S10 LEZIONE 5 (PROGETTO)

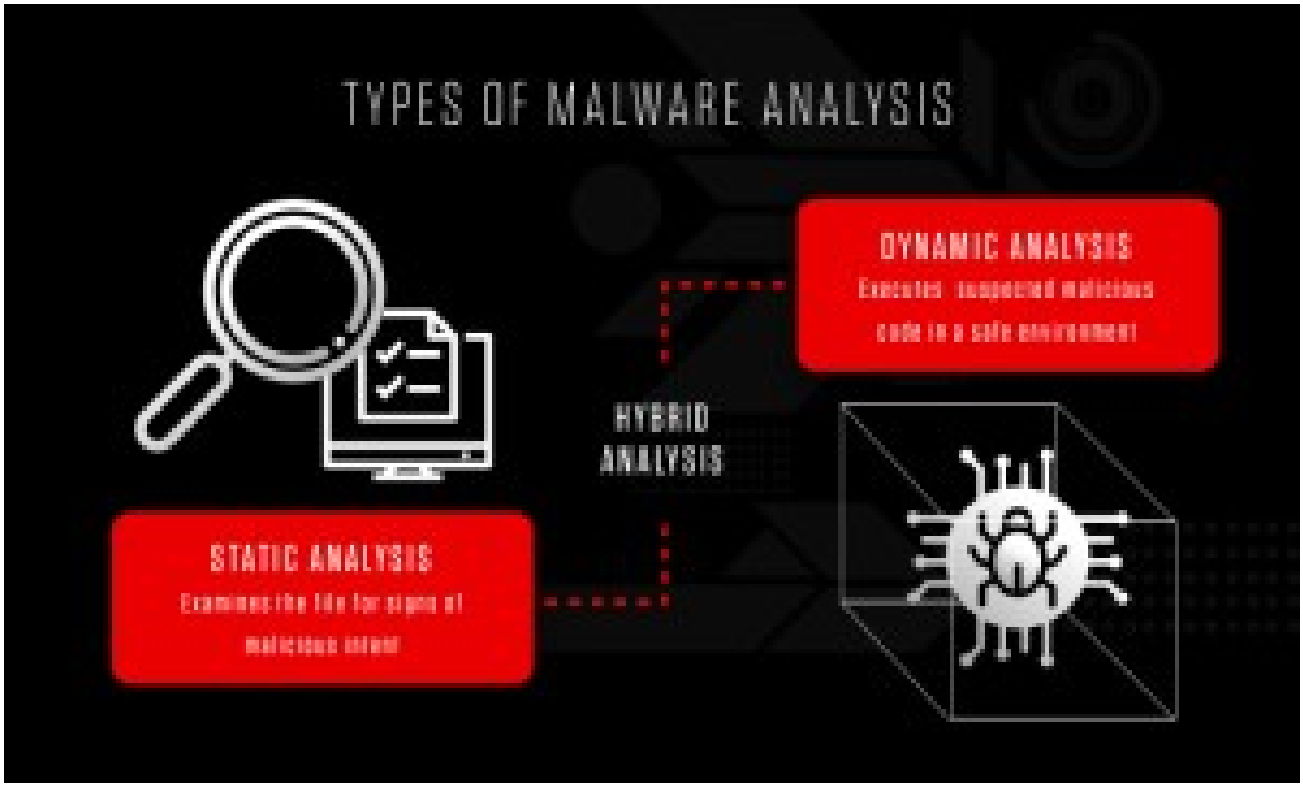
Traccia:

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:

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

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

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



ANALISI MALWARE

L’analisi **malware** è un processo cruciale per la **sicurezza informatica**, volto a comprendere il **comportamento** e le **intenzioni** di un sofwtare sospetto.

Esistono due tecniche principali di analisi malware:

- *analisi statica:* fornisce **tecniche e strumenti** per analizzare il comportamento di un software malevolo **senza la necessità di eseguirlo.**

- *analisi dinamica:* presuppone l’**esecuzione del malware** all’interno di un **ambiente controllato.**

Ambedue si possono dividere in analisi basica e avanzata:

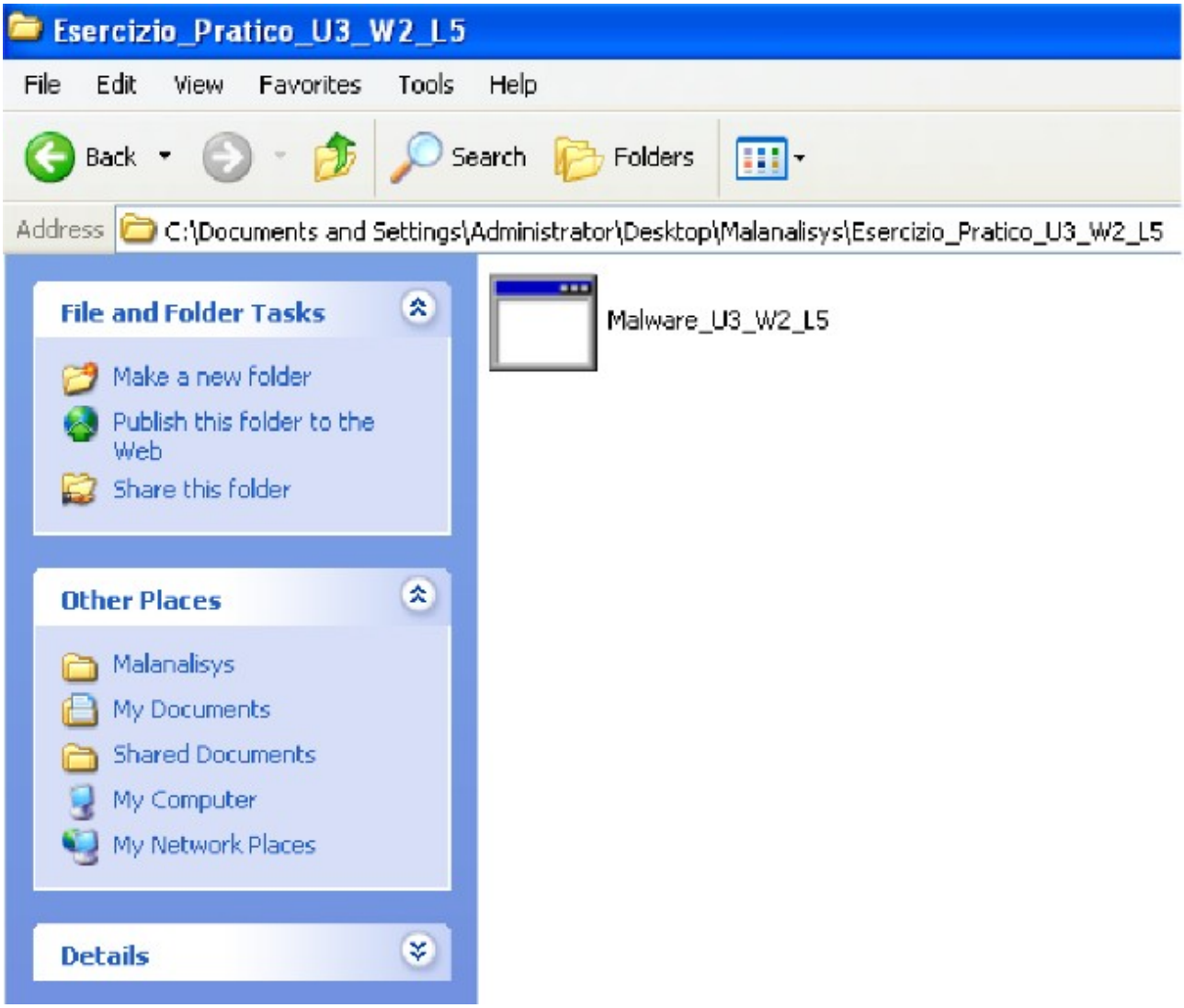
- *analisi statica basica:* consiste nell’esaminare un eseguibile senza vedere le istruzioni di cui è composto. Il suo scopo è quello di **confermare** se un dato **file è malevolo** e fornire informazioni generiche circa le sue **funzionalità**. L’analisi statica basica è sicuramente la più intuitiva e semplice da mettere in atto, ma risulta essere anche la più **inefficiente** soprattutto con **malware complessi.**

- *analisi dinamica basica:* presuppone l’**esecuzione** di un **malware** in modo tale da osservare il suo comportamento. I malware devono essere eseguiti **in un ambiente sicuro e controllato** in modo tale da eliminare ogni rischio di arrecare danno a sistemi o reti. Così come l’analisi statica basica, l’analisi dinamica basica è piuttosto semplice da mettere in atto, ma **non è molto efficace** quando si parla di analizzare **malware più complessi.**

- *analisi statica avanzata:* presuppone la c**onoscenza di fondamenti di reverse engineering** al fine di **identificare il comportamento di un malware** a partire dall’analisi delle **istruzioni che lo compongono.** In questa fase vengono utilizzati **tool** chiamati **disassebler,** i quali ricevono in input un file eseguibile e **restituiscono in output codice in linguaggio assembly**

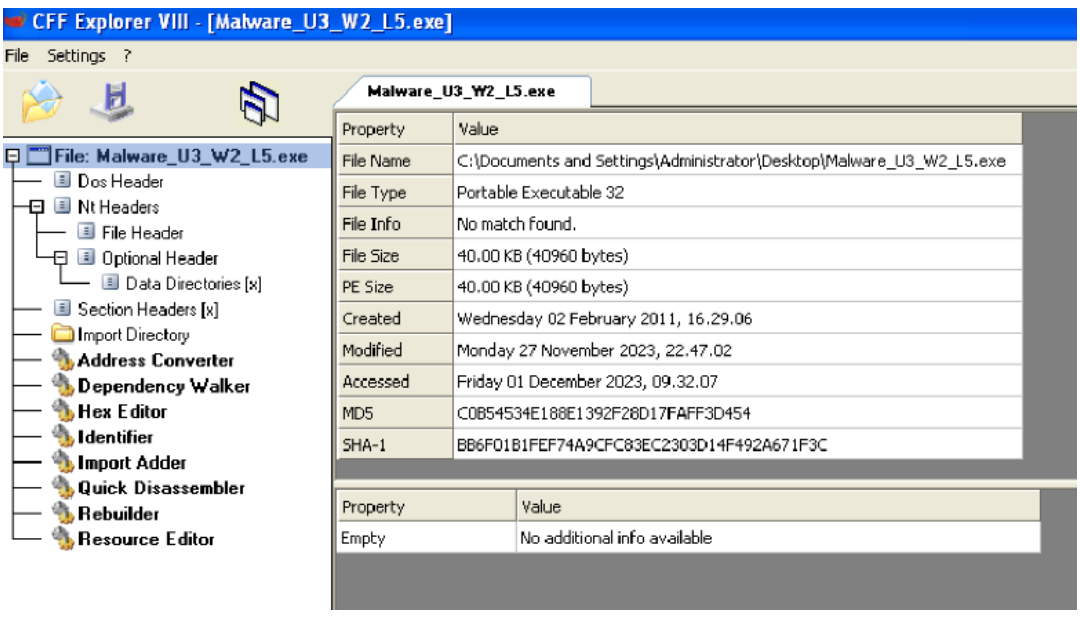
- *analisi dinamica avanzata:* presuppone la conoscenza dei **debugger** per esaminare lo stato di un programma durante l’esecuzione **(esecuzione passo passo del programma)**

ESECUZIONE TEST ANALISI STATICA BASICA Malware_U3_W2_L5

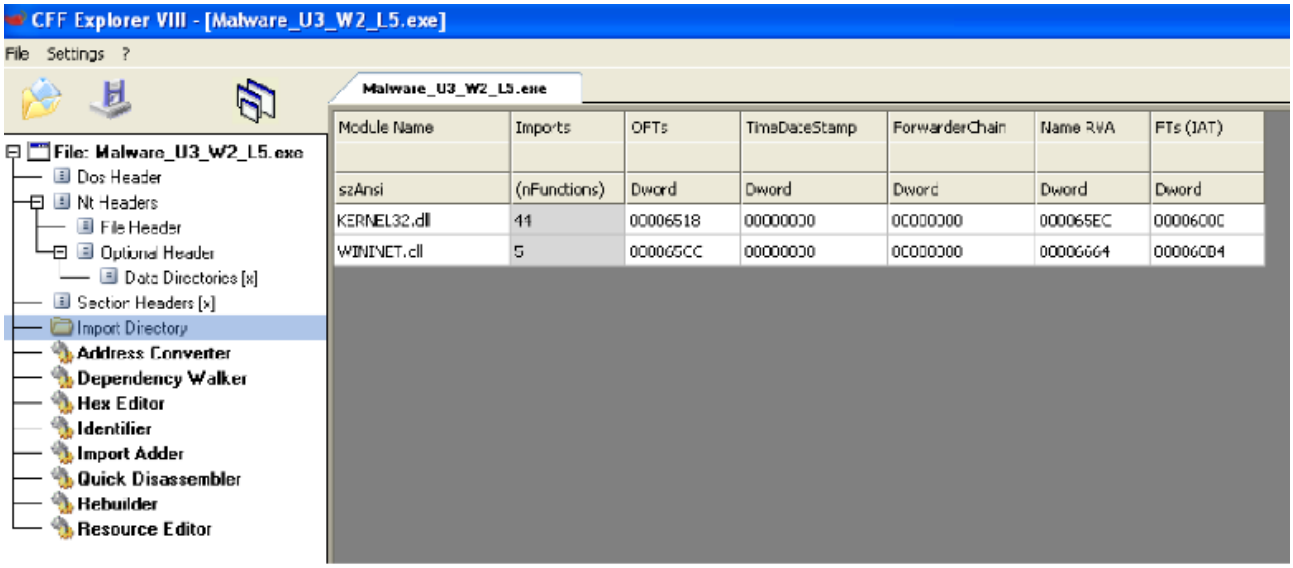


Per eseguire l'analisi utilizzo un **ambiente di test realizzato con Virtualbox** il quale monta il sistema operativo **Windows Xp.**

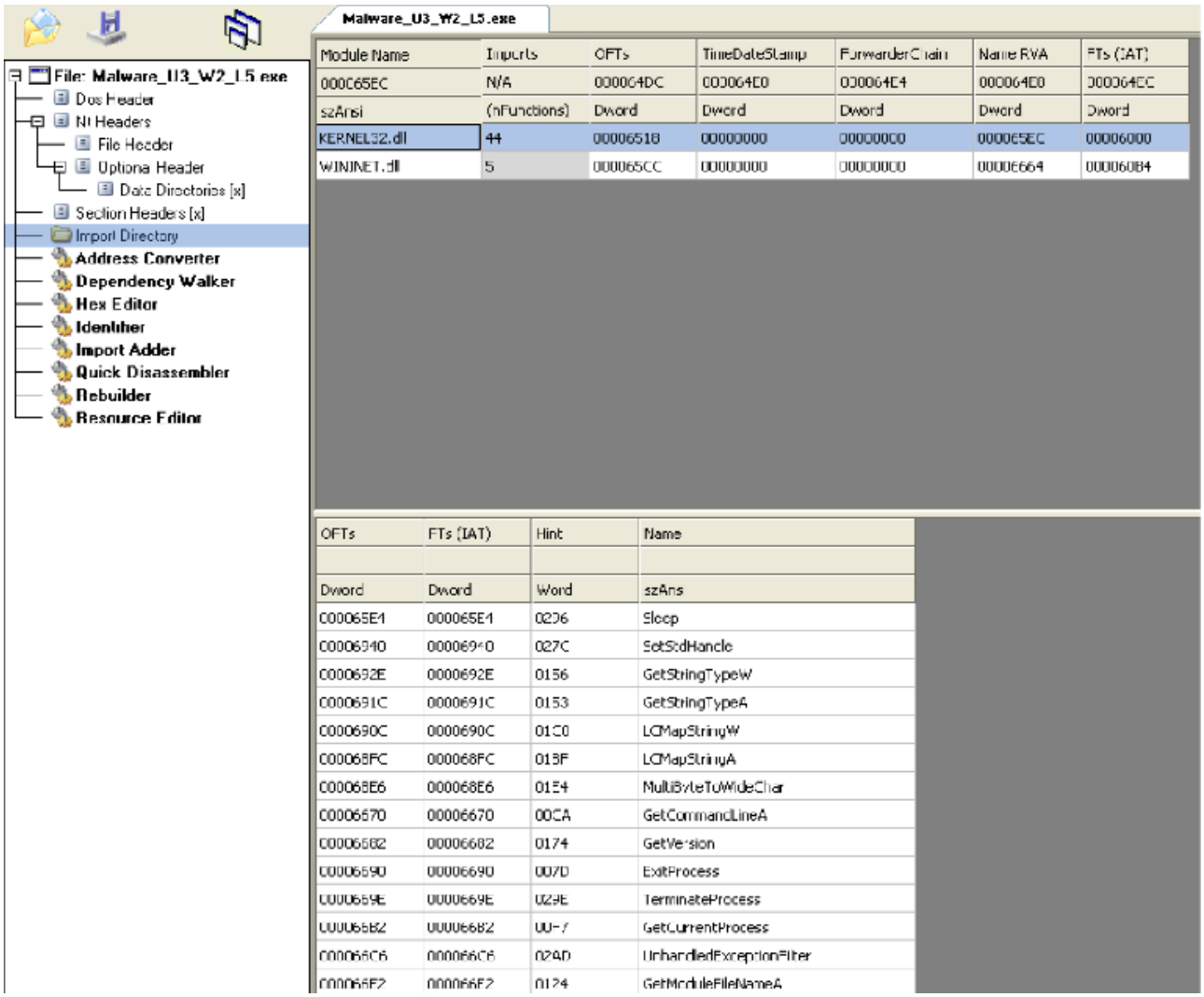
Per analizzare il malware utilizzo il **tool CFF Explorer**, una suite di strumenti per l'**analisi** e la **modifica** di file **eseguibili portatili:**



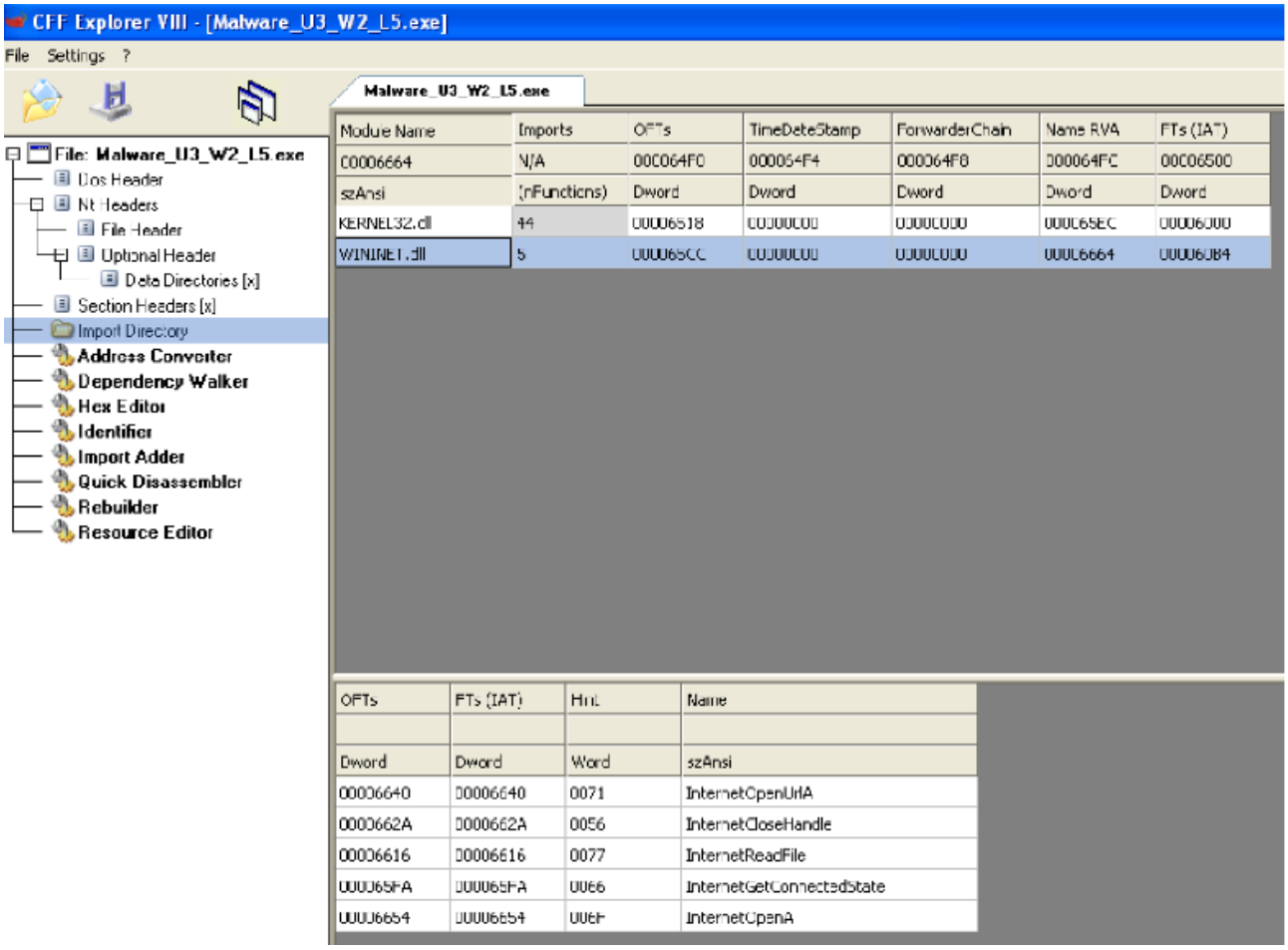
Dopo aver avviato **CFF Explorer** e aver caricato l'**eseguibile del malware**, vado nella sezione **Import Directory** per controllare le **librerie e le funzioni importate:**



KERNEL32.dll: è una libreria di Windows che fornisce funzioni di base per il sistema operativo come **accesso alla memoria**, **esecuzione di processi** e **gestione dei file.**



WININET.dll : è una **libreria** di Windows che fornisce **funzioni** per l'**accesso a internet** come la connessione a siti web, **download di file** e **invio di richieste HTTP.**



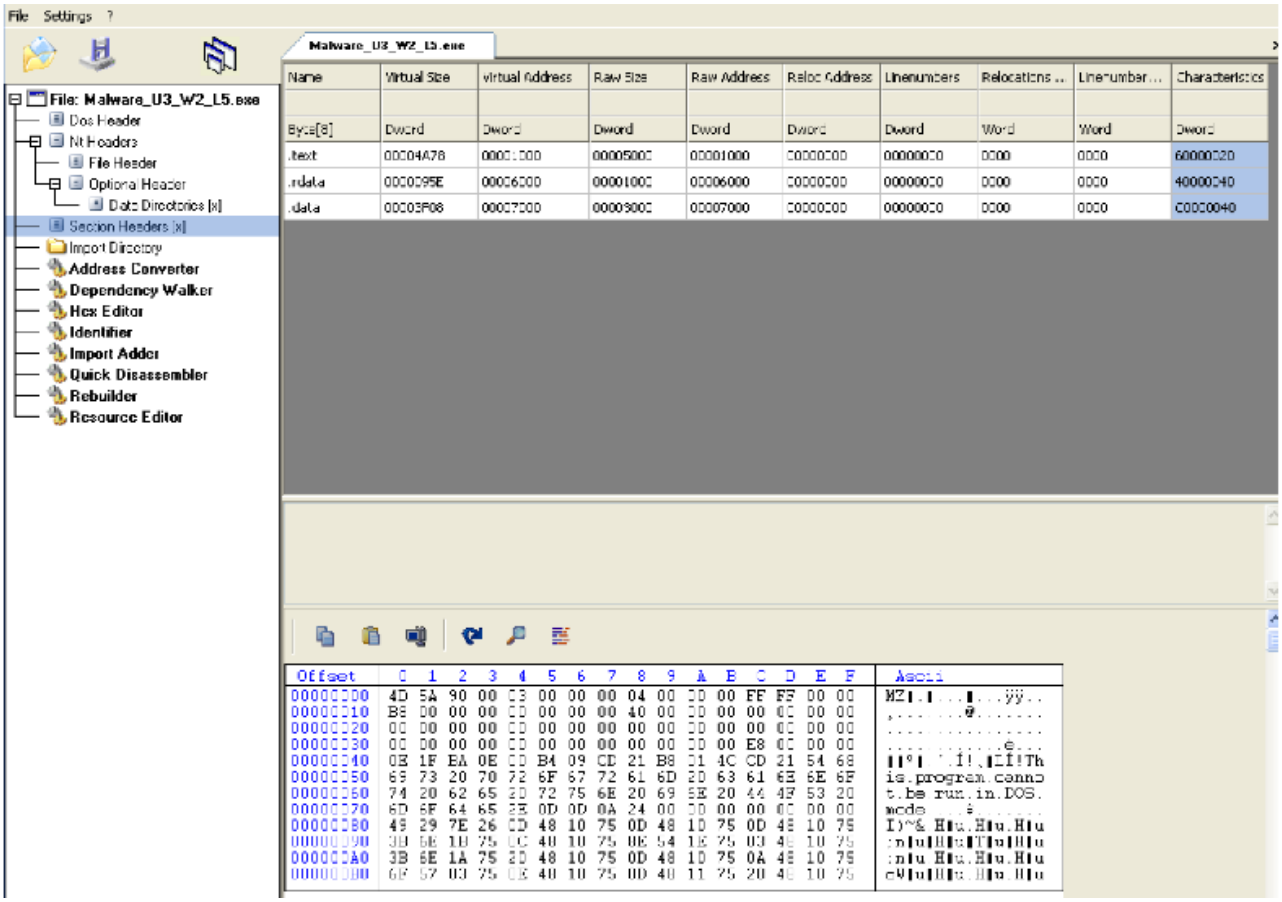
In base a queste informazioni è possibile **ipotizzare** che il malware utilizzi queste librerie per i seguenti scopi:

- eseguire azioni sul sistema operativo: ad esempio con la libreria KERNEL32.dll per accedere alla memoria dell'utente eseguire processi e modificare file.

- accedere a internet: con le funzioni di WININET.dll potrebbe connettersi a internet, scaricare file dannosi e inviare richieste HTTP.

Quindi il **malware** potrebbe potenzialmente risultare **non tracciabile** da software antivirus ed **eseguire codice dannoso in background**, **scaricare file dannosi** dal web e **inviare informazioni sensibili** all'attaccante.

Spostandosi nella sezione **Section Headers** è possibile analizzare **le sezioni di cui si compone** il malware.



Le sezioni sono **parti** che compongono un **software** e svolgono **funzioni specifiche:**

- .text : contiene le **istruzioni** che la **CPU eseguirà** una volta che il software sarà avviato. Questa è l'unica sezione di un .exe che viene eseguita dalla CPU in quanto le altre sezioni contengono dati o informazioni a supporto

- .rdata : include generalmente le **informazioni** circa le **librerie** e le **funzioni importate** ed **esportate dall'eseguibile**

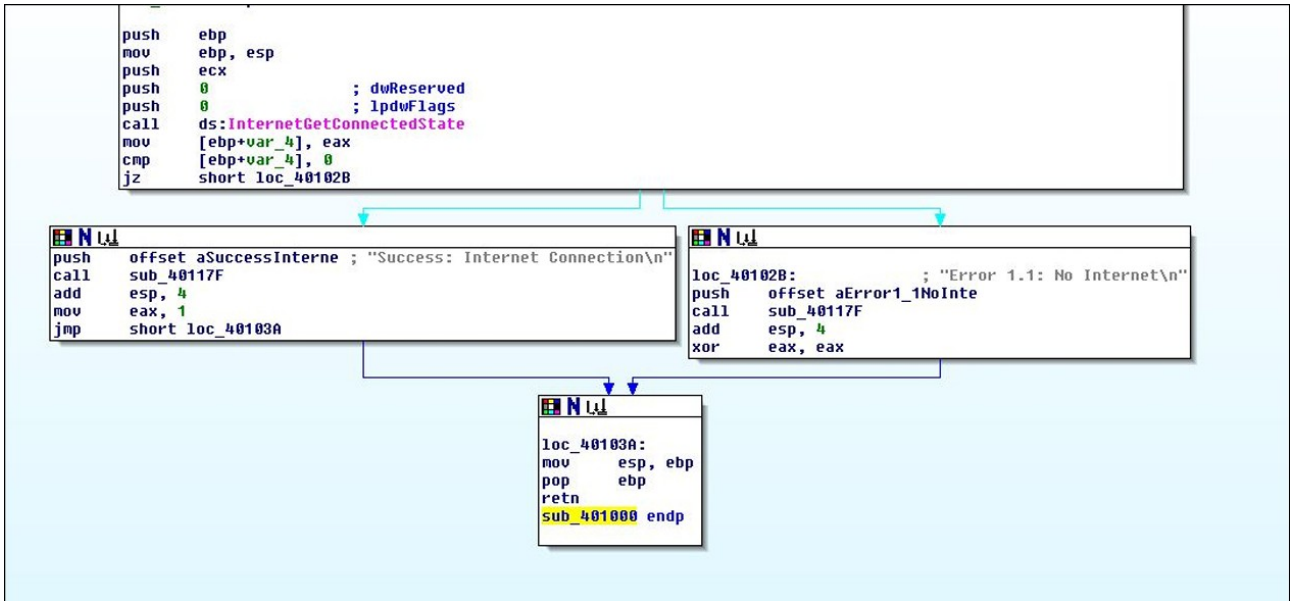
- .data : contiene tipicamente **dati e variabili globali** del programma eseguibile.

Si può dedurre che questo malware sia un **trojan** progettato per **rubare info dal computer** attaccato infatti il malware contiene codice per eseguire operazioni di base come l'**apertura di file e la scrittura su disco** e per **comunicare** con un **server remoto.**

Tipicamente i **trojan** sono utilizzati per:

- rubare informazioni personali
- rubare informazioni finanziarie
- rubare password e credenziali
- installare malware
- prendere il completo controllo del pc

ANALISI STATICA AVANZATA



Passiamo ora all'analisi del codice nel linguaggio **assembly.**
Assembly è un **linguaggio di programmazione a basso livello** che viene utilizzato per **comunicare** direttamente con il **computer** (linguaggio macchina).

Esso viene **utilizzato dai malware analyst** per una serie di scopi:

- *analisi del codice:* può essere utilizzato per **analizzare** il codice a l**ivello più specifico** rispetto ai linguaggi di programmazione ad alto livello
- *rilevamento malware:* utilizzato per **rilevare malware** offuscato e crittografato
- *rimozione malware:* utlizzato per le **rimozione** del malware

Dal codice si possono identificare 2 costrutti:

- *creazione dello stack :* le prime due righe di codice,
  push ebp , move ebp, esp, servono a creare lo **stack**

- *ciclo:* la funzione contiene il **ciclo condizionale** *cmp [ebp+var_4], 0*. Questo **ciclo** verifica se il valore della **variabile** è **uguale a 0**, se lo è il ciclo salta alla *loc401028.*

Questo **codice** implementa il **controllo** della **connessione ad internet.** La variabile *var_4* contiene lo stato della connessione, se quest'ultimo è uguale a zero, la connessione risulta non essere attiva e in questo caso stampa un messaggio di errore.
Se lo stato è diverso da zero, la connessione risulta essere attiva e viene stampato un messaggio di succeso e restituisce il valore 0.

Esaminiamo il codice riga per riga:

.push ebp : Salva il valore di EBP sullo stack

mov ebp, esp : Imposta EBP come puntatore allo stack

push ecx : Salva il valore di ECX sullo stack

push 0 : Pusha 0 sullo stack

push lpdwFlags : Pusha il valore di lpdwFlags sullo stack

call ds:InternetGetConnectedState : Chiama la funzione InternetGetConnectedState

mov [ebp+var_4], eax : Salva il valore restituito da InternetGetConnectedState nella variabile locale var_4

cmp [ebp+var 4], 0 : Confronta il valore di var_4 con 0

jz loc 401028 : Se uguale a 0, salta a loc_401028

push offset aSuccessInternet : Pusha l'offset di aSuccessInternet sullo stack

call sub 40117F : Chiama la funzione sub_40117F

jmp short loc 40103A : Salta a loc_40103A

add esp, 4 : Incrementa il puntatore allo stack di 4

push offset aError1_1NoInternet : Pusha l'offset di aError1_1NoInternet sullo stack

mov eax, 1 : Assegna 1 a EAX

call sub 40117F : Chiama la funzione sub_40117F

jmp short loc : Salta a loc_40103A

add esp, 4 : Incrementa il puntatore allo stack di 4

xor eax, eax : Assegna 0 a EAX

pop ebp : Ripristina il valore di EBP dallo stack

retn : Restituisce il controllo al chiamante