S10L5,

BENVENUTI LUIGI

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:

1. Quali librerie vengono importate dal file eseguibile?
2. Quali sono le sezioni di cui si compone il file eseguibile del malware?

Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:

3. Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti)
4. Ipotizzare il comportamento della funzionalità implementata
5. BONUS fare tabella con significato delle singole righe di codice assembly

Procediamo con la risoluzione dell'esercizio.

1- Librerie importate

Windows utilizza per la maggior parte dei file eseguibili il formato **PE** (Portable Executable).

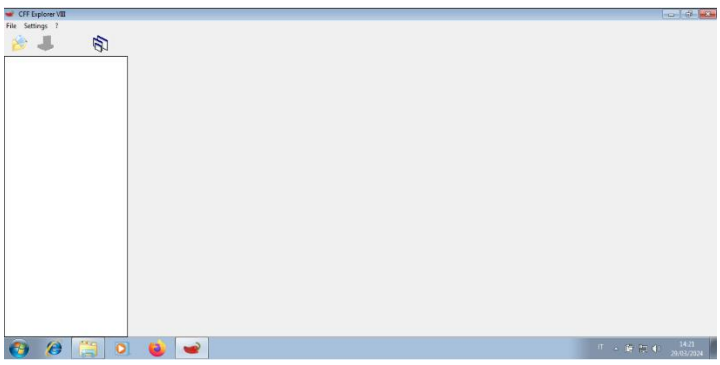
Un file in formato PE contiene al suo interno delle informazioni necessarie al sistema operativo per capire come gestire il codice del file.

Un esempio di ciò sono le librerie: esse contengono un insieme di funzioni.

Quando un programma ha bisogno di una funzione fa riferimento ad una libreria al cui interno è definita la funzione necessaria.

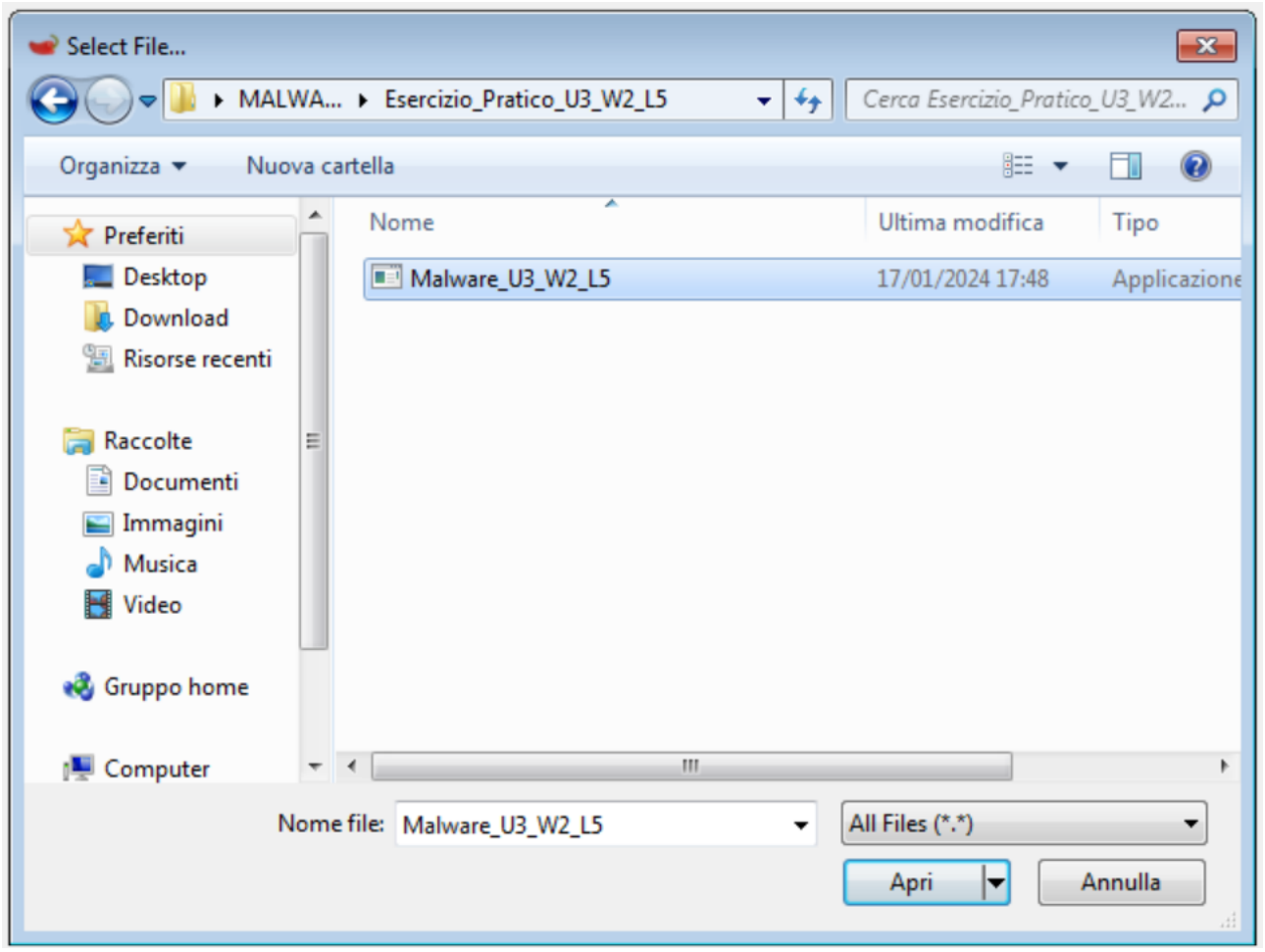
Viene richiesto di verificare quali librerie vengono importate dal malware; questa informazione è preziosa in quanto può risultare estremamente utile nella comprensione dello scopo generale del malware.

Avviata la macchina Windows 7 contenente il malware denominato <<Esercizio_Pratico_U3_W2_L5>> all'interno della cartella <<Malware>> sul desktop, si può risalire alle librerie importate in un qualsiasi codice grazie a "**CFF Explorer**", un tool già preinstallato nella Virtual Machine, situato nella cartella <<Software malware Analysis>>.

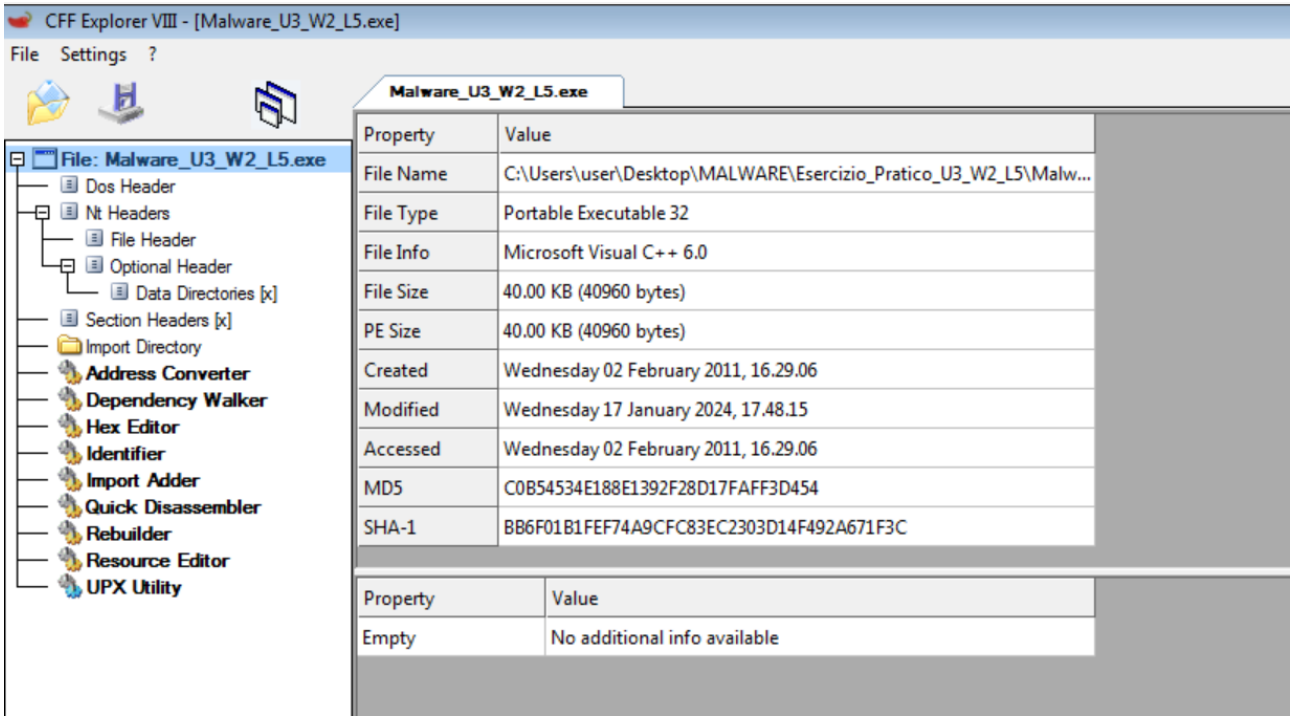


L’icona in alto a sinistra (raffigurante una cartella) permette di aprire un file eseguibile al fine di visionarne le informazioni relative, fra cui anche le librerie.

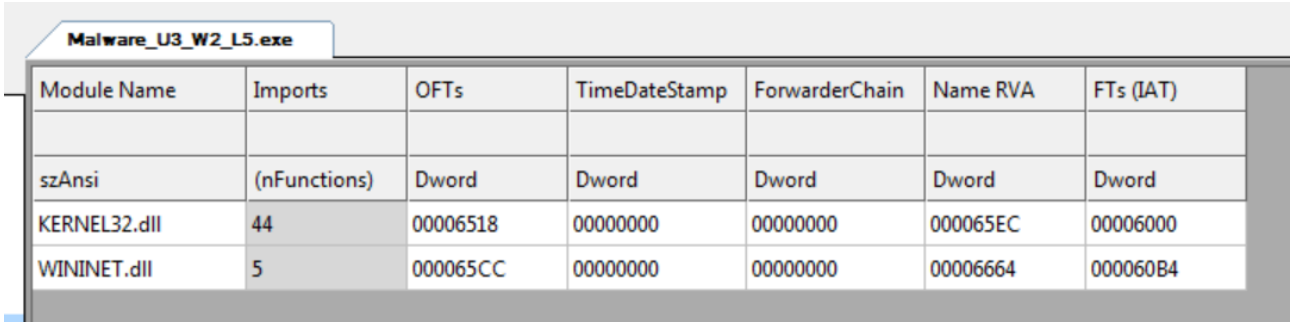
Selezionare dunque il file richiesto (<<Malware_U3_W2_L5>>):



Verifichiamo l’output del programma:



Per consultare le librerie importate dal programma basta selezionare dal menù di sinistra la voce “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

CFF Explorer esplicita, dunque, l’importazione di due librerie da parte del programma:

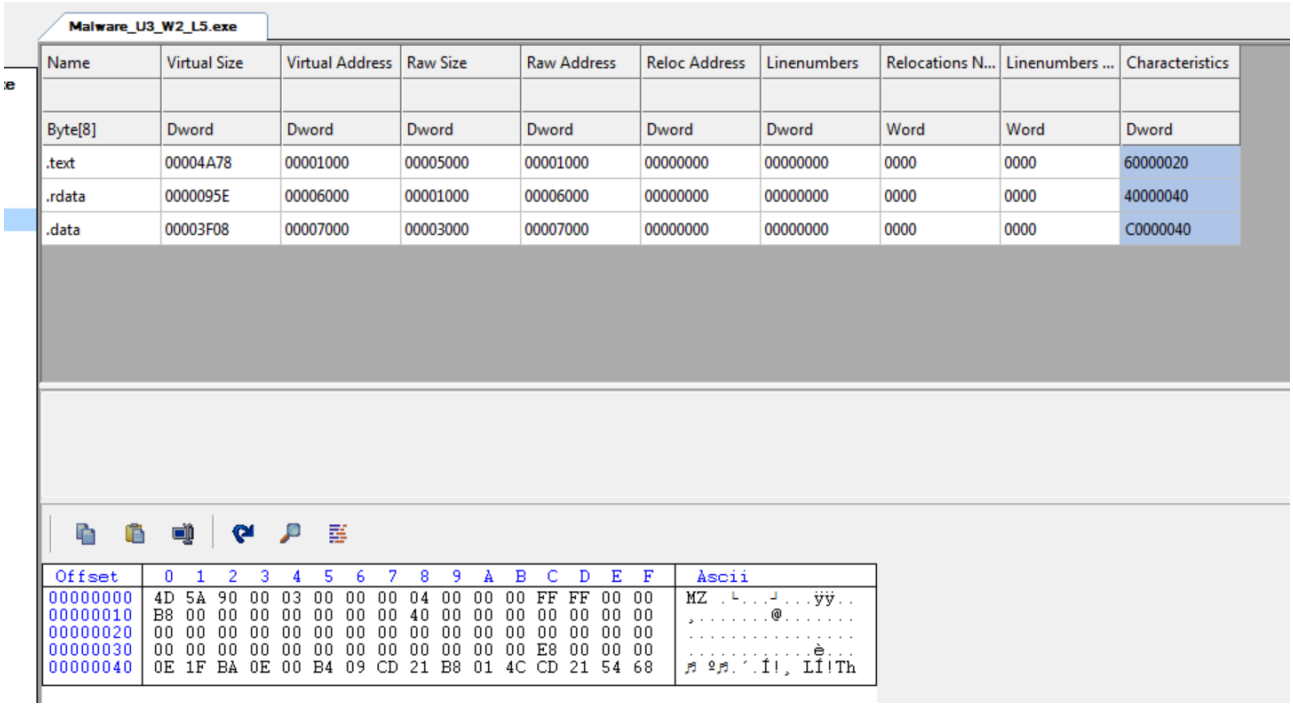
- **KERNEL32.dll**: contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria.
- **WININET.dll:** contiene le funzioni per l’implementazione di alcuni protocolli di rete come HTTP, FTP, NTP.

La tabella fornisce inoltre nella colonna <<Imports>> il numero di funzioni presenti nella libreria selezionata, stilandone anche una lista (con descrizione).

2 – Sezione dell’eseguibile

Oltre alla funzione che permette di visionare le librerie importate, un’altra funzionalità di CFF Explorer è il controllo delle sezioni di un file PE, ovvero delle porzioni di elementi da cui il software stesso è composto.

Per sfruttare questa potenzialità di CFF Explorer, dopo aver selezionato il malware come visto precedentemente, bisogna selezionare dal menù di sinistra la voce “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

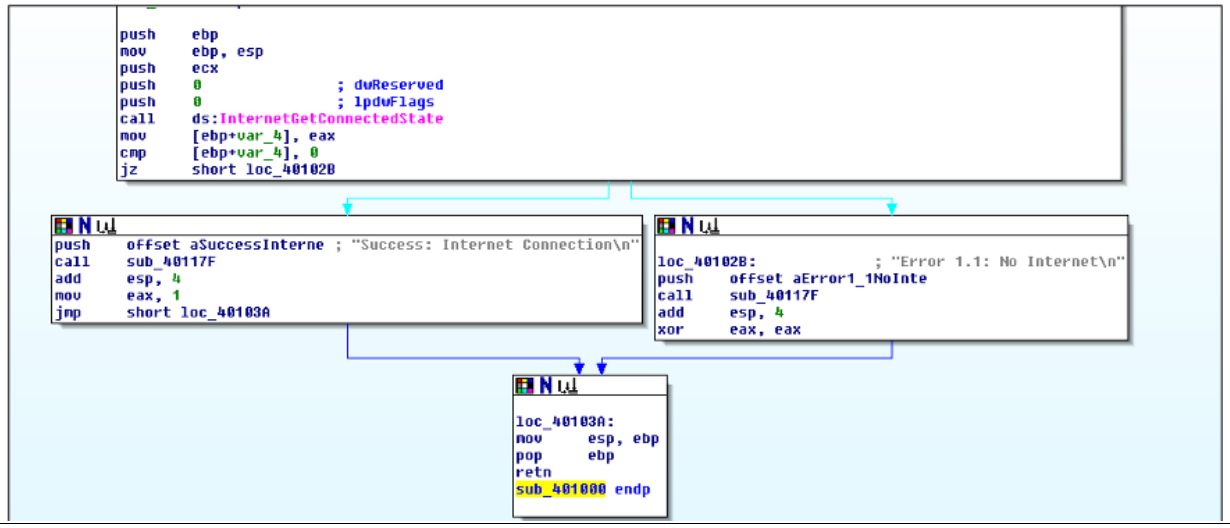
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ ..└...┘...ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.......@.......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	............è...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	♬ ▼♬.´.Í!¸.LÍ!Th

Nel malware preso in esecuzione sono presenti tre sezioni, fra quelle più frequenti in questo tipo di analisi:

- **.text:** contiene le istruzioni che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
- **.rdata**: include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile, informazione che come abbiamo visto possiamo ricavare con CFF Explorer.
- **.data:** contiene tipicamente le variabili globali (accessibile da qualsiasi funzione all'interno dell'eseguibile) del programma eseguibile.

3 – Identificare i costrutti noti:

Figura 1

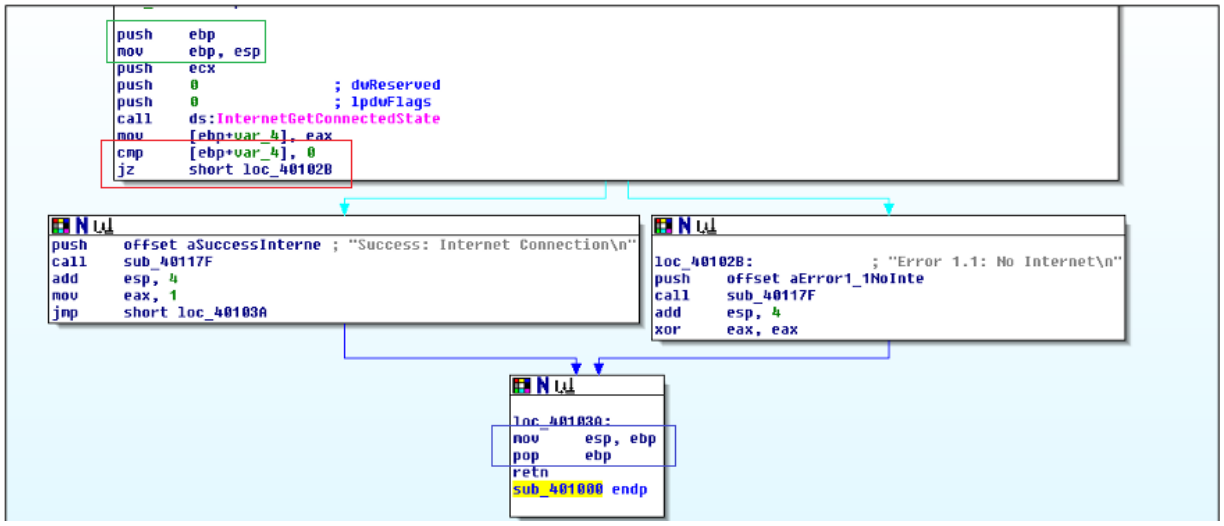


Si possono dividere i costrutti noti in quattro tipologie:

- **Creazione/eliminazione** dello stack;
- **Cicli** (for, while);
- **Scelte** (If, if-else, switch);

A seguire la figura precedente con all'interno cerchiati i costrutti:

Figura 1



1 - Il rettangolo verde contenente le istruzioni <<*push ebp, mov ebp, esp*>> rappresenta la creazione di uno stack, ovvero una "pila di piatti" dove è possibile solamente aggiungere o rimuovere un piatto dalla cima. Viene creato uno stack per ogni chiamata di funzione. Questa struttura prende il nome di «LIFO», ovvero Last-In First-Out (l'ultimo piatto inserito sarà il primo ad essere rimosso). I due elementi EBP ed ESP rappresentano i due puntatori alla base ed alla cima dello stack.

2 - Il rettangolo rosso rappresenta invece un costrutto if, ovvero il salto condizionato ad una locazione di memoria al verificarsi di una condizione. In particolare, il comando cmp (*compare*) esegue il confronto tra una variabile ed il valore 0 modificando di conseguenza i registri ZF (zero flag) e CF (registro dei riporti) secondo la seguente logica:

- Se destinazione = sorgente → ZF = 1; CF = 0.
- Se destinazione < sorgente → ZF = 0; CF = 1.
- Se destinazione > sorgente → ZF = 0; CF = 0.

Il salto condizionato <<jz loc>> esegue una jump alla posizione di memoria inserita come operando solo nel caso in cui la destinazione sia uguale alla sorgente e dunque il flag ZF sia 1.

3 - Il rettangolo blu rappresenta invece l'ultimo costrutto individuato, ovvero l'eliminazione dello stack precedentemente creato, tramite le due righe <<mov esp, ebp; pop ebp>>.

4 – Ipotesi comportamento funzionalità programma

Analizzando la funzione <<InternetGetConnectedState>> si viene a conoscenza che si tratta di una funzione il cui scopo è verificare la presenza o meno di una connessione ad internet, facente parte della libreria <<WININET.h>>; ad essa vengono passati due parametri di nome <<dwReserved>> e <<lpdwFlags>>.

Si può dunque suppore da analisi dinamica basica del malware che lo scopo dello stesso è di verificare la presenza o meno di una connessione ad internet tramite il costrutto if precedentemente analizzato, con due possibili esiti:

- Se il valore di ritorno della funzione è 0, viene stampata a schermo la stringa “Error 1.1: No Internet”, e terminata l’esecuzione del programma;
- Se il valore di ritorno della funzione è invece diverso da 0, viene scritto a schermo il messaggio “Success: Internet Connection”.

Volendo presumere una classificazione del malware, dovremmo procedere ad un’analisi più accurata, in quanto gli elementi raccolti fino ad ora non garantiscono un’identificazione univoca del comportamento.

- **5 BONUS fare tabella con significato delle singole righe di codice assembly**

```
push    ebp             ; Salva il valore di EBP sullo stack
mov     ebp, esp        ; Imposta EBP come stack pointer corrente
push    ecx             ; Mette il valore di ECX sullo stack
push    0               ; inizializza a 0 il parametro dwReserved
push    0               ; inizializza a 0 il parametro lpdwFlags
call    ds:InternetGetConnectedState  ; Chiamata di funzione a InternetGetConnectedState
mov     [ebp+var_4], eax   ; sposta il contenuto nella variabile in [EBP-4]
cmp     [ebp+var_4], 0     ; controllo variabile = 0
jz      short loc_40102B   ; salto condizionale

push    offset aSuccessInterne ; Push dell’indirizzo di memoria della stringa "Success: Internet Connection\n" sullo stack
call    sub_40117F       ; Chiamata alla subroutine sub_40117F
add     esp, 4           ; Aggiunge 4 allo stack pointer per pulire lo stack
mov     eax, 1           ; setta il valore del registro eax ad 1
jmp     short loc_40103A   ; salto incondizionato a loc_40103A
loc_40102B:              ; etichetta di riferimento salto
push    offset aError1_1NoInte ; push della stringa "Error 1.1: No Internet\n" sullo stack
call    sub_40117F       ; chiamata subroutine sub_40117F
add     esp, 4           ; Aggiunge 4 allo stack pointer per pulire lo stack
xor     eax, eax         ; reset a 0 registro eax tramite operatore logico XOR

loc_40103A:              ; etichetta di riferimento salto
mov     esp, ebp         ; reimpostazione dello stack pointer ESP alla posizione del base pointer EBP
```

```
pop     ebp             ; Pop del valore di EBP dallo stack
retn                    ; valore di ritorno subroutine
sub_401000 endp         ; termina la subroutine
```