

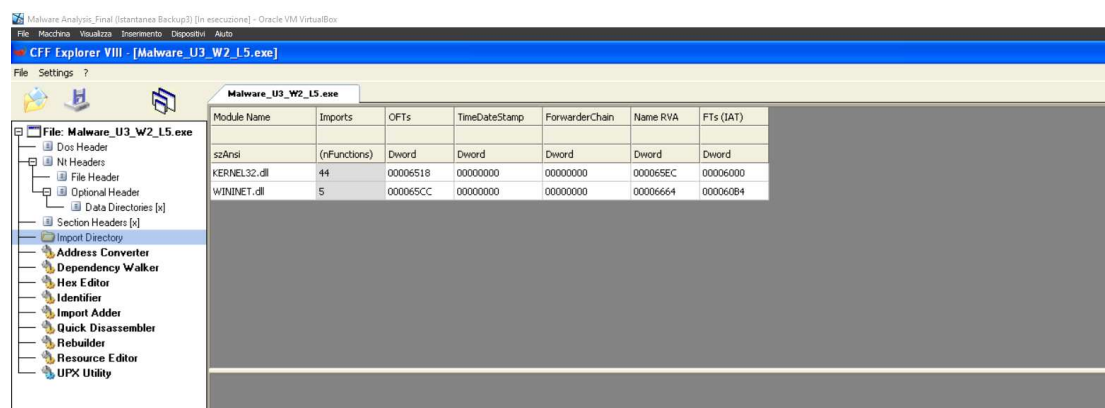
S10/L5 - Analisi statica e dinamica: Un approccio pratico

La prima parte dell'esercizio di oggi consiste nell'effuare un'analisi statica di un malware presente sulla nostra macchina virtuale, il nome del file è Malware_U3_W2_L5.

La traccia chiede di trovare le librerie che il malware importa e quali sono le sezioni di cui si compone il file eseguibile del malware.

1.Librerie importate

Per cercare le librerie importate dal malware ho utilizzato il tool CFF Explorer, che una volta inserito il file al suo interno mi dice chiaramente quali sono come in figura sotto:



Questo malware importa due librerie: KERNEL32.dll e WININET.dll.

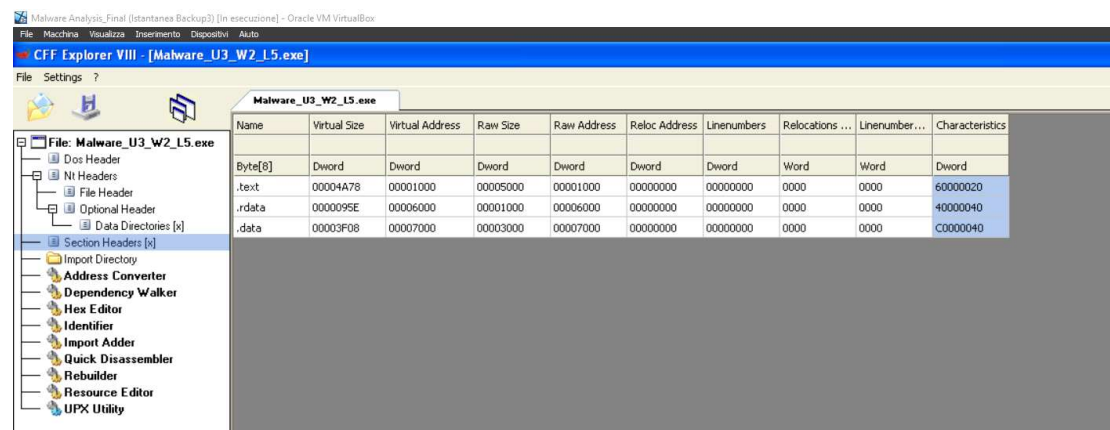
KERNERL32.dll è una delle librerie di sistema principali in ambienti Windows, contiene molte funzioni di base del sistema operativo di Windows e fa parte del kernel. Questa libreria è essenziale per il funzionamento di Windows dato che la maggior parte delle applicazioni dipendono da questa libreria.

WININET.dll è una libreria di sistema in ambienti Windows. Questa DLL è specificamente associata alle funzionalità di Internet e fornisce un'interfaccia per l'accesso a risorse su Internet. Essa è molto importante per le connessione internet da parte delle applicazioni, quindi senza di essa non si riuscirebbe a utilizzare la rete.

2.Sezioni del malware

Per cercare le sezioni di cui si compone il file in questione ho utilizzato sempre CFF

Explorer come possiamo vedere in figura:



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	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

Le sezioni presenti in questo file sono 3: .text , .rdata e .data

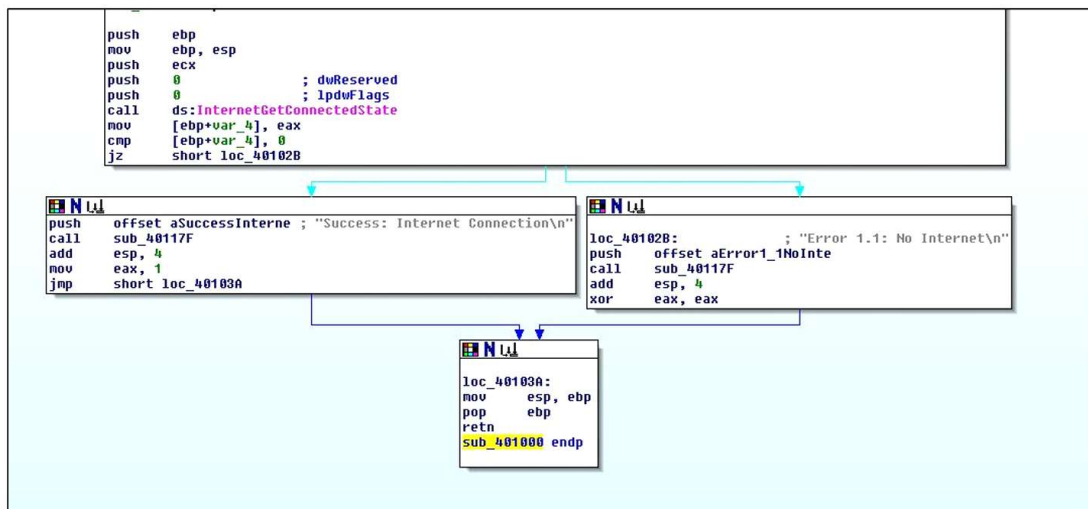
.text Questa sezione è la parte del file eseguibile analizzato che contiene il codice macchina eseguibile, ovvero contiene le istruzioni del programma che saranno eseguite dalla CPU.

.rdata Questa sezione contiene le informazioni delle librerie e delle funzioni importate ed esportate dall'eseguibile, questi dati sono accessibili in modalità di solo lettura.

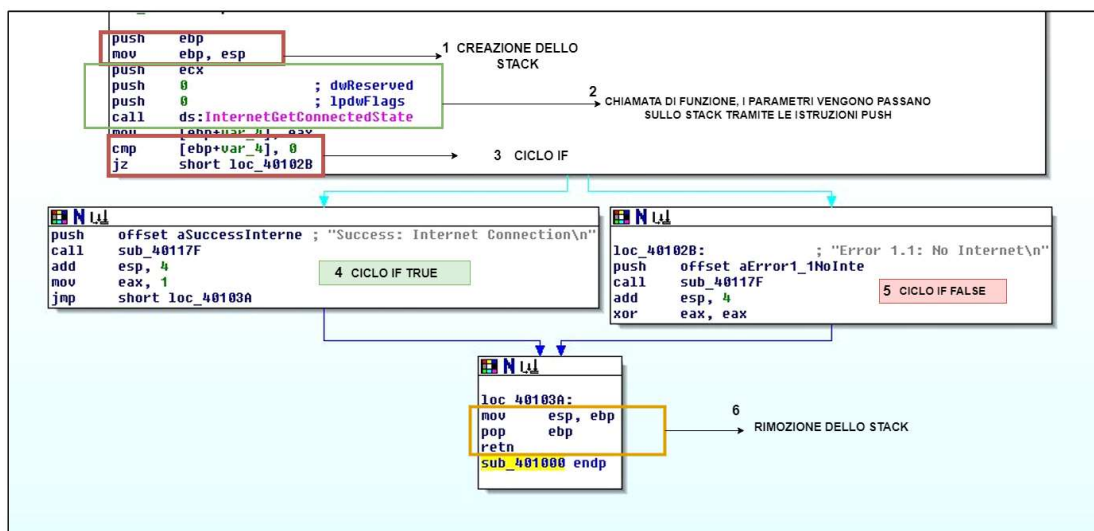
.data Questa sezione contiene i dati globali inizializzati che possono essere modificati durante l'esecuzione del programma a differenza dall'.rdata. Questa sezione viene utilizzata per salvare le variabili globali di un programma.

Come secondo compito andremmo ad analizzare un codice assembly in figura sotto e dovremmo:

1. Identificare i costrutti noti
2. Ipotizzare il comportamento della funzionalità implementata



1. Identificare i costrutti noti



1 CREAZIONE DELLO STACK

2 CHIAMATA DI FUNZIONE, I PARAMETRI VENGONO PASSANO SULLO STACK TRAMITE LE ISTRUZIONI PUSH

3 CICLO IF

4 CICLO IF TRUE

5 CICLO IF FALSE

6 RIMOZIONE DELLO STACK

2. Ipotizzare il comportamento della funzionalità implementata

Questo codice potrebbe essere parte di un malware downloader dato che sembrerebbe che controlli se la macchina vittima ha la connessione ad internet o no

attraverso "InternetGetConnectedState". In base al risultato del ciclo IF vengono stampati due messaggi diversi: in caso il ciclo IF risulti true quindi verificato stampa il messaggio "Success: Internet Connection\n" in caso contrario verrà stampato "Error 1.1: No Internet\n". Alla fine della procedura viene eliminato lo stack creato all'inizio.

BONUS: Fare una tabella con il significato delle singole righe di codice assembly

push ebp	-pusha ebp per la creazione della base dello stack
mov ebp, esp	-copia il valore di esp in ebp per creare un frame di registro
push ecx	-salva il valore di ecx nello stack
push 0; dwReserved	-pusha il valore zero nello stack come parametro dwReserved
push 0 ; lpdwFlags	-pusha il valore zero nello stack come parametro lpdwFlags
call ds:InternetGetConnectedState	-sta chiamando la funzione InternetGetConnectedState
mov [ebp+var_4], eax	-sta copiando il valore contenuto nel registro eax nella memoria indicata da [ebp+var_4]
cmp [ebp+var_4], 0	-confronta il valore memorizzato nella variabile [ebp+var_4] con zero.
jz short loc_40102B	-è un'istruzione di salto condizionale, se il risultato della comparazione precedente è 0, verso loc_40102B
push offset aSuccessInterne	-inserisce l'offset di una stringa, denominata aSuccessInterne ovvero stampa la stringa
call sub_40117F	-chiama una subroutine o una funzione all'indirizzo indicato
add esp, 4	-incrementa il valore di esp di 4 byte
mov eax, 1	-assegna il valore 1 al registro eax
jmp short loc_40103A	-salto "corto" verso l'indirizzo specificato
push offset aError1_NoInte	-stampa la stringa
call sub_40117F	-chiama la subroutin o funzione all'indirizzo
add esp, 4	-incrementa il valore di esp di 4 byte
xor eax, eax	-esegue un'operazione di XOR tra il registro eax e se

stesso, il che comporta che il registro `eax` viene posto a zero.