S10 L5

Report Progetto

GiuliaSalani

INDICE

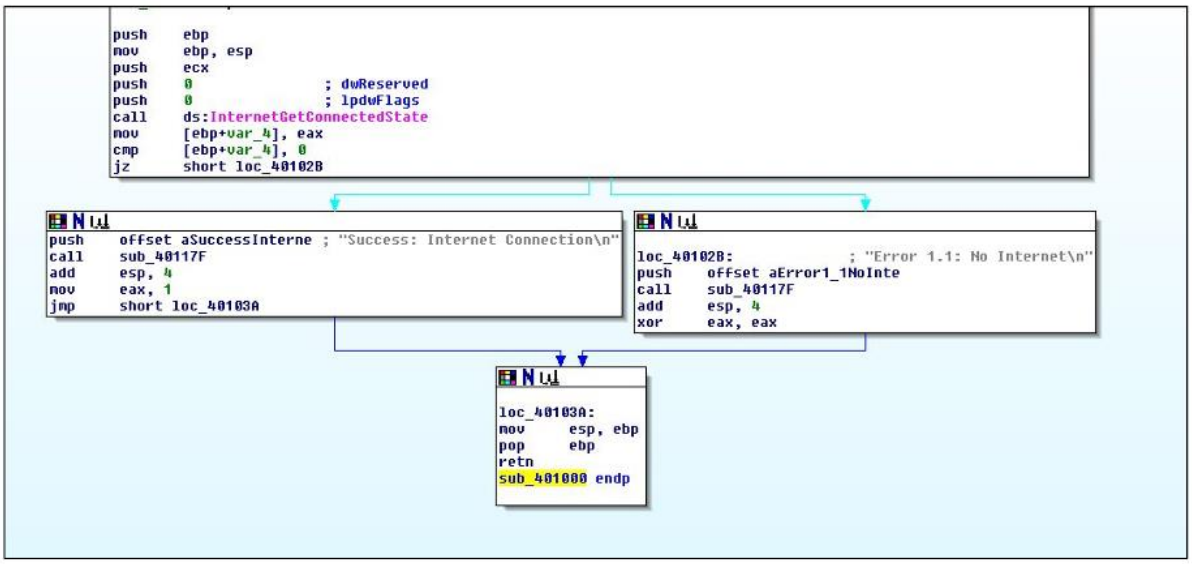
TRACCIA

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 […], risponde ai seguenti quesiti:

1. Identificare i costrutti noti (creazione dello stack, eventuali cicli, costrutti)
2. Ipotizzare il comportamento della funzionalità implementata



ANALISI STATICA BASICA

1. PREPARAZIONE

1.1 ANALISI STATICA BASICA: DEFINIZIONE

L'analisi statica del malware è una tecnica di sicurezza informatica che **esamina il codice sorgente o l'eseguibile di un software dannoso senza eseguirlo**, al fine di identificarne le caratteristiche senza attivare comportamenti dannosi.

L'obiettivo principale dell'analisi statica è **comprendere la struttura e le funzionalità del malware**, identificando firme e comportamenti dannosi, senza esporre il sistema a rischi.

Individuare le librerie è essenziale per capire quali risorse esterne il malware può sfruttare. Questo aiuta a prevedere le potenziali azioni dannose che il malware potrebbe intraprendere.

Analizzare le diverse sezioni del malware consente di comprendere come il codice è strutturato e quali risorse o funzionalità specifiche sono coinvolte in diverse fasi dell'esecuzione.

1.2 CFF EXPLORER: STRUMENTO PER L'ANALISI STATICA

CFF Explorer è uno **strumento avanzato di analisi statica** utilizzato prevalentemente nel campo della sicurezza informatica e del reverse engineering. **Permette agli analisti di esaminare dettagliatamente i file eseguibili, inclusi quelli sospetti di essere malware**.

Fornisce informazioni dettagliate sulla struttura interna di un eseguibile, inclusi i dettagli delle sezioni, le risorse, le librerie collegate e le intestazioni PE (Portable Executable).

Grazie a CFF Explorer, è possibile identificare firme, individuare pattern e comprendere la struttura di un file eseguibile, facilitando così la creazione di firme di rilevamento e la comprensione delle potenziali minacce senza dover eseguire il malware effettivamente.

1.3 EXEINFO PE: STRUMENTO PER L'ANALISI STATICA

EXEinfo PE è uno **strumento di analisi statica specifico per i file eseguibili di Windows** (PE, Portable Executable).

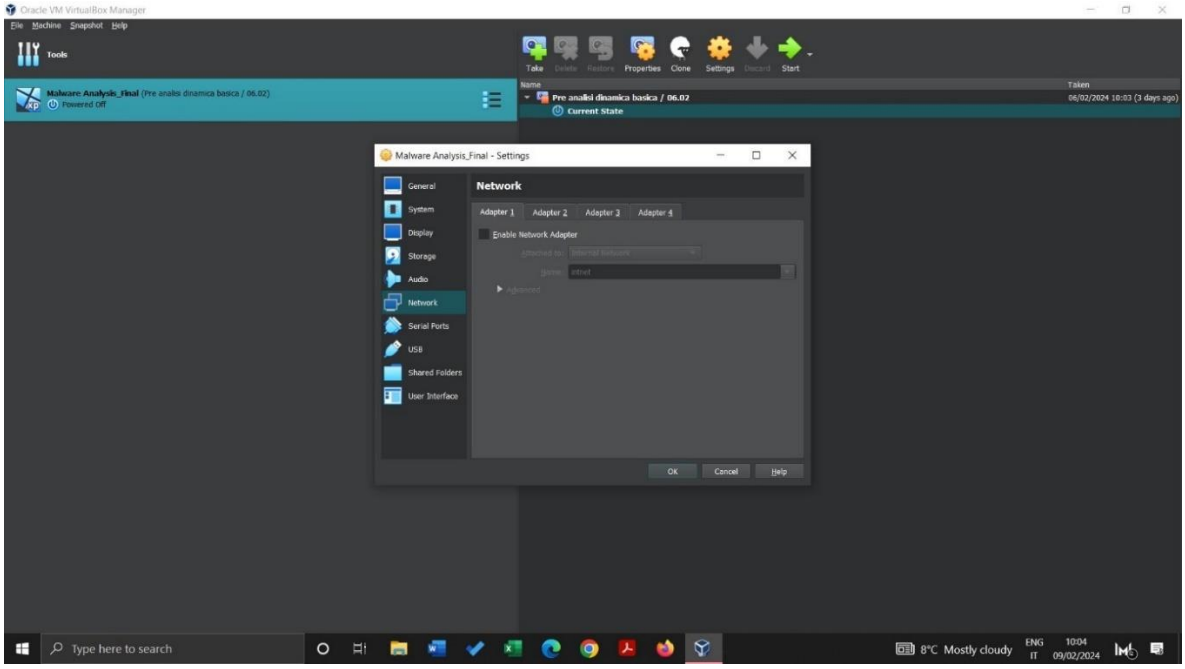
Fornisce informazioni dettagliate sulle caratteristiche di un file eseguibile, inclusi dettagli come firma digitale, compilatore utilizzato, librerie collegate e molti altri attributi.

Facilita il riconoscimento di firme specifiche, la verifica dell'integrità del file e la comprensione preliminare delle caratteristiche del malware, contribuendo così alla sicurezza informatica e al reverse engineering.

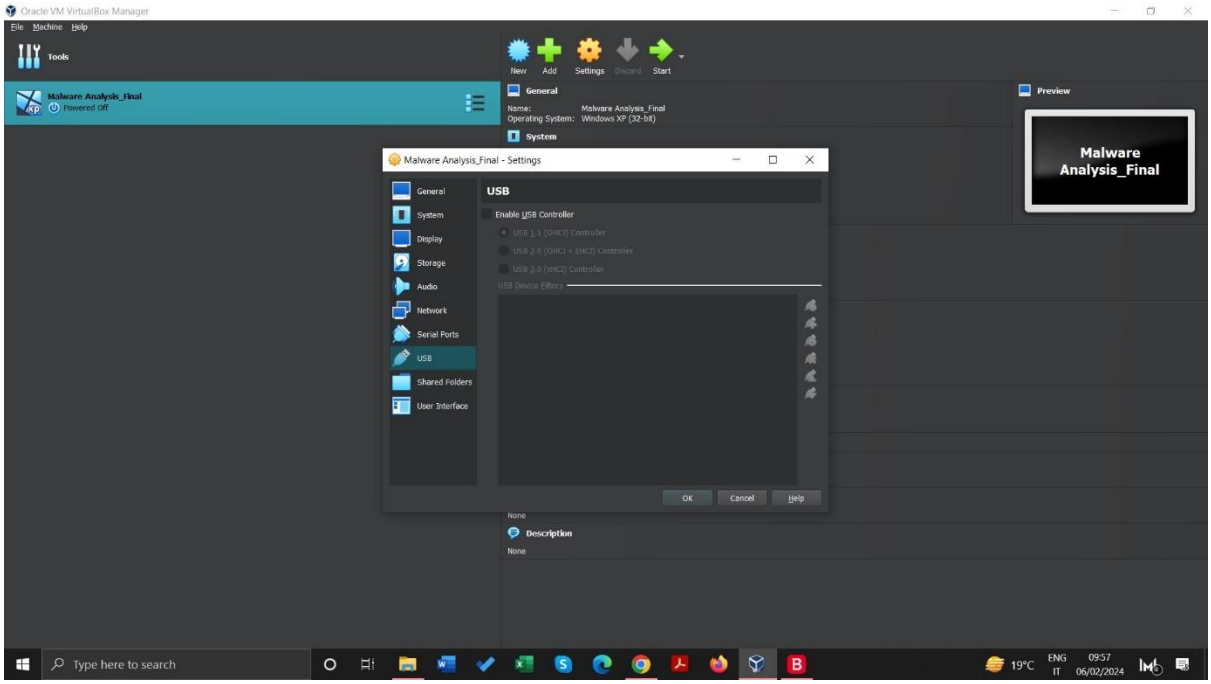
1.4 PREPARAZIONE AMBIENTE: ISTRUZIONI

In questa attività maneggiamo un ambiente che contiene veri e propri malware, dunque la corretta preparazione di un ambiente sicuro è un passaggio fondamentale. Occorre prendere una serie di misure per tutelare la macchina ospite.

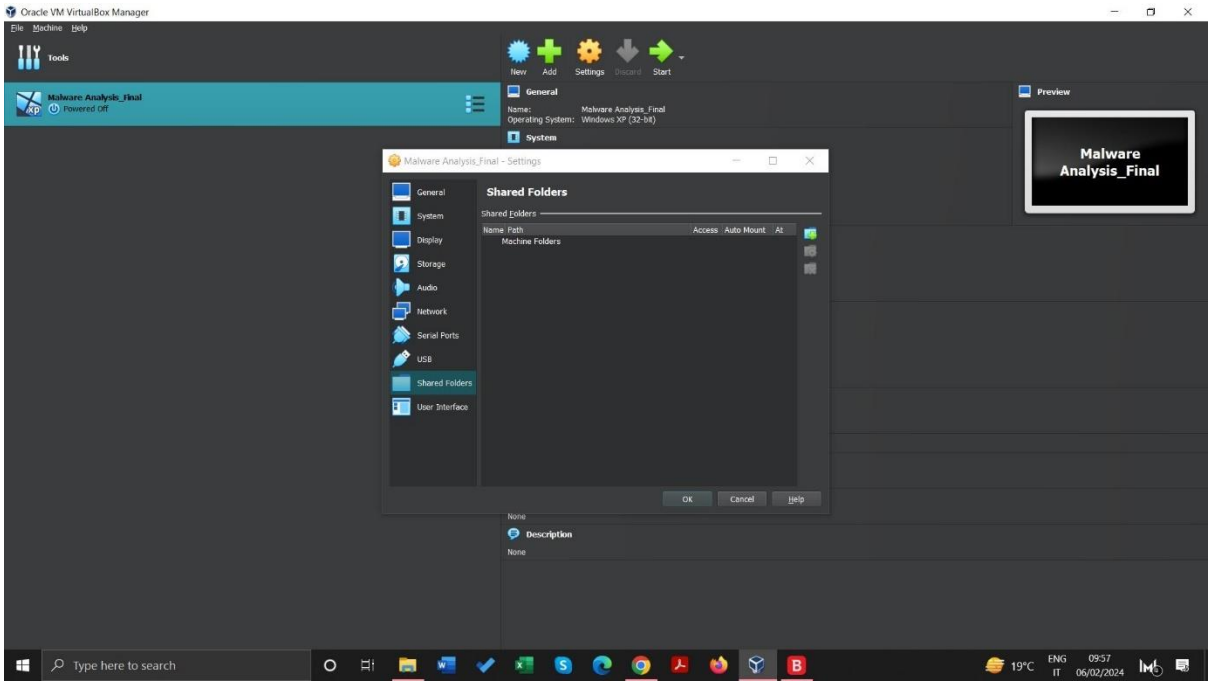
Apriamo Oracle VM Virtualbox, selezioniamo la nostra macchina virtuale e poi il pulsante "Settings". Spostiamoci sulla tab Network. Disattiviamo tutte le NIC:



Spostiamoci ora su USB e assicuriamoci che l'opzione sia disabilitata, per evitare qualsiasi propagazione che utilizzi questo mezzo:



Scendiamo su Shared Folders e assicuriamoci che non vi siano cartelle condivise, sempre per evitare di mantenere punti scoperti che il malware potrebbe utilizzare per diffondersi:

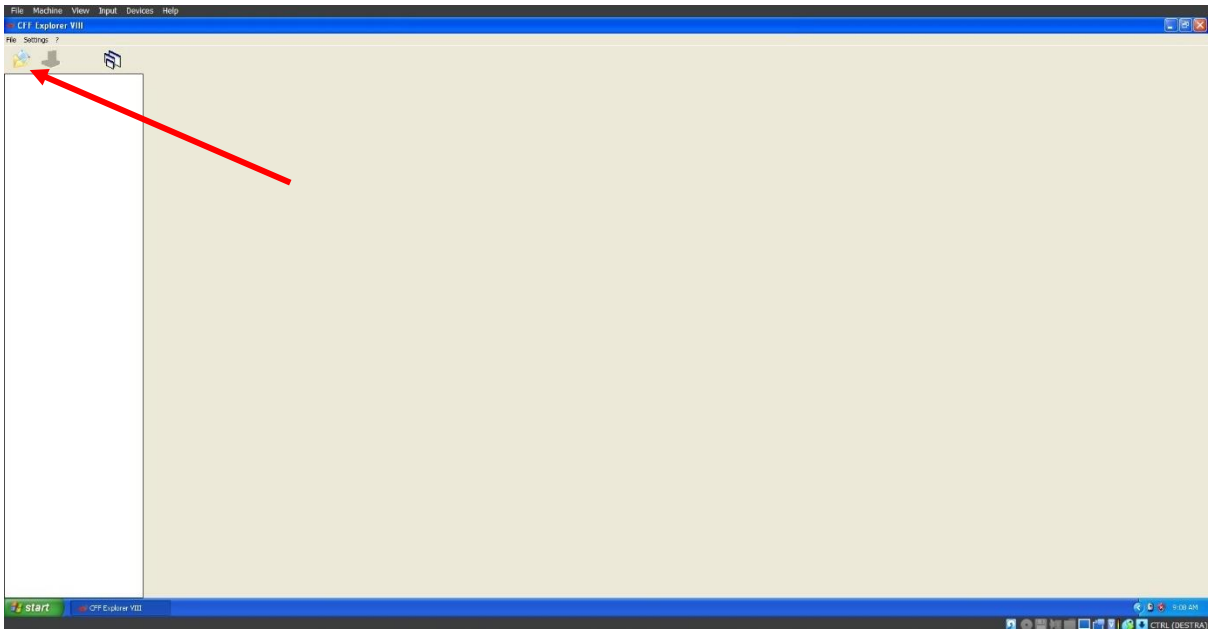


Ora possiamo avviare la macchina.

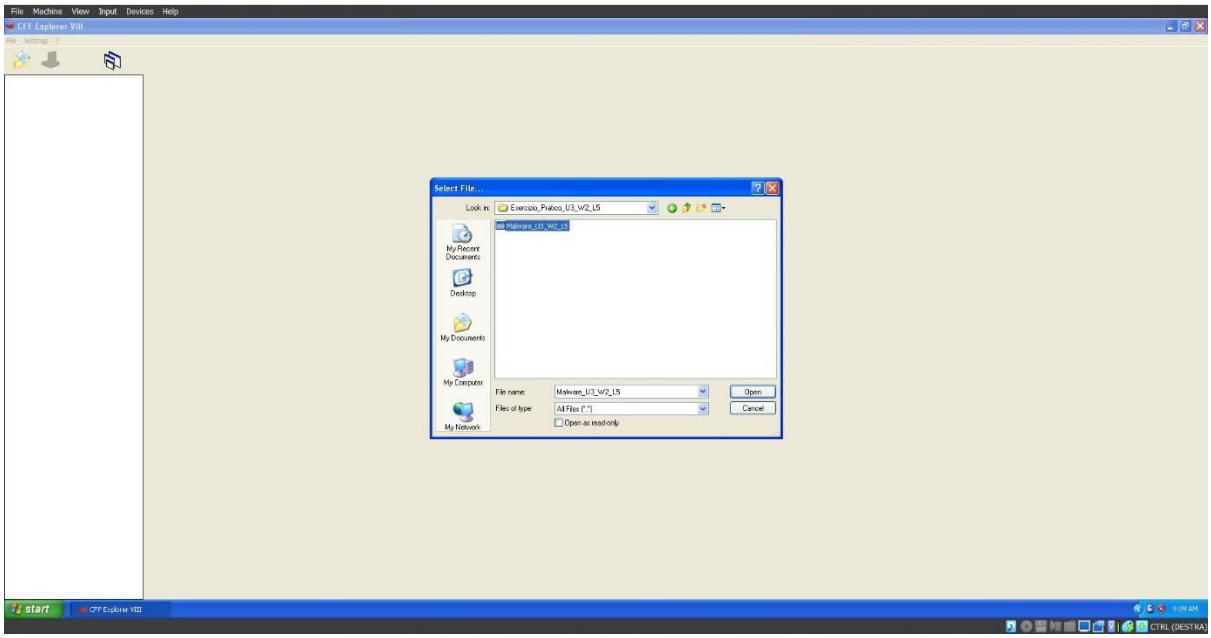
2. CFF EXPLORER: ESECUZIONE

Individuiamo il programma sul desktop della nostra macchina virtuale:

All'apertura, il programma si presenta così. Selezioniamo il pulsante a forma di cartella in alto a sinistra:



Selezioniamo il file che intendiamo analizzare e clicchiamo su open:

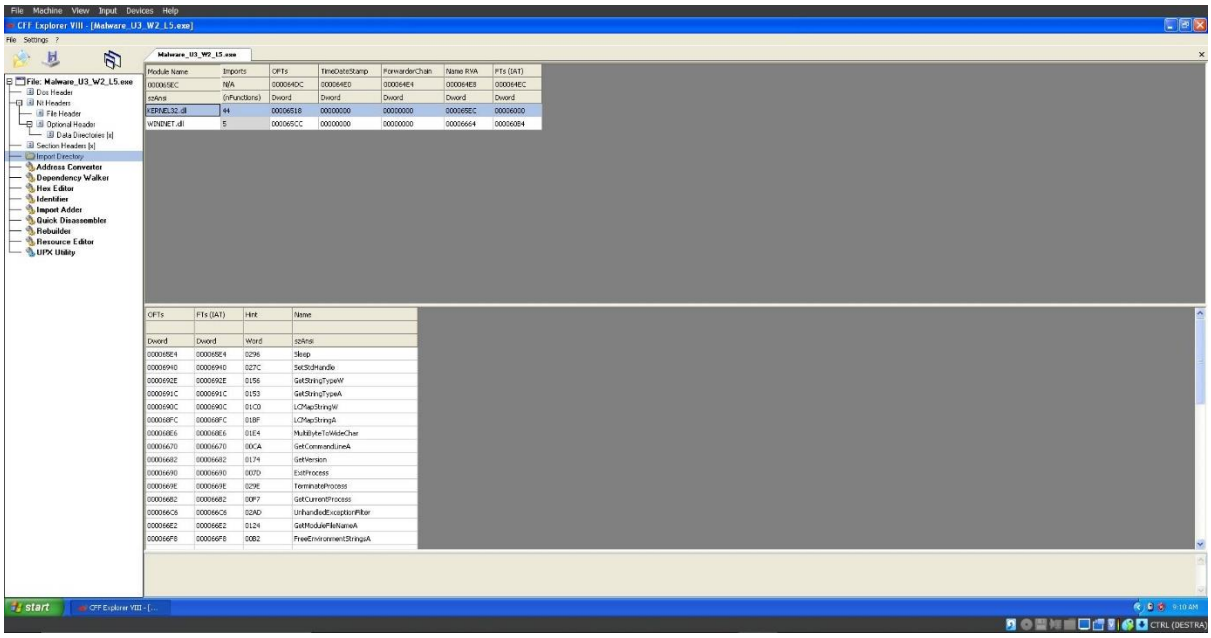


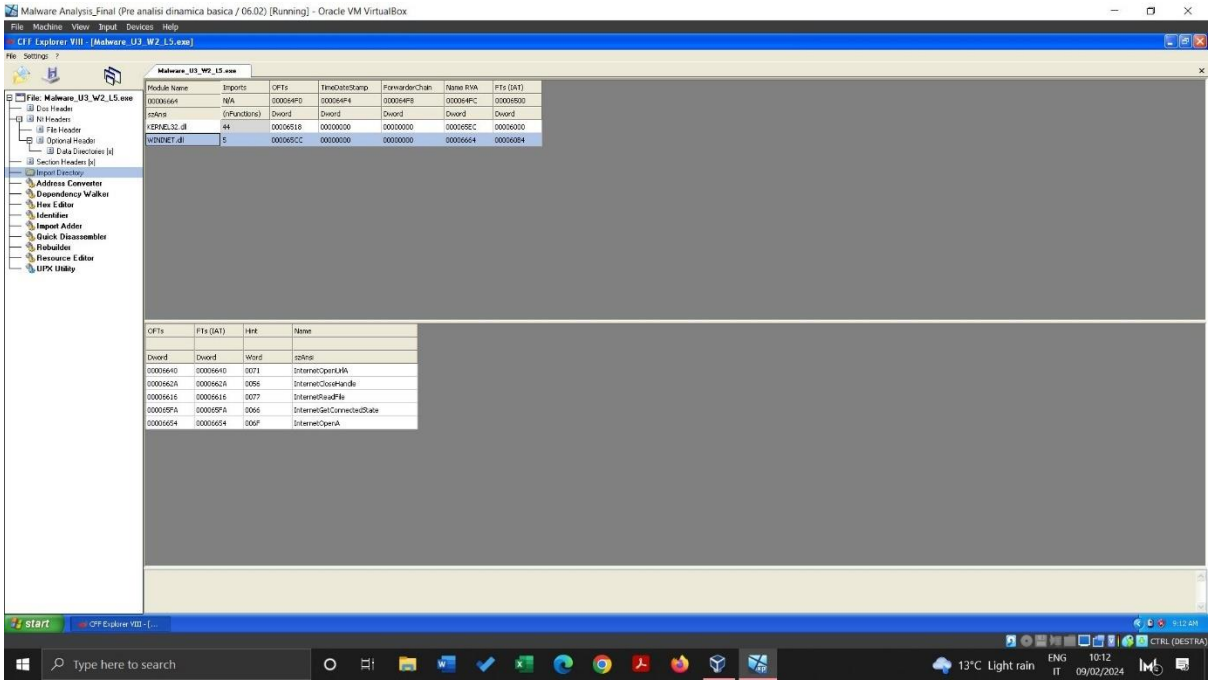
Ora può incominciare l'analisi.

2.1 LIBRERIE

Dalla lista presente sulla sinistra, selezioniamo "Import Directory". In questa parte del software vengono mostrate le librerie caricate dal malware.

Notiamo subito che le librerie sono due: KERNEL32.dll e WININET.dll:





KERNEL32.dll

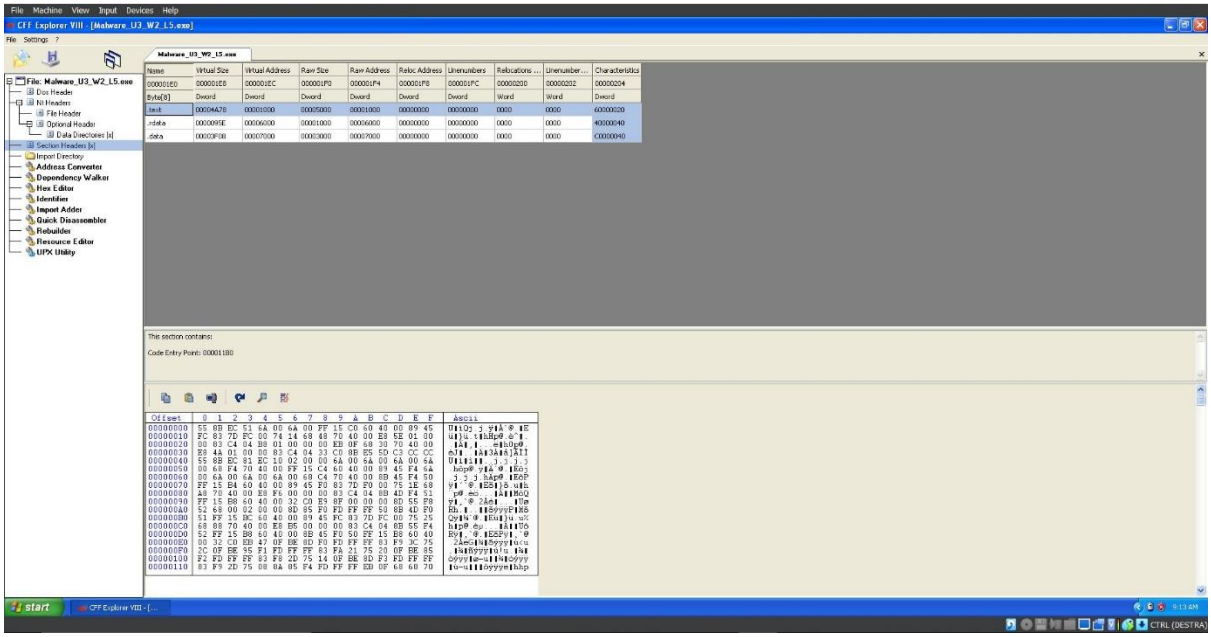
kernel32.dll è una libreria di sistema fondamentale in ambienti Windows che fornisce molte funzioni di basso livello necessarie per la gestione dei processi, della memoria, degli input/output, dei file e altro ancora. Nei malware, **kernel32.dll può essere sfruttata per nascondere processi dannosi, manipolare la memoria del sistema e interagire con il sistema operativo**. Il malware può caricare dinamicamente funzioni da questa libreria per eseguire operazioni complesse in modo discreto.

WININET.dll

wininet.dll è una libreria di sistema di Windows specializzata nelle operazioni di networking e accesso a Internet. Essa fornisce funzionalità per la gestione delle connessioni HTTP, FTP e altri protocolli Internet. Nei malware, **wininet.dll può essere sfruttata per eseguire attività legate alla rete, come il download di file dannosi, l'invio di informazioni a server remoti o il controllo di comunicazioni Internet**. I malware spesso utilizzano questa libreria per stabilire connessioni di rete e interagire con risorse online.

2.2 SEZIONI

Per accedere alla parte di software che analizza le sezioni del malware, clicchiamo sulla riga Section Headers (subito sopra la riga che avevamo selezionato prima):



Possiamo notare che le sezioni sono 3:

1. .text
2. .rdata
3. .data

Visto che il nome non è camuffato, vediamo cosa contengono generalmente queste sezioni:

.text

Contiene il codice eseguibile del programma. Nel contesto di un malware, la sezione .text contiene le istruzioni effettive del malware, inclusi i payload dannosi e le istruzioni per l'esecuzione di attività malevole.

.rdata

La sezione .rdata contiene dati di sola lettura, come stringhe costanti e altre informazioni che non devono essere modificate durante l'esecuzione. Nei malware, questa sezione potrebbe includere informazioni utilizzate internamente dal malware, come stringhe di testo o costanti utilizzate per nascondere o crittografare il codice dannoso.

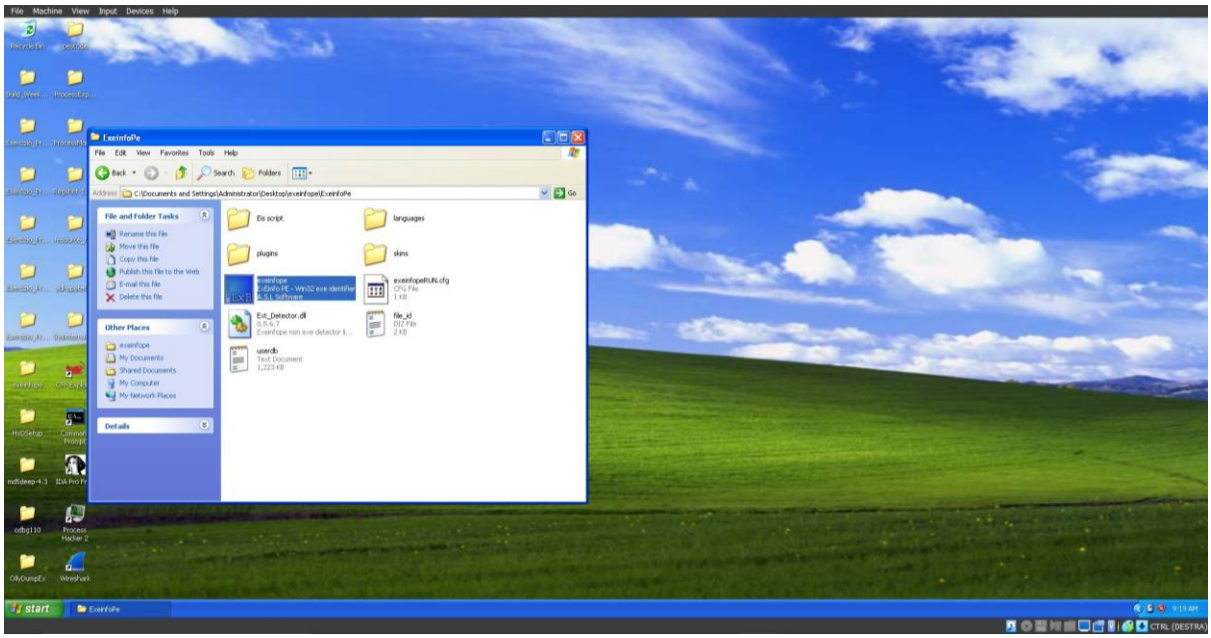
.data

La sezione .data contiene dati che possono essere modificati durante l'esecuzione del programma. Nei malware, questa sezione potrebbe includere variabili e dati necessari per l'esecuzione delle funzionalità del malware. Ad esempio, potrebbe contenere informazioni sullo stato dell'infezione o dati rubati durante l'attività dannosa.

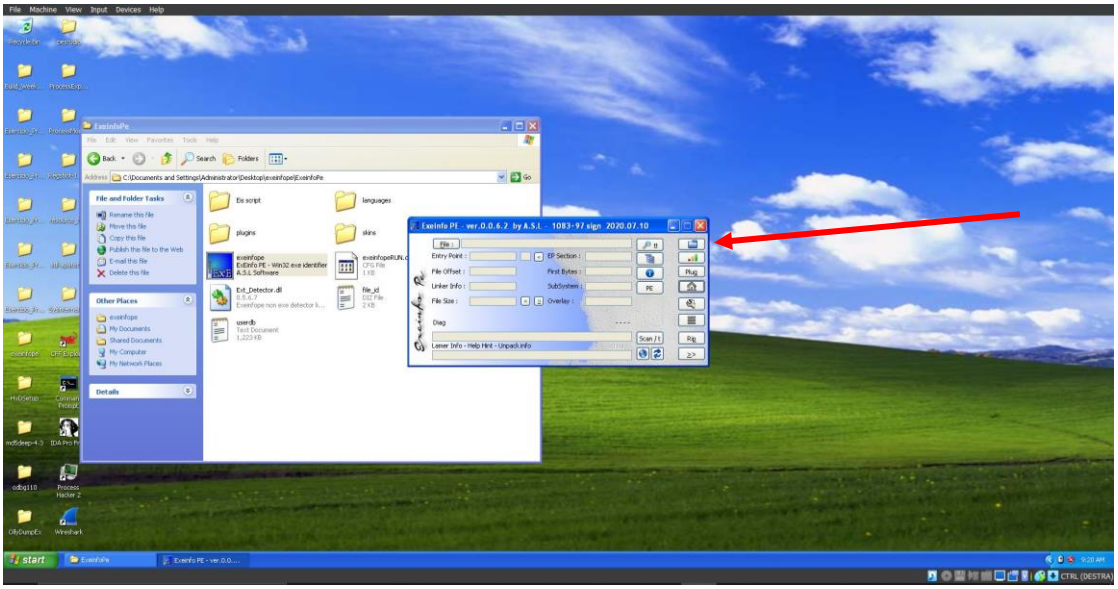
3. EXEINFO PE: ALTERNATIVA DI ESECUZIONE

A scopo didattico, visualizziamo le sezioni del malware anche su Exeinfo PE, altro programma per l'analisi statica dei malware.

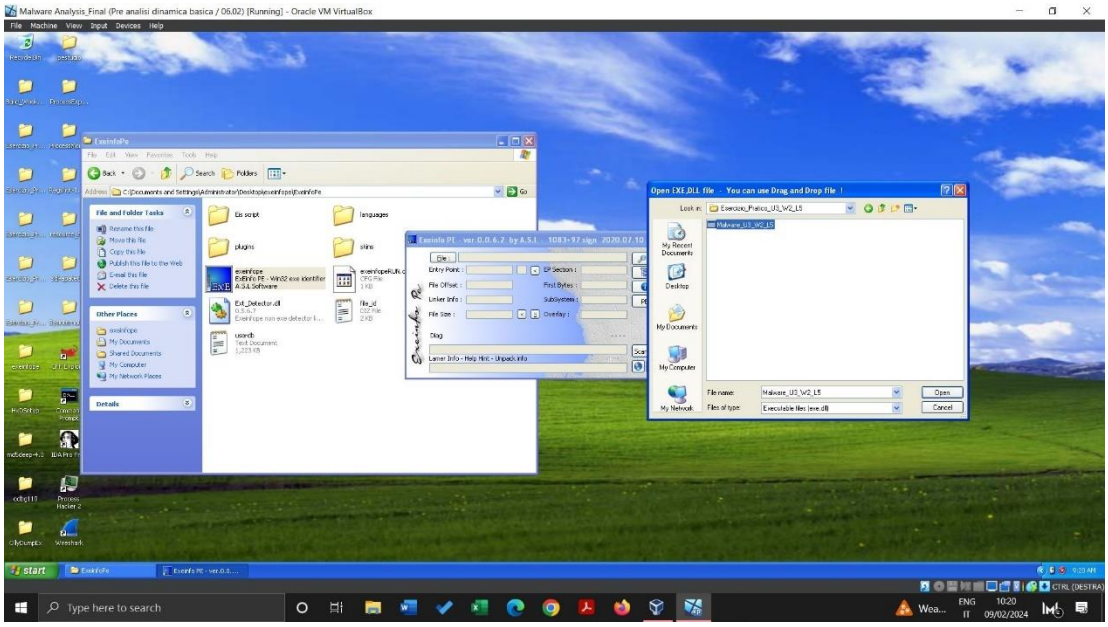
Individuiamo il programma sul desktop della macchina (è la cartella di fianco a CFF Explorer) e lanciamolo:



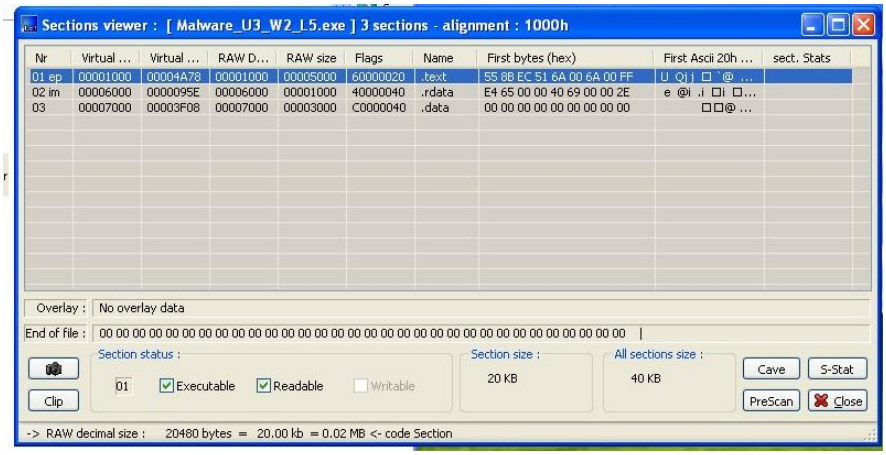
Non ci resta che aprire il file che andremo ad anlizzare. Lo facciamo cliccando il pulsante evidenziato dalla freccia, quello che raffigura una cartella:



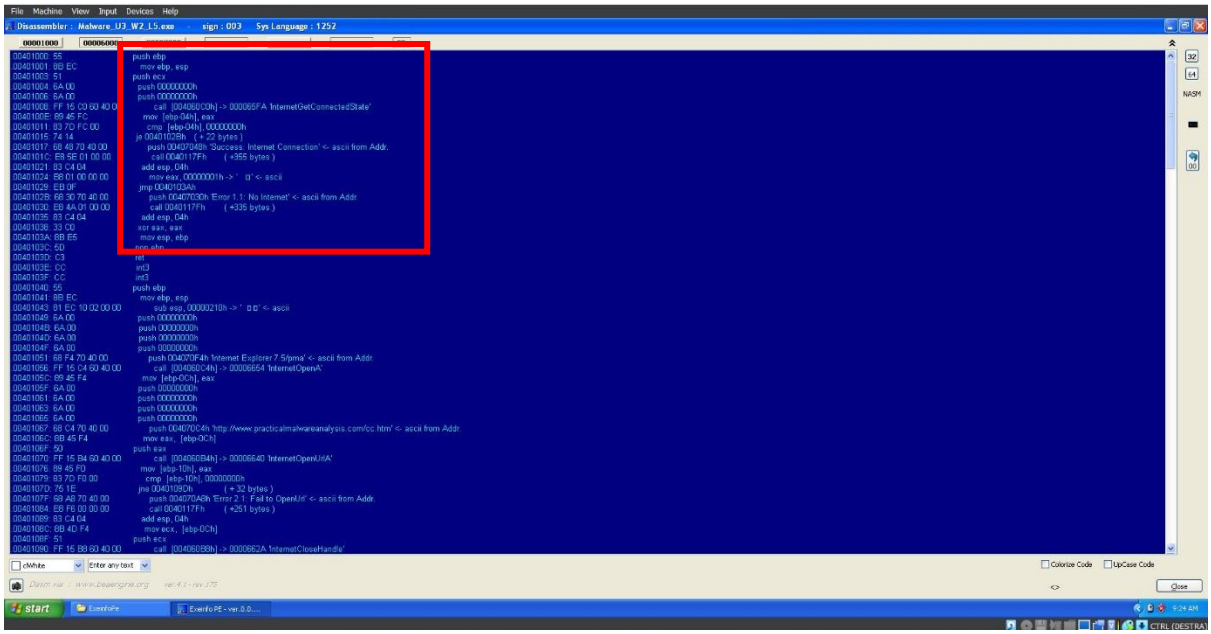
Cerchiamo e selezioniamo il file, poi clicchiamo su open:



Con una GUI diversa, anche qui possiamo individuare le tre sezioni del malware; il software ci indica anche le dimensioni della sezione e del malware intero:



Cliccando con il tasto sinistro del mouse, è inoltre possibile esplorare la sezione in diversi formati. Se si sceglie il linguaggio ASM, ovvero Assembly, cercando nel codice si può individuare questa sezione. Notiamo che i comandi sono molto simili al codice che andremo ad analizzare nella seconda consegna del progetto:



ANALISI STATICA: ASSEMBLY

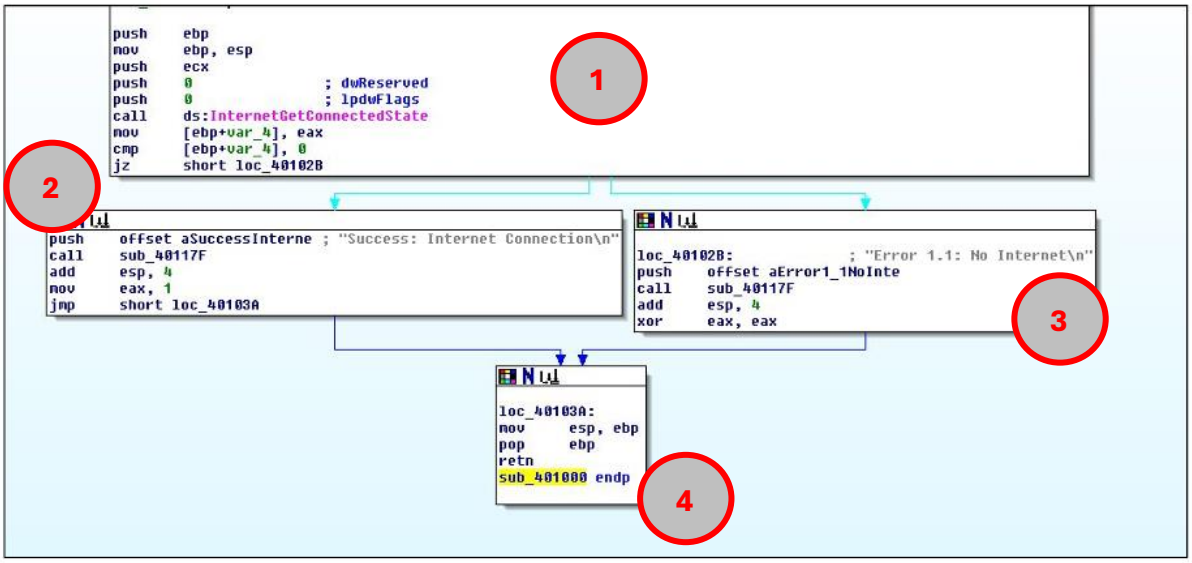
3.1 ASSEMBLY: DEFINIZIONE

Assembly è un linguaggio di basso livello, legato strettamente all'architettura del processore, composto da istruzioni mnemoniche comprensibili dalla CPU. Contrariamente ai linguaggi di alto livello, è più vicino al linguaggio macchina e offre un controllo diretto sull'hardware.

Nel contesto della malware analysis, l'assembly è cruciale. **Il codice sorgente di malware spesso non è disponibile**, e il reverse engineering coinvolge l'analisi del codice binario. **Gli analisti utilizzano l'assembly per disassemblare e comprendere il comportamento del malware**. Questo coinvolge l'identificazione delle funzioni, la comprensione dei flussi di controllo e la rivelazione delle tecniche evasive utilizzate dal malware per sfuggire alla rilevazione.

L'assembly è inoltre fondamentale per capire le vulnerabilità sfruttate dai malware e sviluppare contromisure di sicurezza. Conoscere approfonditamente assembly consente agli analisti di malware di individuare e comprendere le tattiche di attacco, aiutando nella protezione e nella mitigazione dei rischi associati alle minacce informatiche.

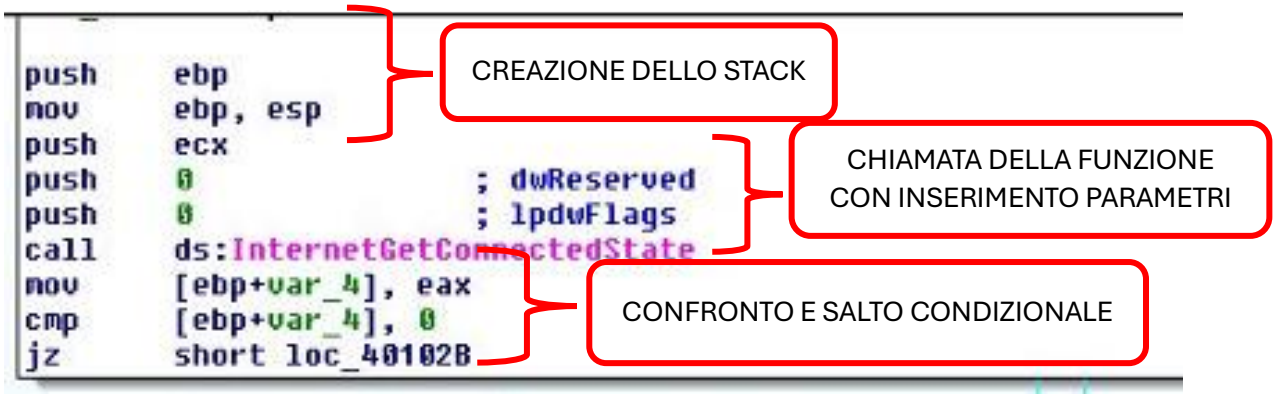
3.2 PANORAMICA DEL CODICE ASSEGNATO



Nella figura, il codice è schematizzato con la struttura di un diagramma di flusso:

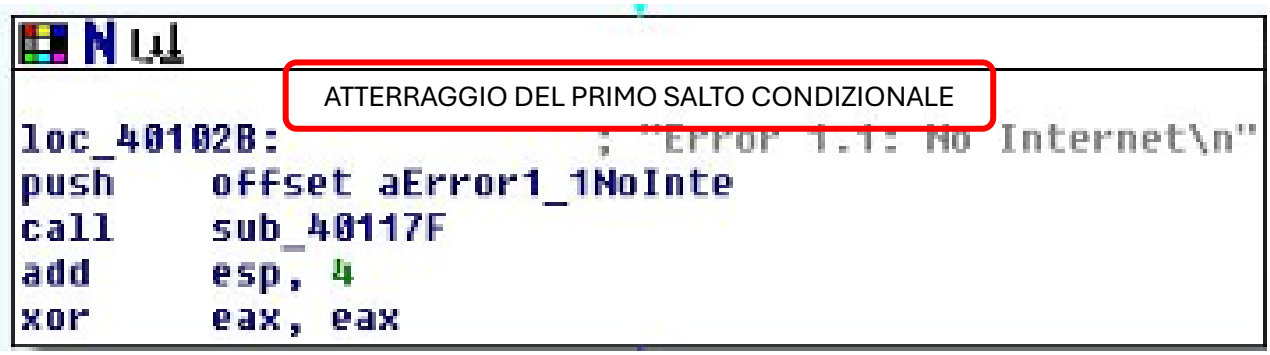
1) La prima parte rappresenta l'avvio del programma: la creazione di uno stack e la chiamata di funzione. Al suo termine si apre una comparazione (a partire da "cmp") che è seguita da un salto condizionale. Lo associamo ad una struttura if-else con due possibili alternative a seconda della condizione;
2) Se il programma non soddisfa la condizione del salto, prosegue nella sezione 2;
3) Se invece la condizione del salto viene soddisfatta, il programma salta appunto a loc_40102B, dove le istruzioni sono rappresentate in sezione 3;
4) In entrambi i casi, il programma prosegue con la sezione 4.

1
IMPOSTAZIONE STACK + PREPARAZIONE PER LA CHIAMATA + CONTROLLO CONNESSIONE

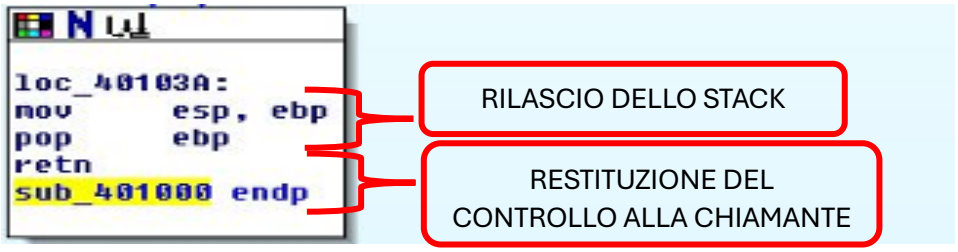


2
GESTIONE DEL CASO "SUCCESSO"

3
GESTIONE DEL CASO "ERRORE"



4
FINE DELLA FUNZIONE



3.3 IL CODICE NEL DETTAGLIO

Analizziamo ora il codice descrivendo passo a passo cosa fa ciascuna riga:

push ebp	Salva il valore del registro di base (EBP) nello stack.
mov ebp, esp	Crea un nuovo frame della pila, copiando il valore di "esp" (puntatore alla cima dello stack) nel registro di base "ebp".
push ecx	Salva il valore del registro ECX nello stack, al fine di conservare il valore originale del registro prima di eventuali modifiche.
push 0 ; dwReserved	Inserisce il valore 0 nello stack. Questo verrà utilizzato come parametro dwReserved per la chiamata successiva.
push 0 ; lpdwFlags	Inserisce nuovamente il valore 0 nello stack. Questo sarà il parametro lpdwFlags per la chiamata successiva.
call ds:InternetGetConnectedState	Chiama la funzione InternetGetConnectedState, per la quale i parametri sono stati posti precedentemente nello stack.
mov [ebp+var_4], eax	Memorizza il risultato della chiamata InternetGetConnectedState nella variabile locale [ebp+var_4].
cmp [ebp+var_4], 0	Confronta il valore memorizzato in [ebp+var_4] con 0.
jz short loc_40102B	Se il risultato della comparazione è zero, salta a loc_40102B.
push offset aSuccessInterne ; "Success: Internet Connection\n"	Inserisce l'indirizzo della stringa "Success: Internet Connection\n" nello stack come argomento per la successiva chiamata.
call sub_40117F	Chiama la funzione sub_40117F.
add esp, 4	Libera lo spazio nello stack precedentemente occupato dai parametri della chiamata.
mov eax, 1	Imposta eax a 1.
jmp short loc_40103A	Salta a loc_40103A.
loc_40102B : ; "Error 1.1: No Internet\n"	Posizione a cui salta il programma in caso di assenza di connessione ad internet.
push offset aError1_1_NoInte	Inserisce l'indirizzo della stringa "Error 1.1: No Internet\n" nello stack come argomento per la successiva chiamata.
call sub_40117F	Chiama la funzione sub_40117F.
add esp, 4	Libera lo spazio nello stack precedentemente occupato dai parametri della chiamata.
xor eax, eax	Esegue un XOR tra eax e se stesso, azzera il registro.
loc_40103A:	Posizione a cui il programma salta in ogni caso.
mov esp, ebp	Ripristina il valore originale di esp dalla variabile ebp.
pop ebp	Ripristina il valore originale del registro di base ebp dalla pila.
retn	Restituisce il controllo alla funzione chiamante.
sub_401000 endp	Questa riga indica la fine della procedura.

3.4 IPOTESI SULLA FUNZIONE DEL CODICE

In seguito alle analisi effettuate nelle sezioni precedenti, possiamo dedurre che il codice verifica se c'è una connessione a Internet utilizzando la funzione InternetGetConnectedState. Se la connessione è attiva, mostra un messaggio di successo e restituisce 1; altrimenti, mostra un messaggio di errore e restituisce 0. L'organizzazione del codice utilizza sezioni specifiche e istruzioni di stack. La funzione principale, sub_401000, controlla il flusso decisionale in base allo stato della connessione.

CONSIDERAZIONE FINALI

Nella prima parte dell'esercizio, abbiamo utilizzato CFF Explorer ed EXEINFO PE per analizzare staticamente un malware. Così abbiamo potuto individuare due librerie caricate dal malware e tre sezioni che ne caratterizzano il flusso.

Scorrendo rapidamente il codice in formato Assembly abbiamo notato una struttura che poi abbiamo ritrovato nella seconda parte del progetto odierno.

Dopo avere analizzato i costrutti in Assembly, individuando in particolare alcune chiamate di funzione e una struttura if-else, abbiamo potuto concludere che si tratta di un programma che verifica lo stato della connessione Internet e gestisce messaggi di output di conseguenza. Al centro del codice, la funzione InternetGetConnectedState.