S10L5

Consegna:

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?

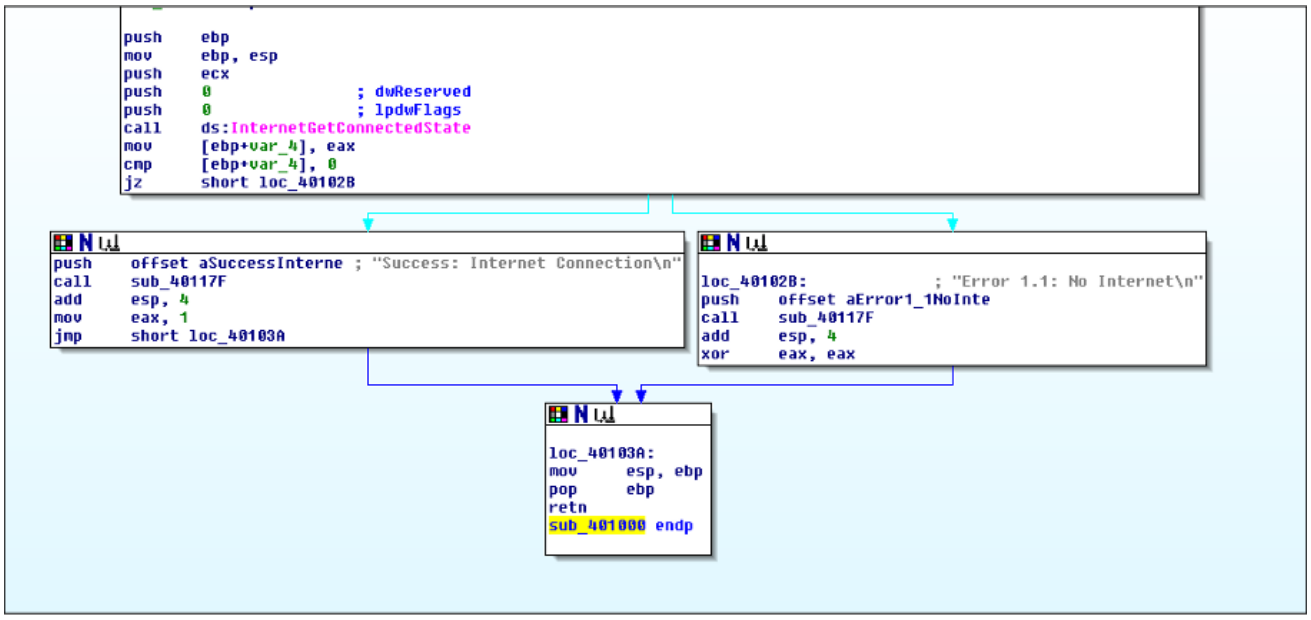
2. Quali sono le sezioni di cui si compone il file eseguibile del malware? Con riferimento alla figura in slide

Con riferimento all'immagine fornita:

3. Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti)

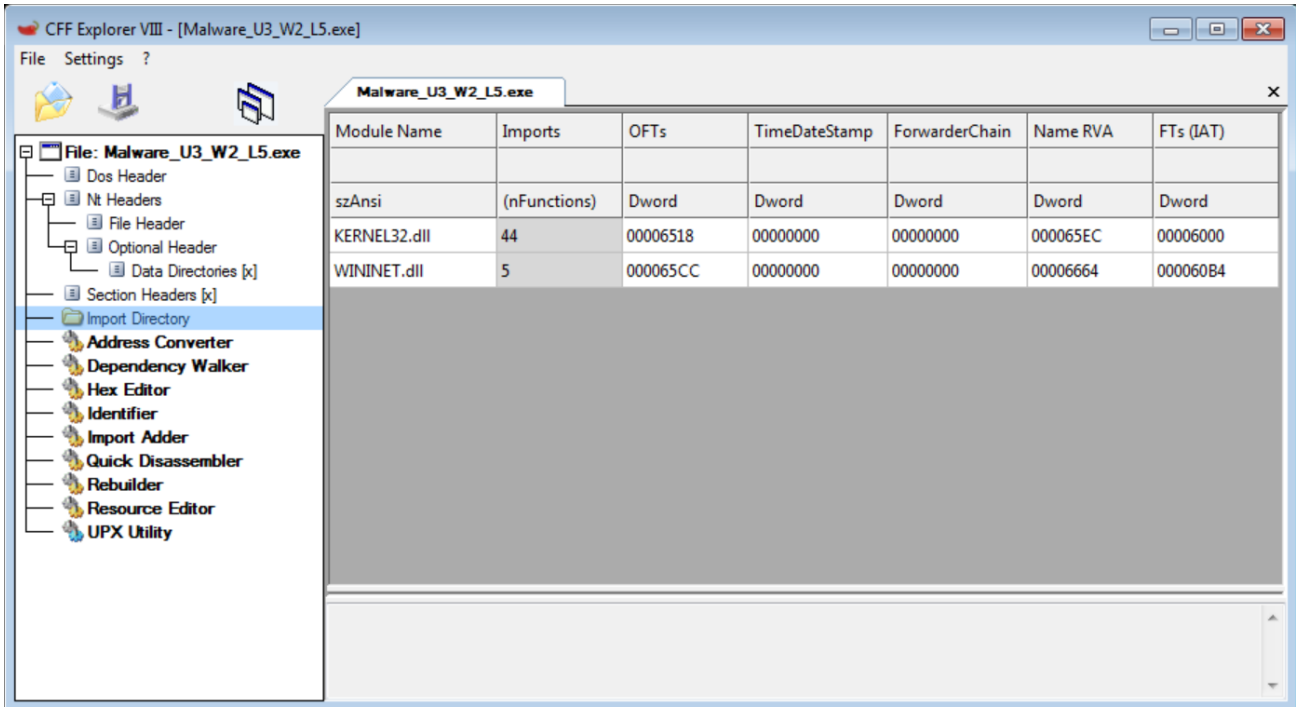
4. Ipotizzare il comportamento della funzionalità implementata

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



Esecuzione:

1) Per capire quali librerie vengono importate da un file eseguibile bisogna controllare su **CFF Explorer**. Infatti, sarà possibile importare il malware per avere una panoramica generale su che cosa il file in questione possa fare. Dopo aver fatto ciò, per controllare le librerie di cui necessita l'eseguibile, basterà accedere alla sezione "Import Directory" dove sarà possibile consultare anche le funzioni importate dalle librerie.



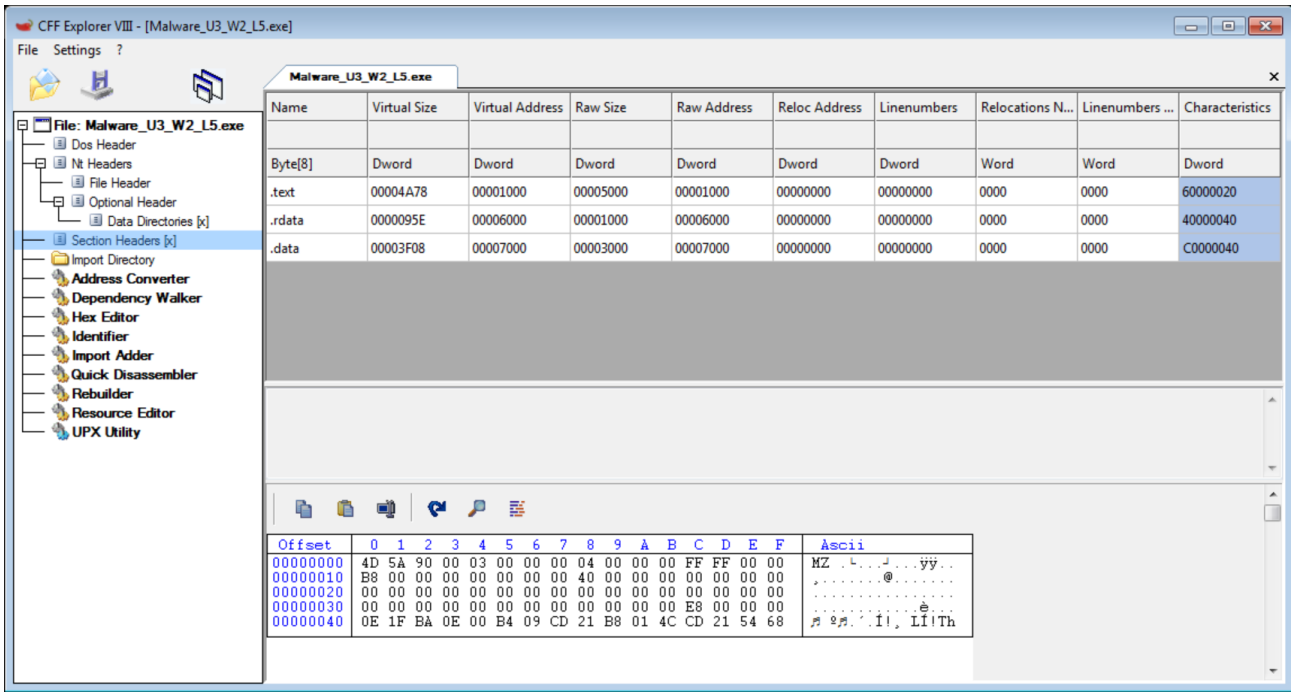
Come si può evincere dall'immagine, le librerie consultate sono 2:

-**KERNEL32.dll**: questa libreria contiene funzioni che facilitano l'interazione con il sistema operativo attaccato (andando spesso a manipolare la gestione dei file o della memoria).

-**WININET.dll**: questa libreria contiene funzioni apposite per implementare alcuni protocolli come HTTP, FTP, NTP…

2) Similmente alla prima parte della consegna, per controllare le sezioni di cui si compone il file in questione bisognerà utilizzare CFF Explorer per accedere alla sezione "Section

Headers”. Così facendo sarà disponibile una panoramica generale sulle sezioni del codice.



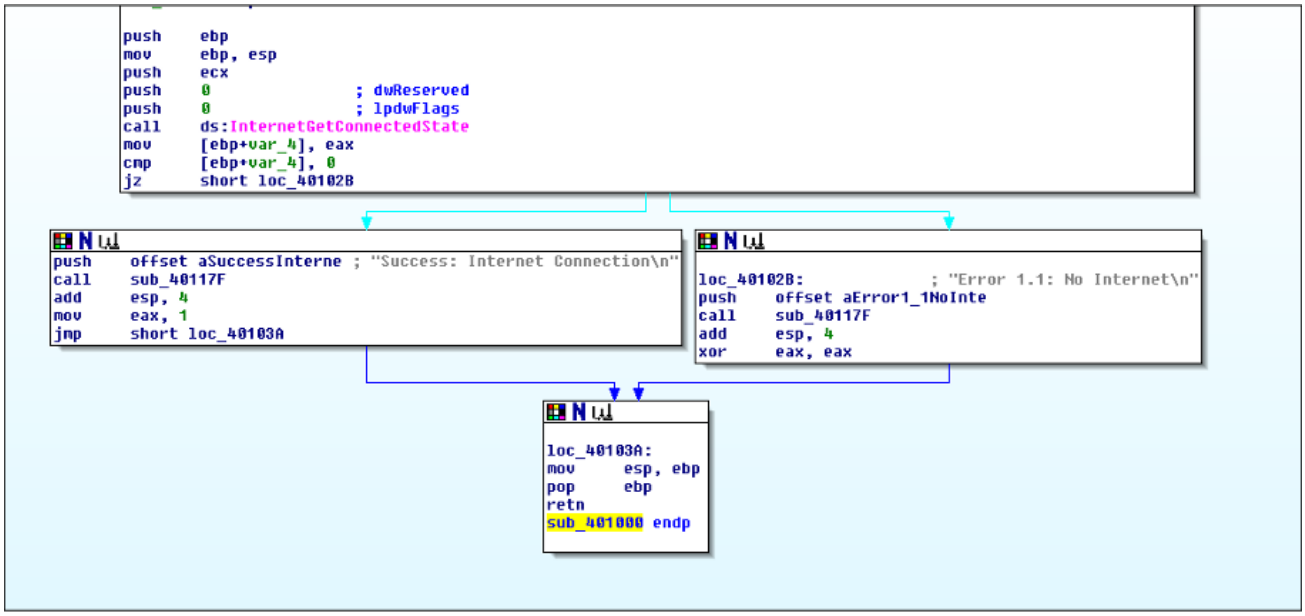
Come viene illustrato le sezioni che compongono il codice sono 3:

-**.text**: questa sezione contiene le informazioni che verranno processate dalla CPU per poter eseguire le istruzioni che le sono state impartite.

-**.rdata**: questa sezione è relativa alla documentazione necessaria per poter eseguire le funzioni delle librerie.

-**.data**: questa sezione è necessaria per poter contenere tutte le variabili globali che verranno utilizzate per l’esecuzione del codice.

3) Questa parte dell’esercizio richiede di analizzare il codice assembly dato dalla consegna per poterne identificare dei costrutti.



Dall'immagine si possono riconoscere alcuni costrutti come:

-**Creazione di uno stack** (prime due righe del primo blocco)

-**Chiamata di funzione** (successive quattro righedel primo blocco)

-**Blocco IF** (nelle ultime tre righe del primo blocco)

-**Chiusura dello stack** (prime tre righe dell'ultimo blocco)

4) Il codice è scritto in linguaggio assembly x86 e, per via del nome della funzione chiamata, si può presupporre che vada a controllare lo stato della connessione ad Internet. Il primo blocco, come già mostrato in precedenza, nelle prime due righe si occupa di creare uno stack, che serve per poter contenere le variabili che verranno poi inserite per poter svolgere la funzione "InternetGetConnectionState" chiamata nella sesta riga. Il

codice viene poi diviso da un costrutto IF reso tramite l'attribuzione del valore del registro EAX alla variabile [ebp+var_4] che viene poi confrontata a 0 grazie all'istruzione cmp, e, in base al risultato di questa comparazione, viene saltata una porzione di codice grazie all'istruzione jz, che esegue un salto in base all'attivazione dello Zero Flag (un flag che assume il valore di 1 solamente se un numero risulta essere uguale a 0). Il blocco a sinistra si occupa di stampare un messaggio di conferma della connessione, per poi passare alla rimozione dello stack, non prima di aver chiamato la funzione allocata nella porzione di memoria 40117F, aver aumentato il valore di ESP di 4 e EAX di 1. Il blocco di destra stampa una stringa che avvisa della fallita connessione ad Internet, poi chiama la funzione 40117F, dopo di che aumenta il valore di ESP di 4 e confronta tramite l'operatore logico XOR (che risulta vero solo quando i suoi operandi sono tutti diversi) EAX ed ESP. Infine, l'ultimo blocco si occupa della pulizia dello stack terminando la funzione.

5)

Istruzione	Significato
push ebp	Salva il valore del registro come base dello stack
mov ebp, esp	Imposta lo stack per la funzione da chiamare creando la cima dello stack
push ecx	Salva il valore del registro ECX nello stack
push 0	Salva il valore 0 nello stack
call ds:InternetGetConnectedState	Chiama la funzione "InternetGetConnectionState"
mov [ebp+var_4], eax	Memorizza il risultato della funzione nella variabile var_4
cmp [ebp+var_4], 0	Compara La variabile Var_4 con 0
jz short loc_40102B	Salta alla locazione di memoria specificata se il risultato è uguale a 0
push offset aSuccessInterne	Mette l'indirizzo di memoria del messaggio nello stack
call sub_40117F	Chiama la funzione nell'indirizzo di memoria specificato
add esp, 4	Aumenta di 4 il valore di ESP
mov eax, 1	Attribuisce 1 al valore di EAX
jmp short loc_40103A	Salta alla locazione di memoria specificata
push offset aError1_1NoInte	Scrive il messaggio di Errore
call sub_40117F	Chiama la funzione che risiede all'indirizzo di memoria specificato
add esp, 4	Aumenta di 4 il valore di ESP
xor esp, ebp	Esegue l'operatore XOR ai valori riportato di seguito
pop ebp	Ripulisce lo stack
retn	rimuove lo stack
	restituisce il controllo al chiamante