PROGETTO GIORNO 5

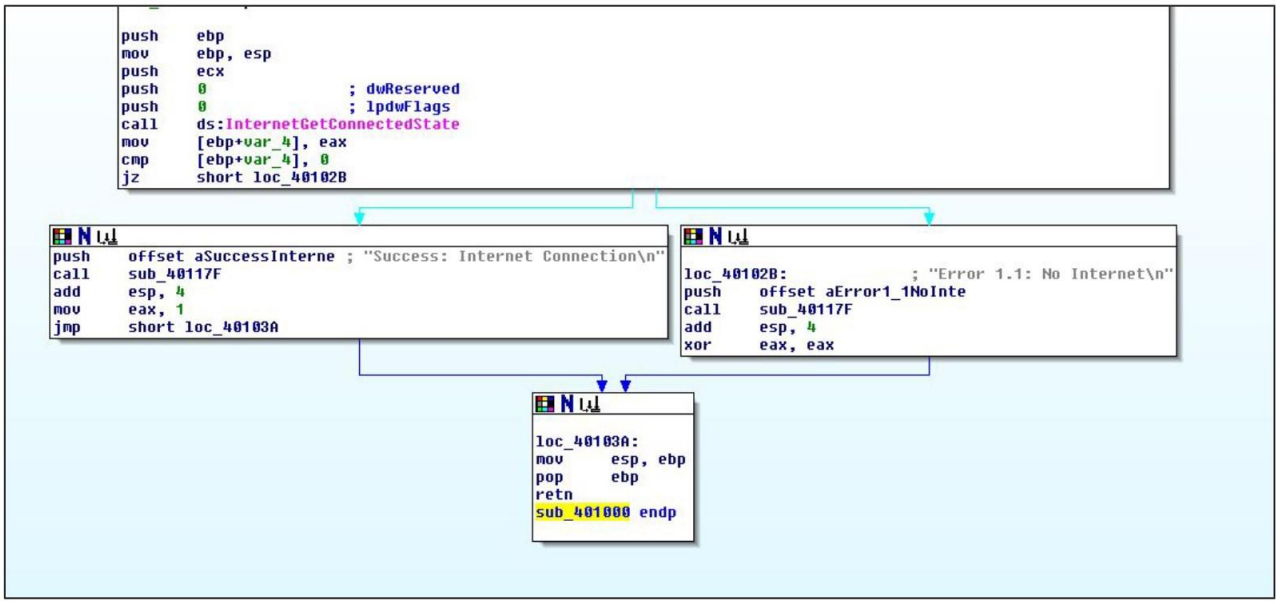
Traccia:

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:

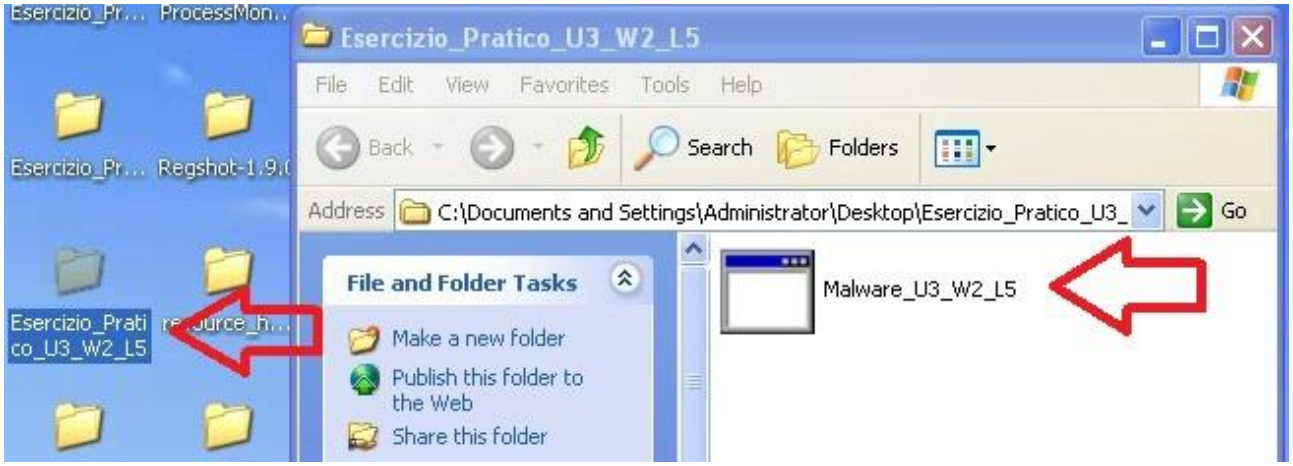
1. Quali **librerie** vengono importate dal file eseguibile?
2. Quali sono le **sezioni** di cui si compone il file eseguibile del malware?

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

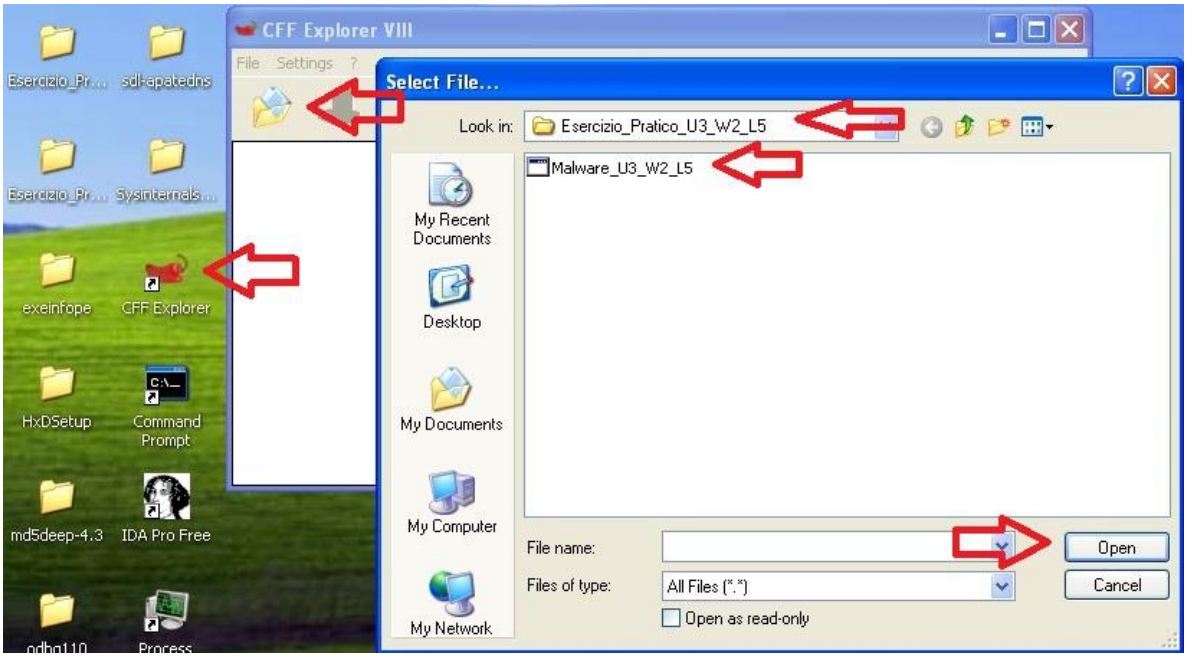


Come prima cosa siamo andati ad avviare la nostra macchina virtuale per iniziare la nostra analisi malware.

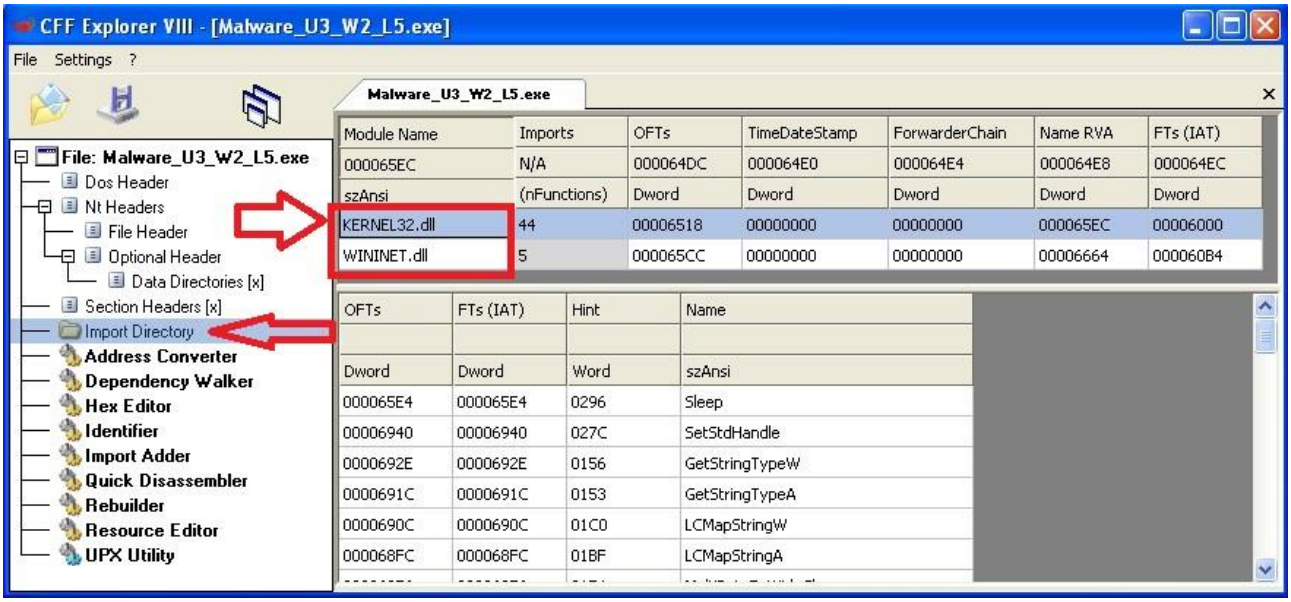


Abbiamo quindi iniziato l'analisi del malware per rispondere ai quesiti.

1. **Quali librerie vengono importate dal file eseguibile?**
   Windows utilizza per la maggior parte dei file eseguibili in formato PE (Portable Execution). Questo formato contiene al suo interno delle informazioni necessarie al sistema operativo per capire come gestire il codice del file, come le librerie. Le informazioni circa le librerie e le funzioni richieste dall'eseguibile sono contenute nell'header del formato PE. Controllare quali sono le librerie e le funzioni importate ed esportate è fondamentale per capire lo scopo del malware. Per farlo, possiamo utilizzare il tool **CFF Explorer**.
   Come prima cosa abbiamo quindi aperto il nostro malware con il tool CFF Explorer presente sulla nostra macchina virtuale.



   Una volta aperto il tool ci siamo spostati nella sezione "**Import Directory**" per poter vedere quali librerie vengono importate.

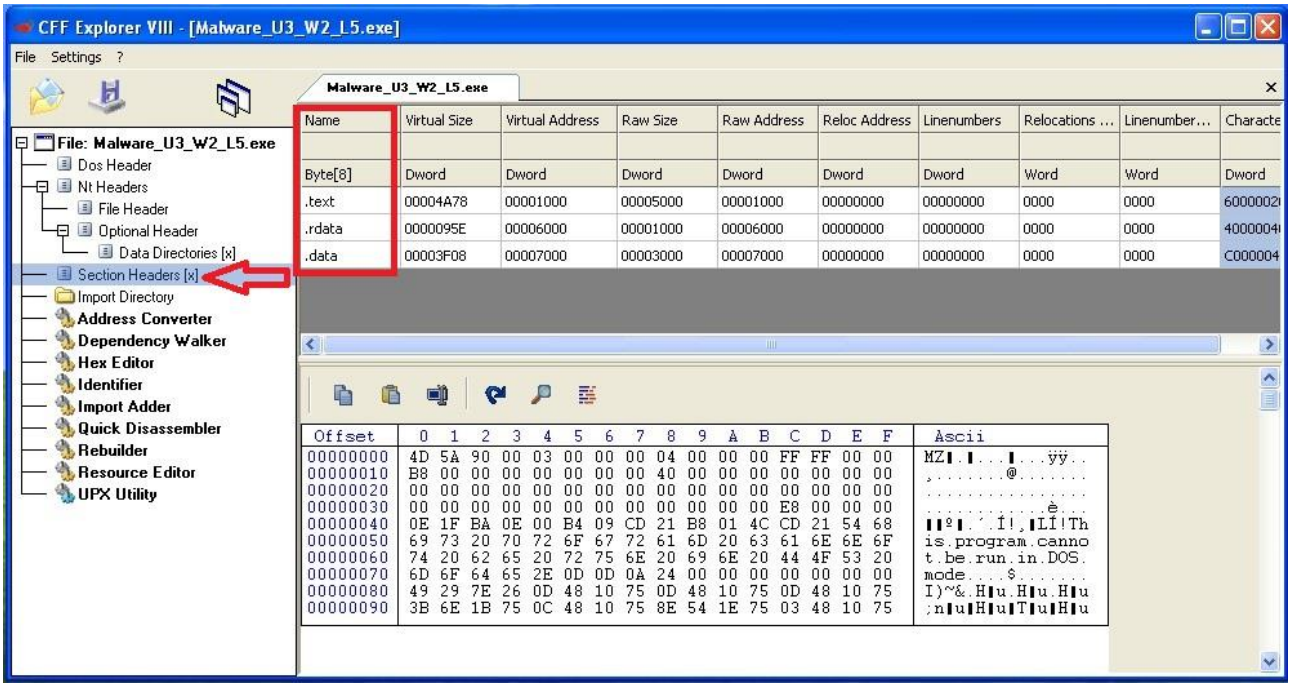


   Come sarà possibile vedere le librerie importate dal malware saranno:

   - **Kernel32.dll**: l: libreria piuttosto comune che contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria.
   - **WININET.dll**: libreria che contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP.

Dopo aver risposto al primo quesito, rimanendo sul tool CFF Explorer siamo andati a rispondere al secondo:

2. **Quali sono le sezioni di cui si compone il file eseguibile del malware?**
   Per rispondere a questa domanda ci siamo spostati sulla sezione del tool "**Section Headers [x]**" che ci permette di controllare le sezioni di cui si compone il file eseguibile del malware, in quanto l'header del formato PE fornisce anche le sezioni di cui si compone il software.

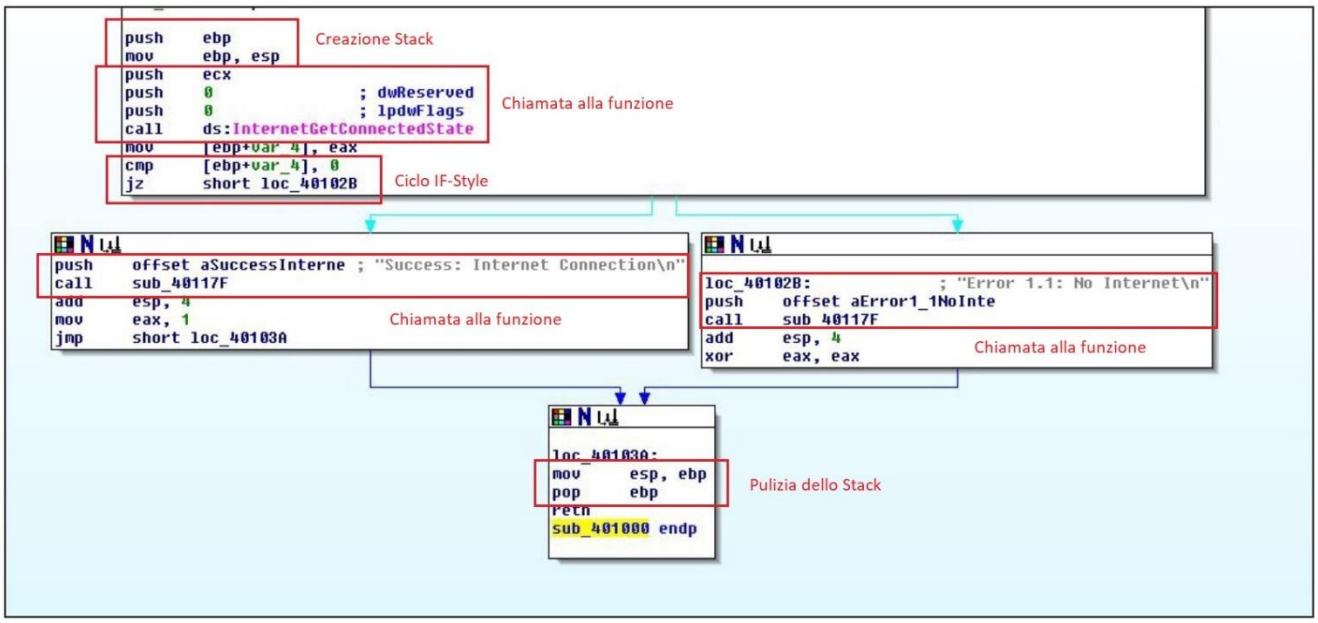


   Le sezioni trovate saranno:
   - **.text**: questa sezione contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
   - **.rdata**: questa sezione include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile.
   - **.data**: questa sezione contiene tipicamente i dati/le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma. Una variabile si dice globale quando non è definita all'interno di un contesto di una funzione, ma bensì è globalmente dichiarata ed è di conseguenza accessibile da qualsiasi funzione dell'eseguibile.

   Questa schermata ci riporterà non solo il nome delle sezioni ma anche altre informazioni quali:
   - Virtual Size: che indica lo spazio allocato per la sezione durante il processo di caricamento dell'eseguibile in memoria.
   - Raw Size: indica lo spazio occupato dalla sezione quando è sul disco.

Ci siamo quindi spostati ad analizzare il codice in Assemblyx86 in figura per rispondere ai quesiti successivi.

3. **Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti):**
   Chi scrive malware non bada alla singola riga di codice, ma piuttosto struttura il codice in base alla funzionalità che vuole implementare e per farlo fa uso dei costrutti, come possono essere i ben noti cicli if, cicli for, gli statement switch, ed i loop come il for ed il while.



Tra i costrutti evidenziati troviamo:

- ```
  push    ebp
  mov     ebp, esp
  ```

  Queste istruzioni Assembly servono per **creare lo stack** di una funzione. Lo stack è una porzione di memoria dedicata per il salvataggio delle variabili locali di una data funzione. Esso viene definito dai puntatori allo stack EBP (Estraction Base Pointer) che punta alla sua base ed ESP (Estraction Stack Pointer) che punta alla cima.

- ```
  push    ecx
  push    0                   ; dwReserved
  push    0                   ; lpdwFlags
  call    ds:InternetGetConnectedState
  ```

  Queste istruzioni indicano il modo in cui la funzione chiamante invia i parametri necessari alla funzione chiamata per poter svolgere il suo compito sullo stack, i parametri vengono inseriti con “**push**” sullo stack prima della **chiamata alla funzione InternetGetConnectedState** che permetterà di determinare se la macchina ha accesso ad internet.

- ```
  cmp     [ebp+var_4], 0
  jz      short loc_40102B
  ```

  Il costrutto noto in questo codice è uno Statement SWITCH, costrutto che viene utilizzato per prendere decisioni in base al valore di una determinata variabile, e utilizza una sintassi simile ad un **ciclo IF (IF-Style)**. La caratteristica principale degli IF-Style è la presenza di una serie di salti condizionali; il blocco dei salti condizionali è composto da un’istruzione “cmp” seguita da un’istruzione di salto, in questo caso “jz loc”, dove “loc” è una locazione di memoria. L’istruzione “cmp” unita all’istruzione “jz loc” controllano l’uguaglianza tra il valore contenuto in [ebp+var_4] e 0, modificando il flag ZF (Zero Flag) in base al risultato; ZF = 0 se gli operandi sono diversi tra loro e il risultato dell’operazione sarà quindi diverso da zero. ZF = 1 quando gli operandi sono uguali tra loro e quindi il risultato dell’operazione sarà uguale a 0, in questo caso “jz loc” salterà alla locazione di memoria specificata.
  ```

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

- Queste istruzioni verranno eseguite se la **risposta del ciclo IF-Style** darà ZF = 0, dirà quindi che è presente una connessione internet.

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

- Queste istruzioni verranno eseguite se la **risposta del ciclo IF-Style** darà ZF = 1 ed è stato effettuato il salto con "jz loc", dirà quindi che non è presente una connessione internet.

```
mov     esp, ebp
pop     ebp
```

- Queste istruzioni sono utilizzate per **rimuovere lo stack** una volta che la funzione ha terminato il suo compito.
  Quando viene chiamata una nuova funzione si crea un nuovo stack in memoria. Tuttavia, quando la funzione chiamata finisce il suo compito, bisognerà eliminare il suo stack e le sue variabili locali non più necessarie per riportare l'esecuzione alla funzione chiamante.
  Questa pratica viene detta pulizia dello stack.

4. **Ipotizzare il comportamento della funzionalità implementata:**
   In seguito alle nostre analisi delle librerie possiamo ipotizzare che il malware ricada nella categoria delle **backdoor**, tipo di malware che utilizza costrutti di tipo "switch" per consentire di eseguire localmente alla macchina una serie di azioni o comandi basati sul valore di una variabile, oppure che sia un **downloader**, un malware che contatta un dominio per scaricare un altro file eseguibile, ad esempio altri malware.
   Sembra che il malware invii una chiamata alla funzione InternetGetConnectedState, funzione che restituisce True o False in base all'esistenza di una connessione e ne controlla quindi la presenza o meno. Tramite il ciclo IF-Style avverrà il controllo del risultato della funzione che se sarà uguale a 0 (**ZF = 1**) darà "**Error 1.1: No Internet**" e terminerà l'esecuzione, mentre se il risultato della funzione sarà diverso da 0 (**ZF = 0**) darà "**Success: Internet Connection**" e quindi terminerà l'esecuzione.

5. **Fare una tabella con significato delle singole righe del codice Assembly.**

CODICE	DESCRIZIONE
Push ebp	Inserisce il valore del registro ebp nello stack
Mov ebp, esp	Sposta il valore del registro esp nel registro ebp
Push ecx	Inserisce il valore del registro ecx nello stack

Push 0 ;dwReserved	Inserisce il valore 0 in cima allo stack in riferimento a dwReserved
Push 0 ;lpdwFlags	Inserisce il valore 0 in cima allo stack in riferimento a lpdwFlags
Call ds:InternetGetConnectedState	Chiama la funzione InternetGetConnectedState
Mov [ebp+var_4], eax	Sposta il valore del registro eax nella posizione di memoria di [epb+var_4]
Cmp [ebp+var_4], 0	Confronta il valore 0 con quello contenuto in [ebp+var_4]
Jz short loc_40102B	Esegue il salto condizionale alla locazione di memoria loc_40102B se ZF = 1
Push offset aSuccessInternet ; "Success: Internet Connection\n"	Carica un indirizzo di memoria che punta alla stringa di testo che verrà stampata a schermo
Call sub_40117F	Chiama la stringa di codice 40117F
Add esp, 4	Aggiunge 4 al valore del registro esp, pointer utilizzato per tenere traccia della posizione in stack
Jmp short loc_40103A	Esegue un salto non condizionale alla locazione di memoria loc_40103A
Loc_40102B: ; "Error 1.1: No Internet\n"	Indirizzo di memoria che punta alla stringa di testo che verrà stampata a schermo
Push offset aError1_1NoInte	Indirizzo di memoria che punta alla stringa precedente
Call sub_40117F	Chiama la stringa di codice 40117F
Add esp, 4	Aggiunge 4 al valore del registro esp, pointer utilizzato per tenere traccia della posizione in stack
Xor eax, eax	Utilizza l'operatore XOR per impostare il registro eax a 0
Loc_40103A	Indirizzo di memoria che viene puntato dal salto non condizionale precedente
Mov esp, ebp	Sposta il valore del registro ebp nel registro esp
Pop ebp	Rimuove il valore in cima allo stack
Retn	La funzione torna al punto di chiamata
Sub 401000 endp	Termina la procedura