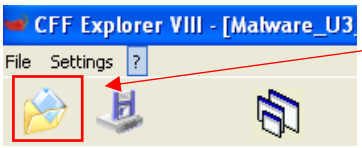
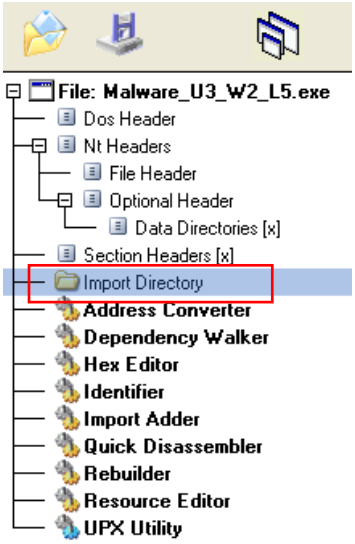
ANALISI MALWARE E CODICE ASSEMBLY

1. Quali librerie vengono importate dal file eseguibile?

Per capire quali librerie vengono importate, possiamo effettuare un'analisi statica basica dell'eseguibile attraverso il tool CFF Explorer, che permette di controllare le funzioni importate ed esportate dal malware; bisogna aprire il programma e selezionare l'eseguibile da analizzare tramite l'icona cartella:



Si aprirà un menù a tendina dove, per controllare le librerie, andrà selezionata la voce **Import Directory:**



Abbiamo quindi scoperto le librerie che sono:

KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

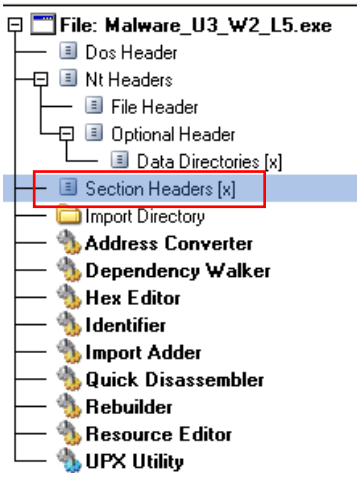
KERNEL32.dll che è un modulo del kernel (parte centrale di un sistema operativo che esegue le operazioni di base e fondamentali tra cui la gestione della memoria, le operazioni di input/output e gli interrupt) di Windows. È una libreria a collegamento dinamico a 32 bit utilizzata nei sistemi operativi Windows. All'avvio del sistema, kernel32.dll viene caricato in una memoria protetta in modo che non venga danneggiato da altri processi di sistema o utente. Funziona come un processo in background e svolge funzioni importanti come la gestione della memoria, operazioni di input/output e interruzioni;

WININET.dll che fornisce l'interfaccia tra le applicazioni che utilizzano WinInet e Windows Sockets. Le applicazioni che utilizzano questa API controllano se esiste una connessione Internet e, una volta verificata, l'applicazione può

aprire un handle alla risorsa remota, richiedere una connessione per un protocollo specifico e aprire sessioni su quell'handle per comunicazioni HTTP, FTP o Gopher. WinINet fornisce funzionalità come caching, cronologia, gestione dei cookie, autenticazione base, NTLM, Kerberos, connessioni sia sicure (schannel) che non sicure, Dial-up, Diretto, Proxy, gestione del protocollo e dell'intestazione HTTP.

2. Quali sono le sezioni di cui si compone il file eseguibile?

Per scoprire quali sono le sezioni, usiamo sempre il tool CFF Explorer; dal menù a tendina selezioniamo la voce **Section Headers:**



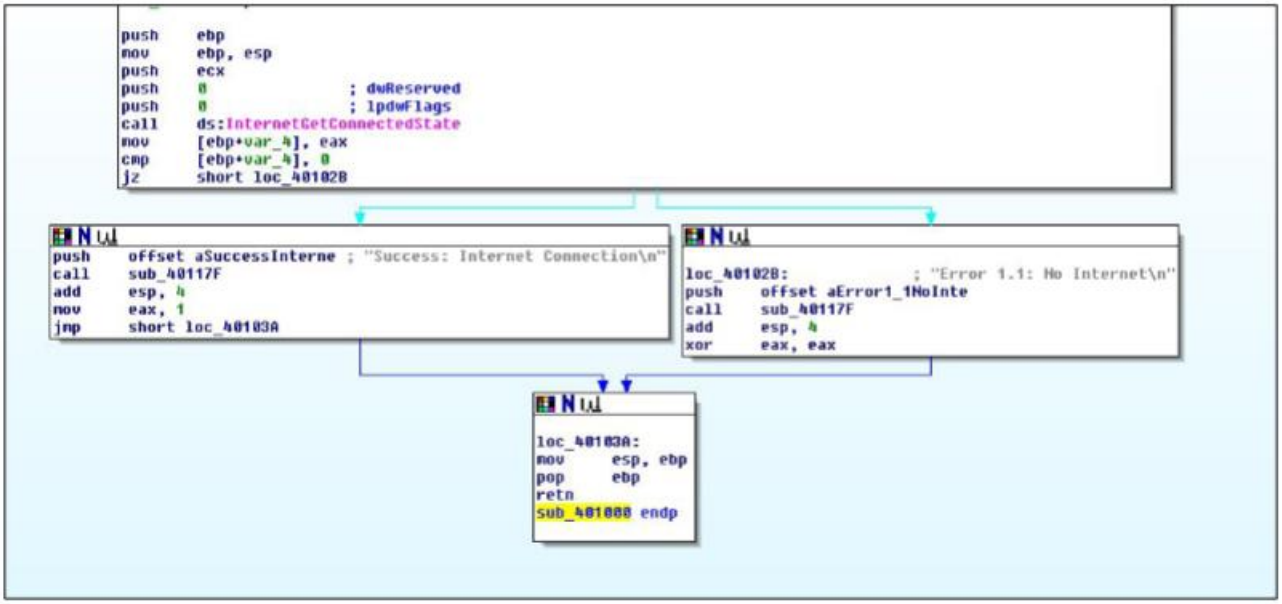
Le sezioni di cui si compone l'eseguibile sono:

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

.text che contiene le righe di codice che la CPU esegue quando il programma verrà avviato; questa è l'unica sezione che viene eseguita dalla CPU;

.rdata che include le informazioni riguardo le librerie e le funzioni importate ed esportate dal software;

.data che contiene le variabili globali (variabili accessibili da qualsiasi funzione) dell'eseguibile.



3. Con riferimento alla figura sopra, identificare i costrutti

```
push    ebp
mov     ebp, esp
```
: creazione dello stack;

```
push    ecx
push    0               ; dwReserved
push    0               ; lpdwFlags
call    ds:InternetGetConnectedState
```
: passaggio dei parametri alla funzione (i 3 push) con conseguente chiamata;

```
cmp     [ebp+var_4], 0
jz      short loc_40102B
```
: costrutto IF;

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```
: chiamata di funzione, pusha la stringa all'inizio dello stack e chiama probabilmente un printf (sub_40117F);

```
loc_40102B:             ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
add     esp, 4
xor     eax, eax
```
: chiamata di funzione simile alla precedente, vediamo infatti che la locazione di memoria è sempre la stessa (sub_40117F), quindi un'altra probabile stampa della stringa pushata;

```
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```
: pulizia/eliminazione dello stack.

4. Ipotizzare il comportamento della funzionalità implementata

Con la funzione InternetGetConnectedState, l'eseguibile andrà a controllare se la macchina ha una connessione ad Internet oppure no e, una volta fatto il controllo, va a stampare a schermo se c'è connessione o meno.

5. Spiegare ogni singola riga del codice in assembly

push ebp spinge il registro ebp in cima allo stack:

mov ebp, esp sposta il contenuto del registro esp all'interno del registro ebp;

push ecx spinge il registro ecx in cima allo stack;

push 0 ;dwReserved spinge in cima allo stack una variabile inizializzata a 0;

push 0 ;lpdwFlags spinge in cima allo stack una variabile inizializzata a 0;

call ds:InternetGetConnectedState chiamata di funzione tramite il puntatore ds;

mov [ebp+var_4], eax sposta il valore del registro eax all'interno dell'indirizzo di memoria specificato;

cmp [ebp+var_4], 0 compara la destinazione con 0 per poter effettuare i successivi jump;

jz short loc_40102B salta alla locazione di memoria 40102B se lo zero flag è 1;

push offset aSuccessInterne spinge il contenuto della stringa in cima allo stack;

call sub_40117F chiama la funzione allocata in 40117F;

add esp, 4 aggiunge 4 al valore del registro esp;

mov eax, 1 sposta 1 all'interno del registro eax;

jmp short loc_40103A salta alla locazione di memoria 40103A;

push offset aError1_1NoInte spinge il contenuto della stringa in cima allo stack;

call sub_40117F chiama la funzione allocata in quella parte di memoria;

add esp,4 aggiunge 4 al valore del registro esp;

xor eax, eax altro modo di pulire il registro eax, ha la stessa funzione di mov eax, 0;

mov esp, ebp sposta il contenuto di ebp all'interno del registro esp;

pop ebp toglie l'ebp dalla cima dello stack;

retn istruzione che serve per ritornare alla funzione chiamante;

sub_401000 endp termina il processo principale.