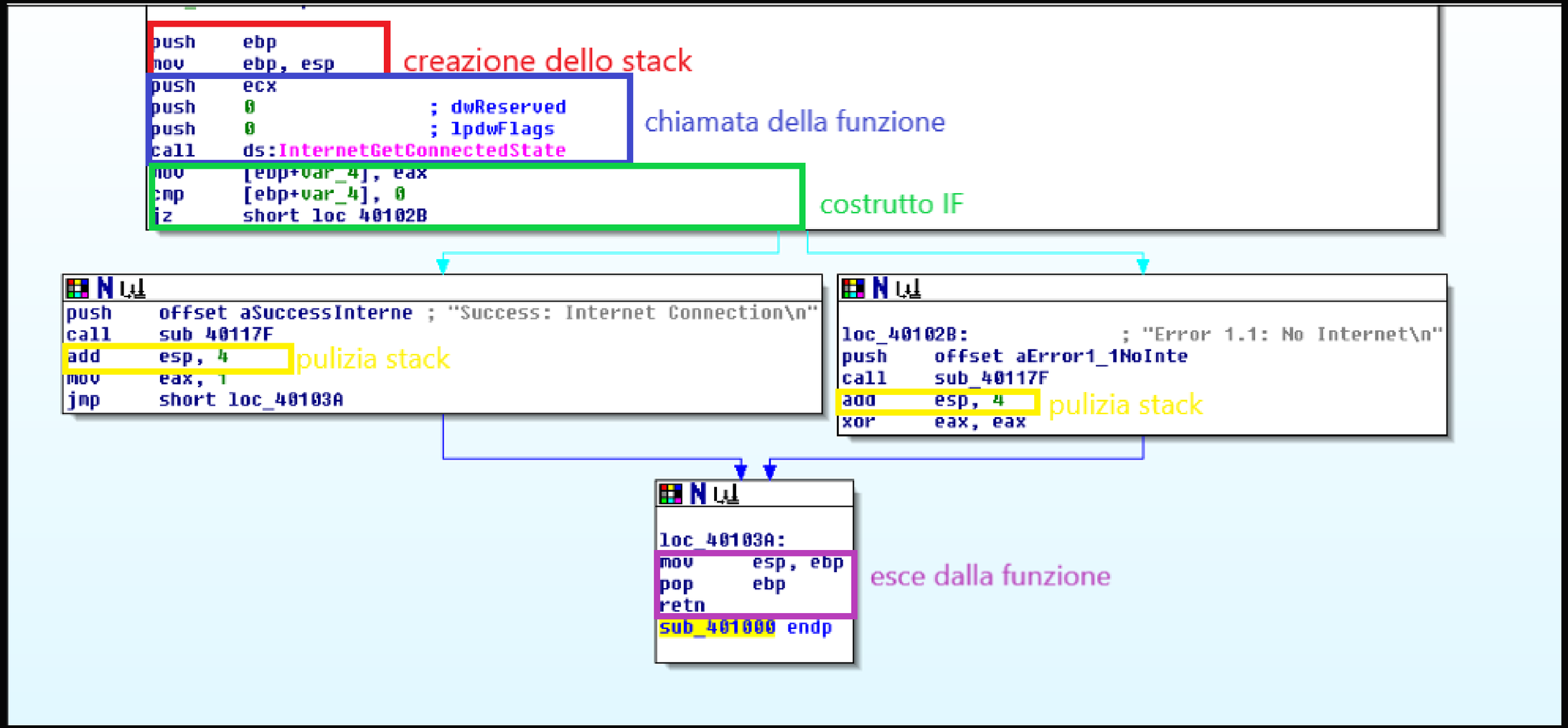
Spiegazione sezioni:

.TEXT → è una delle sezioni più importanti e comuni all'interno di un'eseguibile o di una DLL in ambiente Windows. Questa sezione contiene le righe di codice eseguibile del programma, ovvero le istruzioni che vengono eseguite dalla CPU quando il programma viene avviato.

.DATA → contiene dati e variabili globali utilizzati durante l'esecuzione del programma. Questi dati possono essere letti e scritti dall'applicazione durante l'esecuzione, e dunque modificati durante l'esecuzione del programma e accessibili da diverse parti del codice.

.RDATA → questa sezione include le informazioni sulle librerie e le funzioni importate ed esportate dall'eseguibile, contiene dati di sola lettura che vengono inizializzati durante l'esecuzione del programma.

Con riferimento alla figura in slide 3, rispondere ai seguenti quesiti:
3. Identificare i costrutti noti



4. Ipotizzare il comportamento della funzionalità implementata

Questo codice assembly sembra essere parte di un programma che verifica lo stato della connessione Internet e stampa un messaggio di successo se la connessione è attiva, altrimenti stampa un messaggio di errore.

Inizia impostando il frame pointer (ebp) e lo stack pointer (esp) per gestire le variabili locali e i parametri della funzione. Chiama la funzione InternetGetConnectedState per verificare lo stato della connessione Internet e; salva il valore restituito dalla funzione nella variabile locale var_4.

Tramite un costrutto IF-ELSE, controlla SE il valore di var_4 è zero stampa un messaggio di “error No internet” , ALTRIMENTI se il valore di var_4 non è zero, il programma procede a stampare un messaggio di successo relativo alla connessione Internet.

Alcuni aspetti di questo frammento di codice potrebbero essere considerati sospetti e suggerire che potrebbe far parte di un **MALWARE**, ad esempio:

1. Il codice fa una chiamata a una funzione di sistema (InternetGetConnectedState), che potrebbe essere utilizzata per scopi maligni, ad esempio per verificare la presenza di una connessione Internet per comunicare con un server remoto controllato dal malware.
2. La presenza di messaggi di output come "Success Internet Connection" e "Error 1.1: No Internet" potrebbe indicare che il programma sta interagendo con l'utente in modo sospetto.
3. L'uso di istruzioni come xor eax, eax per azzerare un registro potrebbe essere indicativo di una tecnica di "pulizia" per nascondere il comportamento maligno del programma.

BONUS fare tabella con significato delle singole righe di codice assembly

1)

codice assembly	descrizione
push edb	spinge il valore del registro ebp nello stack.
mov edp, esp	sposta il valore dello stack pointer (esp) nel registro base del frame (ebp). Questo solitamente imposta ebp come un punto di riferimento per accedere alle variabili locali e ai parametri della funzione.
push ecx	spinge/salva il valore del registro ecx nello stack. .
push 0, dwReserved	inserisce il valore 0 nello stack. Il commento ;dwReserved indica che questo valore potrebbe essere utilizzato come parametro in una funzione.
push 0, lpdwFlags	inserisce il valore 0 nello stack. Anche qui, lo lpdwFlags potrebbe rappresentare un parametro in una chiamata di funzione
call ds:InternetGetConnectedState	chiama la funzione InternetGetConnectedState indicata dall'indirizzo ds. Questa funzione è utilizzata per verificare lo stato della connessione Internet
mov [ebp+var_4], eax	memorizza il valore restituito dalla chiamata alla funzione InternetGetConnectedState nella variabile locale var_4, che si trova nel frame dello stack corrente
cmp [ebp+var_4], 0	confronta il valore memorizzato nella variabile var_4 con 0.
jz short loc_40102B	salta all'etichetta loc_40102B se il confronto precedente ha dato esito "zero".
push offset aSuccessInterne	si mette l'indirizzo dell'etichetta a “SuccessInternetConnection” nello stack. L'etichetta sembra indicare un messaggio di successo relativo alla connessione Internet

2)

call sub_40105F	Questa istruzione chiama la subroutine indicata da sub_40105F. Probabilmente questa subroutine è responsabile di stampare il messaggio di successo
add esp, 4	si aggiunge 4 al registro dello stack pointer. Questo serve a liberare lo spazio dello stack occupato dai parametri passati alla funzione chiamata.
mov eax, 1	carica il valore 1 nel registro eax. Probabilmente è utilizzato come valore di ritorno della funzione corrente
jmp short loc_40103A	salta all'etichetta loc_40103A. Presumibilmente, questo è il punto dopo la gestione del successo della connessione Internet
push offset aError1_1NoInte	Mette l'indirizzo dell'etichetta "Error1_1NoInte" nello stack. L'etichetta indica un messaggio di errore relativo alla mancanza di connessione Internet.
call sub_40117F	Chiama la subroutine indicata da sub_40117F, che gestisce il messaggio di errore.
add esp, 4	Aggiunge 4 al registro dello stack pointer per liberare lo spazio dello stack occupato dai parametri passati alla funzione chiamata.

xor eax, eax	forma comune per "azzerare" un registro, poiché qualsiasi numero XOR con se stesso restituirà sempre zero. Quindi, dopo questa istruzione, eax conterrà il valore zero.
mov esp, ebp	Ripristina il valore originale dello stack pointer (esp) con il valore di ebp, elimina il frame dello stack attuale.
pop ebp	Ripristina il valore originale del registro (ebp) dallo stack. Questo è comunemente usato prima di ritornare da una funzione.
retn	Ritorna dalla funzione, utilizzando l'indirizzo di ritorno presente nello stack per determinare dove riprendere l'esecuzione del programma.
sub_401000 endp	Segnala la fine della definizione della subroutine sub_401000, indicando la fine del blocco di codice della subroutine.