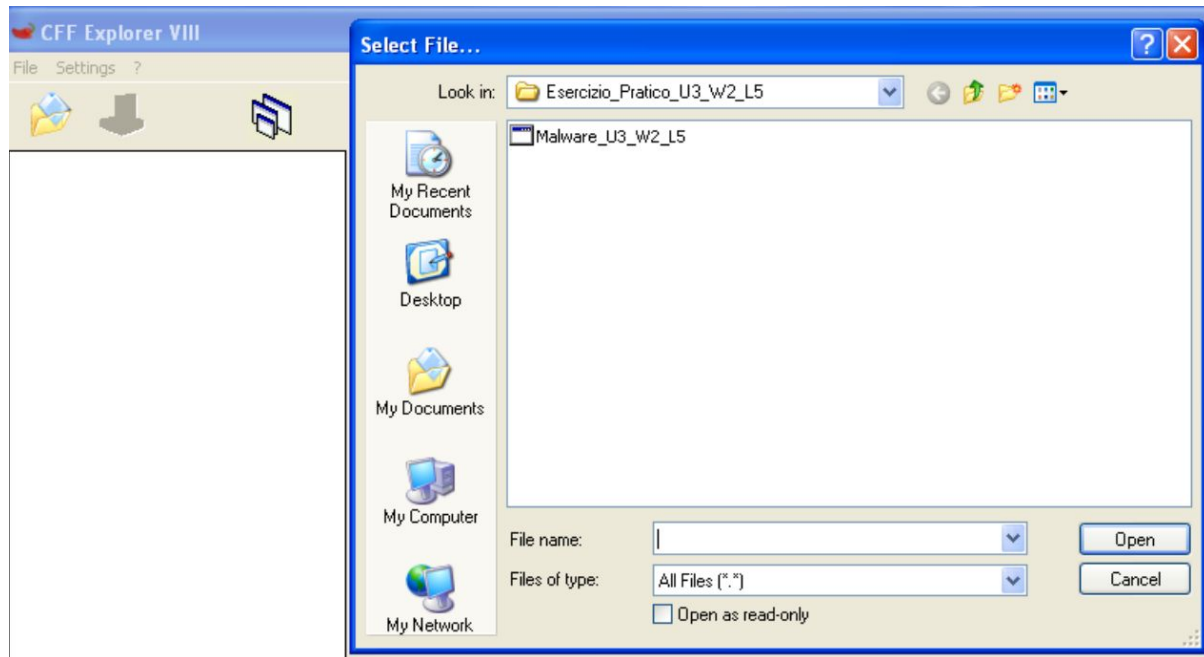


Malware Analysis & Assembly

La **Malware Analysis** consiste nell'analisi approfondita di un determinato malware ed i suoi effetti sulla macchina infetta. Si possono adoperare due approcci: l'analisi statica e l'analisi dinamica.

Il **linguaggio Assembly** viene utilizzato per leggere le istruzioni eseguite dalla CPU in un formato più leggibile per l'uomo; esso cambia da architettura ad architettura di un PC.

1. Malware_U3_W2_L5: Librerie

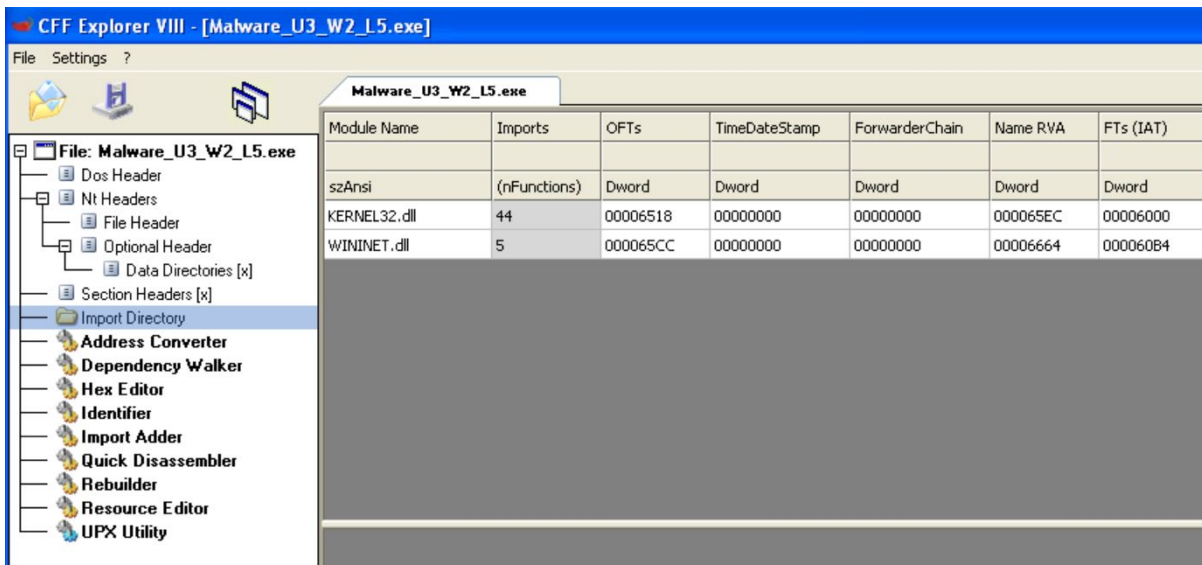


Nell'esercizio di oggi ci è stato richiesto di:

1. individuare le librerie importate dal file .exe
2. le sezioni che compongono il file .exe

Andiamo a fare questa analisi statica basica con il tool **CFF Explorer**.

Andiamo a trascinare la nostra cartella contenente il file .exe malevolo.



Librerie

Ci spostiamo sulla cartella 'Import Directory' per visualizzare le librerie che compongono il file eseguibile.

Le librerie sono un insieme di funzioni, vengono richiamate quando un programma ha bisogno di una funzione contenuta in essa.

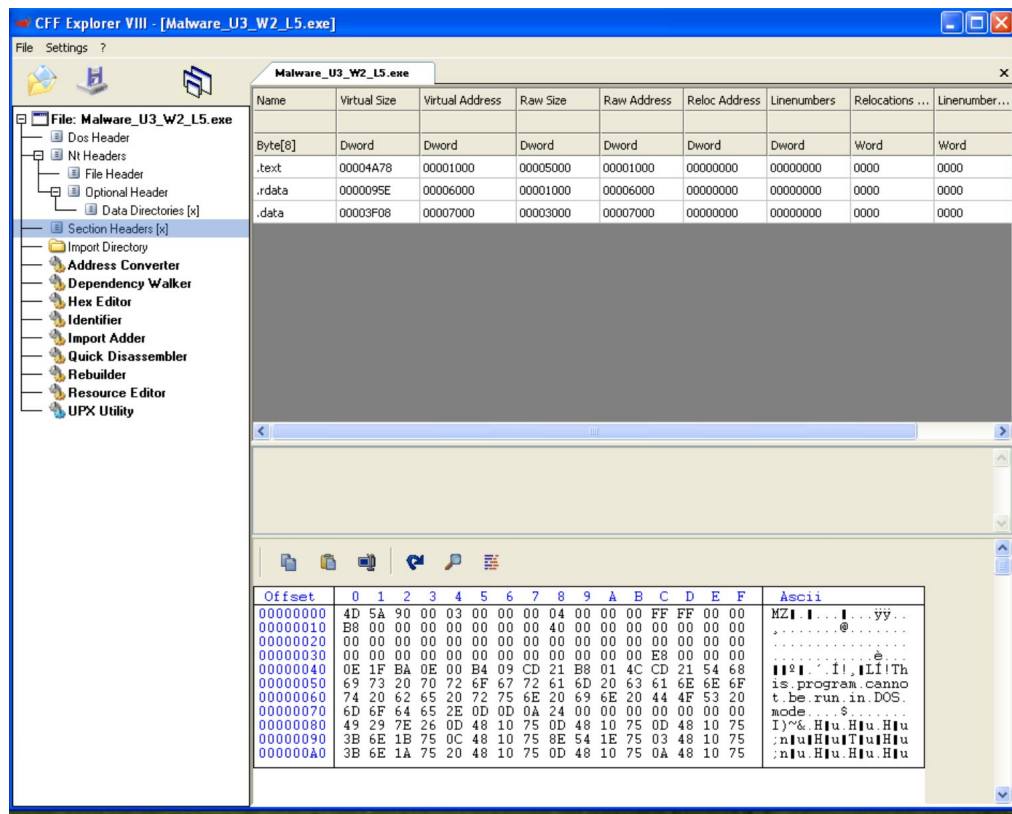
Le librerie trovate sono:

- **KERNEL32.dll** -> contiene le funzioni principali che interagiscono con l'OS.
- **WININET.dll** -> libreria contenente funzioni per l'implementazione di alcuni protocolli di rete (es.http).

Analisi statica basica: consiste nell'esaminare un file .exe senza vedere le istruzioni dalle quali è composto; il suo scopo è quello di riuscire a capire se il file analizzato è malevolo e fornirci le sue funzionalità.

Analisi dinamica basica: osserviamo il malware già in esecuzione per osservare il comportamento sul sistema che viene infettato ai fini di rimuoverlo.

2. Malware_U3_W2_L5: Sezioni



CFF Explorer VIII - [Malware_U3_W2_L5.exe]

File Settings ?

Malware_U3_W2_L5.exe

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000

Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F Ascii

00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ|...|...yy..

00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....

00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00e.....

00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 111.11,111Th

00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program.canno

00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t.be.run.in.DOS.

00000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode.....s

00000080 49 29 7E 26 0D 48 10 75 0D 48 10 75 0D 48 10 75 I)~& Htu.Htu.Htu

00000090 3B 6E 1B 75 0C 48 10 75 8E 54 1E 75 03 48 10 75 ;ntuHtuTtuHtu

000000A0 3B 6E 1A 75 20 48 10 75 0D 48 10 75 0A 48 10 75 ;ntu.Htu.Htu.Htu

Ci spostiamo sulla cartella 'Section Headers' dove troviamo le sezioni che compongono il software.

Quelle trovate sono:

- **.text**: contiene le istruzioni che verranno eseguite dalla CPU una volta eseguito il software
- **.rdata**: contiene informazioni riguardanti le librerie importate ed esportate dall'eseguibile
- **.data**: contiene i dati / variabili globali del programma eseguibile. La *variabile globale* deve essere globalmente dichiarata e accessibile da tutte le funzioni.

Assembly

```
push    ebp
mov     ebp, esp
push    ecx
push    0          ; dwReserved
push    0          ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_401028
```

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

```
loc_401028:
push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
call    sub_40117F
add     esp, 4
xor     eax, eax
```

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

4. Costrutti

Creazione dello stack



```
push    ebp
mov     ebp, esp
```

Chiamata di funzione.
Con push passiamo i
parametri.



```
push    ecx
push    0                ; dwReserved
push    0                ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
```

Ciclo IF



```
cnp     [ebp+var_4], 0
jz      short loc_40102B
```

Possibili output del ciclo IF

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"  
call    sub_40117F  
add     esp, 4  
mov     eax, 1  
jmp     short loc_40103A
```

L'output ci mostra che la connessione è attiva

```
loc_40102B:                ; "Error 1.1: No Internet\n"  
push    offset aError1_1NoInte  
call    sub_40117F  
add     esp, 4  
xor     eax, eax
```

L'output ci mostra che la connessione è assente

Qui riportato in alto abbiamo due possibili output del ciclo IF, iniziato nella slide precedente, se il valore che ci ritorna la funzione è diverso da 0, vuol dire che la connessione è attiva.

```
loc_40103A:  
mov     esp, ebp  
pop     ebp  
retn  
sub_401000 endp
```

Quest'ultima parte chiude il ciclo e ripulisce lo stack

4. Funzionalità del malware

La funzionalità del malware analizzato oggi è di stabilire se vi è una connessione attiva o meno. Il malware invoca la funzione `InternetGetConnectedState` e controlla con il costrutto IF il valore che la stessa ritorna. Se il valore è diverso da 0, questo indica che vi è una connessione attiva.

Possiamo presupporre che il malware in questione è un downloader poiché cerca di vedere se la connessione a internet sia attiva o meno, in caso fosse attiva cerca di connettersi e scaricare altri malware.