**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:

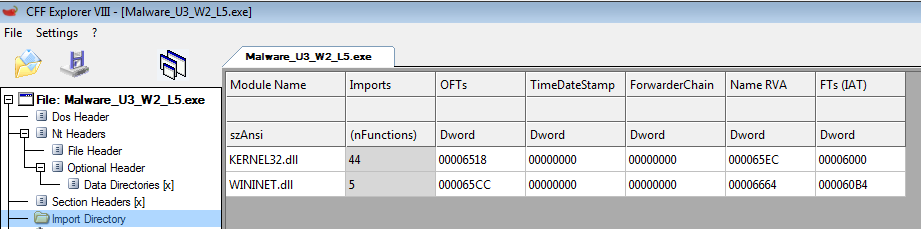
1. Quali librerie vengono importate dal file eseguibile?
2. Quali sono le sezioni di cui si compone il file eseguibile del malware?

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

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

1.---------------------------------------------------------------------------------------------------------------------------------------------------------------|

Apriamo il malware da analizzare con CFF Explorer e andiamo nella sezione apposita per vedere quante e quali tipi di librerie importa il suddetto malware:

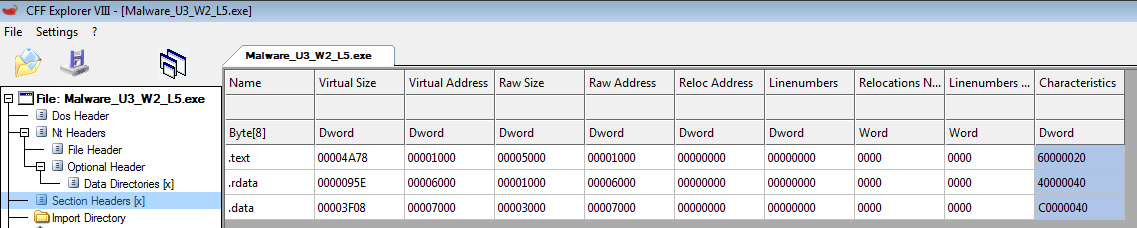


Come si può vedere dall’immagine ce ne sono due, a noi già note:

* **KERNEL32.DLL** 🡪 è una delle librerie di sistema principali di Microsoft Windows, fornisce le *funzioni di base necessarie per molte operazioni di sistema e per l'interazione delle applicazioni utente con il sistema operativo*. Le funzioni contenute in questa libreria coprono una vasta gamma di operazioni di sistema, tra cui gestione della memoria, gestione dei file e delle directory, operazioni di input/output, gestione dei processi e dei thread, sincronizzazione, gestione degli errori, e altro ancora;
* **WININET.dll 🡪** fornisce *funzionalità per la comunicazione in rete e l'accesso a risorse Internet*. Questa libreria è fondamentale per le applicazioni che necessitano di connettersi a Internet, inviare richieste HTTP, scaricare file da server remoti e altre operazioni di rete.

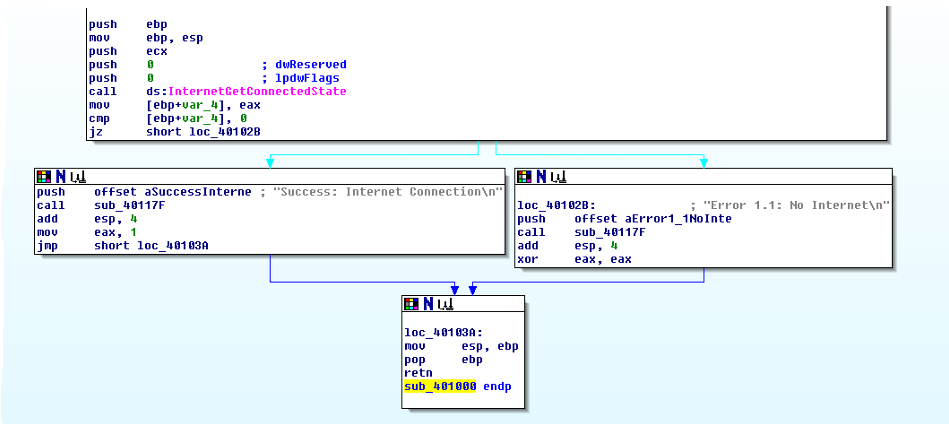
2.------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Per rispondere alla seconda domanda andiamo nella cartella delle sezioni, anche in questo caso ne abbiamo tre già note:

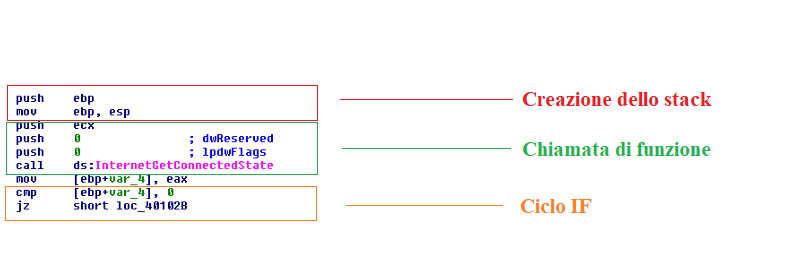


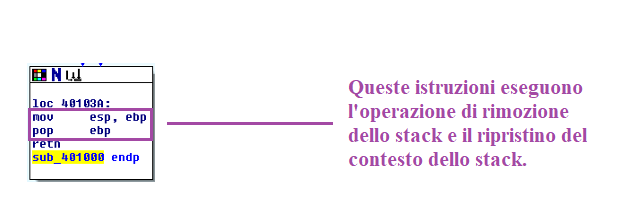
* **.text** 🡪 questa sezione contiene il codice eseguibile del programma, ovvero le istruzioni macchina che vengono eseguite dal processore quando il programma viene avviato;
* **.rdata** 🡪 include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall’eseguibile; questa sezione contiene principalmente dati "read-only" (sola lettura) per proteggere i dati contenuti al suo interno dall'essere modificati accidentalmente durante l'esecuzione del programma;
* **.data** 🡪 contiene tipicamente i dati / le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma. Questa sezione contiene variabili globali e dati che possono essere modificati durante l'esecuzione del programma.

3.-------------------------------------------------------------------------------------------------------------------------------------------------------------------|



**Costrutti noti:**

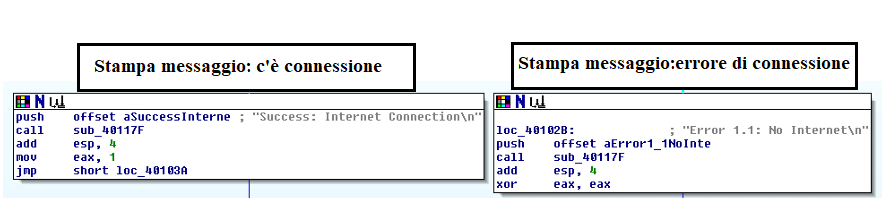




4.---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Dall'analisi del codice, sembra che questo programma stia chiamando la funzione **InternetGetConnectedState** per verificare se la macchina è connessa a Internet.

Se il valore di ritorno di questa funzione è zero, il programma salta a **loc\_40102B**, altrimenti continua l'esecuzione sequenziale.



5.---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Ecco una spiegazione per ogni riga di codice assembly:

1. **push ebp**: Questa istruzione salva il valore del registro di base (ebp) nello stack. È il primo passo tipico nella creazione di un frame di stack per una funzione.
2. **mov ebp, esp**: Questa istruzione copia il valore dello stack pointer (esp) nel registro di base (ebp), creando così un nuovo frame di stack per la funzione corrente.
3. **push ecx**: Questa istruzione salva il valore del registro ecx nello stack.
4. **push 0 ; dwReserved**: Questa istruzione spinge il valore 0 nello stack. Potrebbe essere un parametro da passare alla funzione chiamata successivamente.
5. **push 0 ; lpdwFlags**: Questa istruzione spinge il valore 0 nello stack. Anch'esso potrebbe essere un parametro da passare alla funzione chiamata successivamente.
6. **call ds: InternetGetConnectedState**: Questa istruzione chiama la funzione **InternetGetConnectedState** situata nella sezione dati (ds) del programma.
7. **mov [ebp+var\_4], eax**: Questa istruzione salva il valore di ritorno della funzione **InternetGetConnectedState** nella variabile **[ebp+var\_4]**.
8. **cmp [ebp+var\_4], 0**: Questa istruzione compara il valore salvato in **[ebp+var\_4]** con 0.
9. **jz short loc\_40102B**: Questa istruzione salta a **loc\_40102B** se il confronto precedente ha dato zero, altrimenti continua l'esecuzione sequenzialmente.
10. **push offset asuccessInterne ; "Success Internet Connection\n"**: Questa istruzione mette l'indirizzo della stringa "Success Internet Connection\n" nello stack.
11. **call sub\_40105F**: Questa istruzione chiama la subroutine **sub\_40105F**, presumibilmente per stampare il messaggio sullo schermo.
12. **add esp, 4**: Questa istruzione libera lo spazio nello stack riservato per la stringa, spostando lo stack pointer verso l'alto.
13. **mov eax, 1**: Questa istruzione assegna il valore 1 al registro eax.
14. **jmp short loc\_40103A**: Questa istruzione salta a **loc\_40103A** senza condizioni.
15. **push offset aError1\_1NoInte ; "Error 1.1: No Internet\n"**: Questa istruzione mette l'indirizzo della stringa "Error 1.1: No Internet\n" nello stack.
16. **call sub\_40117F**: Questa istruzione chiama la subroutine **sub\_40117F**, presumibilmente per gestire l'errore di mancanza di connessione internet.
17. **add esp, 4**: Questa istruzione libera lo spazio nello stack riservato per la stringa, spostando lo stack pointer verso l'alto.
18. **xor eax, eax**: Questa istruzione esegue un'operazione XOR tra eax ed eax, che equivale a azzerare il registro eax.
19. **mov esp, ebp**: Questa istruzione ripristina il puntatore dello stack al suo valore precedente prima di entrare nella funzione.
20. **pop ebp**: Questa istruzione ripristina il registro di base (ebp) al suo valore precedente prima di entrare nella funzione.
21. **retn**: Questa istruzione esegue un ritorno dalla funzione corrente.
22. **sub\_401000 endp**: Questo indica la fine della procedura **sub\_401000**.