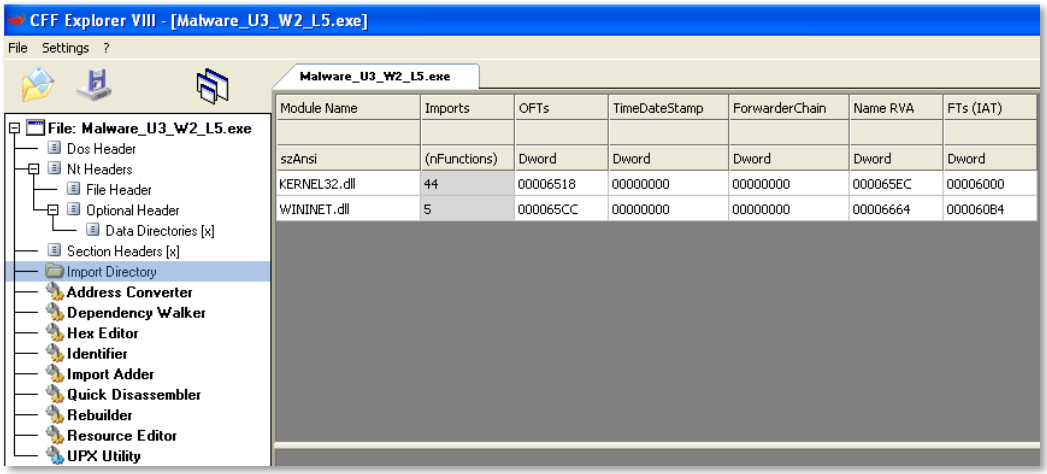
Il progetto odierno prevede l'analisi di un Malware per cercare di capire esattamente il suo comportamento al fine di rimuoverlo dal sistema.

In particolare nella prima parte di questo report andremo ad effettuare un'analisi statica basica, cioè senza eseguirlo tramite l'utilizzo del tool CFF Explorer.

- **Librerie importate dal file eseguibile:**



Le librerie importate dal file sono 2:

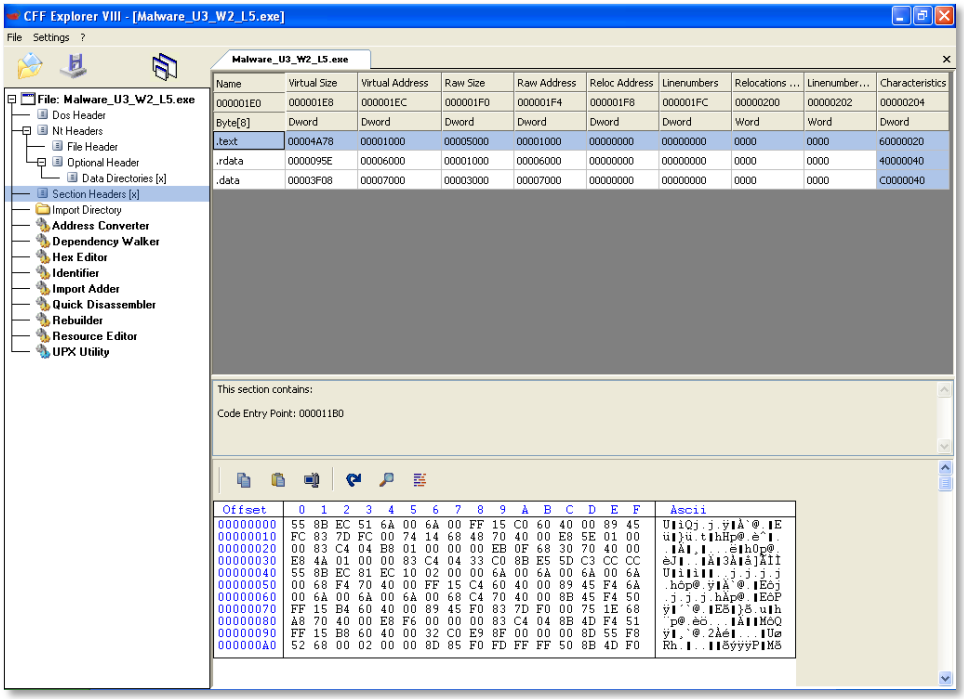
1) **KERNEL32.dll:** contiene le funzioni principali per interagire col sistema operativo, ad esempio: manipolazione dei file, gestione della memoria;

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep
00006940	00006940	027C	SetStdHandle
0000692E	0000692E	0156	GetStringTypeW
0000691C	0000691C	0153	GetStringTypeA
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA
000068E6	000068E6	01E4	MultiByteToWideChar
00006670	00006670	00CA	GetCommandLineA
00006682	00006682	0174	GetVersion
00006690	00006690	007D	ExitProcess
0000669E	0000669E	029E	TerminateProcess

2) **WININET.dll:** contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP,NTP.

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

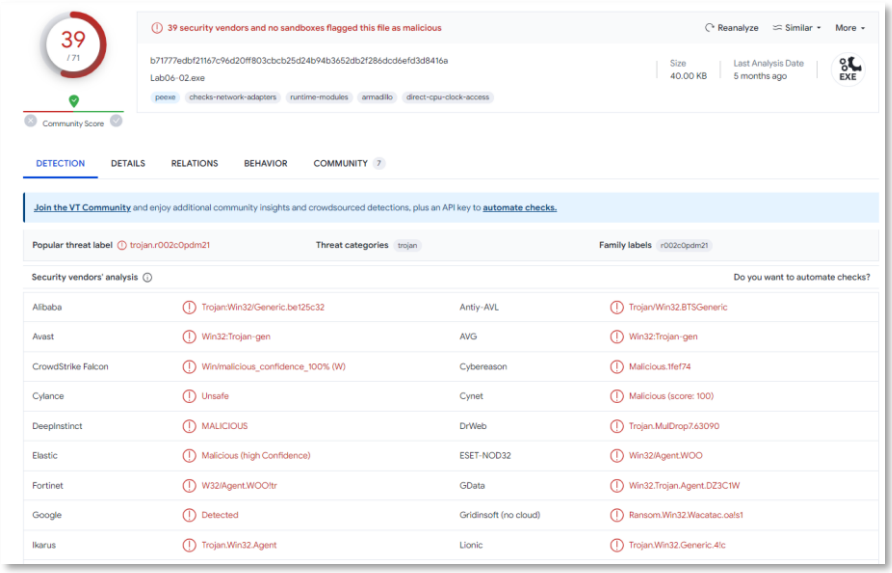
- **Sezioni di cui è composto il file eseguibile:**



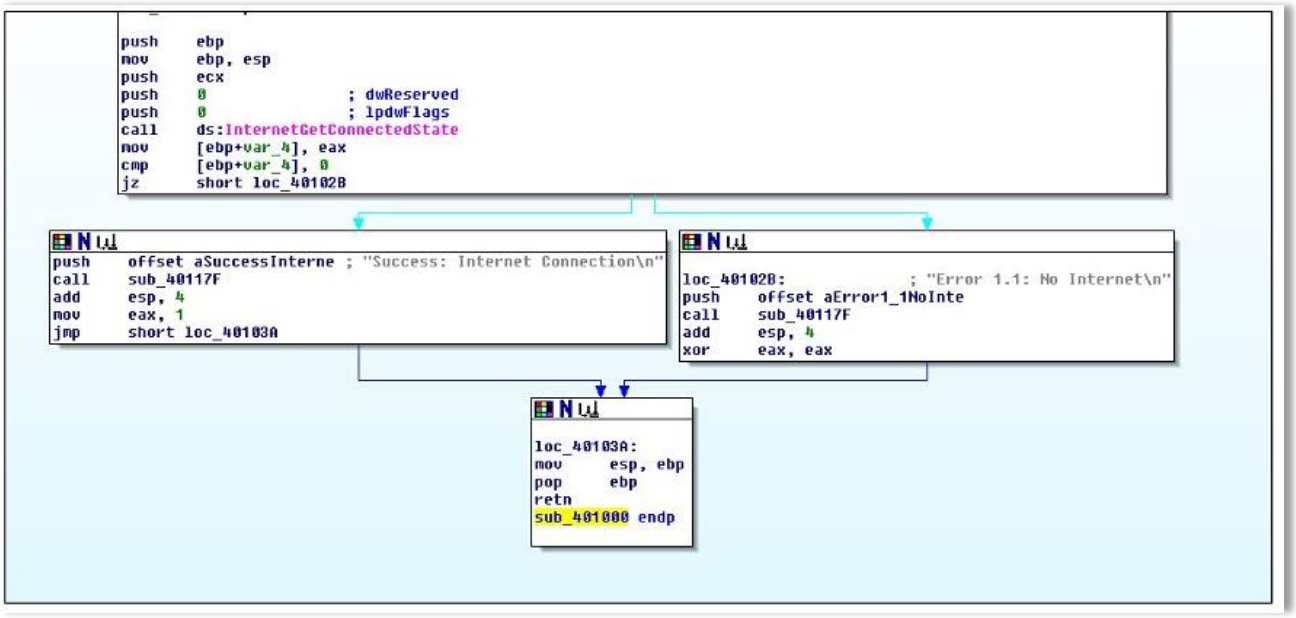
Le sezioni identificate sono 3:

1) **.text:** contiene le istruzione che la CPU andrà ad eseguire una volta avviato il software;
2) **.rdata:** contiene le informazioni su librerie e funzioni importate ed esportate;
3) **.data:** contiene solitamente i dati e le variabili globali del programma.

Inoltre tramite l'hash del file abbiamo potuto constatare che il malware fosse già noto in diversi database antivirus e identificato come trojan. [fonte Virustotal.com]



Nella seconda parte del report, con riferimento alla figura sottostante, verrà analizzato il codice assembly nello specifico riferito all'architettura x86 (32bit).



Abbiamo individuato i costrutti noti:

- **Creazione dello Stack:** è stato possibile individuarlo grazie alla presenza dell'istruzione Push che permette la creazione dello stack tramite l'utilizzo dei registri EBP,ESP che indicano rispettivamente e cima;

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

- **Ciclo IF:** è composto da una prima istruzione Cmp che compara 2 valori, successivamente tramite l'istruzione Jump (jump zero) effettua un salto condizionale verso l'indirizzo di memoria indicato se lo Zero Flag sarà uguale a 1, cioè se i 2 valori comparati saranno uguali.

```
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

- **Rimozione dello Stack:** l'individuazione di questo costrutto è stato possibile grazie alla presenza dell'istruzione Pop che permette la rimozione dello stack sempre abbinato ai registri EBP,ESP visti in precedenza.

```
mov     esp, ebp
pop     ebp
```

Ipotesi comportamento della funzionalità implementata:

Un importante elemento visibile in questo codice è la presenza della funzione "InternetGetConnectedState" che indica se sulla macchina è presente una connessione.

Tramite il costrutto del ciclo IF visto in precedenza viene effettuato un controllo sulla funzione e a seconda del parametro restituito (=0 ,!= 0) ci visualizzerà a schermo la presenza o meno di una connessione Internet, tramite i messaggi "Success: Internet Connection" o "Error 1.1: No Internet".

Il frammento di codice analizzato potrebbe indicare la presenza di una Backdoor.

BONUS – tabella con significato delle singole righe di codice assembly:

CODICE	SIGNIFICATO
Push ebp	Crea lo stack inserendo il valore del registro di base
Mov ebp, esp	Copia il valore di esp in ebp
Push ecx	Inserisce il valore di ecx in cima allo stack
Push 0 ; dwReserved	Inserisce un valore 0 nello stack per inizializzare il parametro dwReserved
Push 0 ; IpdwFlags	Inserisce un valore 0 nello stack per inizializzare il parametro IpdwFlags
Call ds:InternetGetConnectedState	Chiama la funzione indicata
Mov [ebp+var_4], eax	Copia il valore del registro eax nella posizione di memoria indicata
Cmp [ebp+var_4], 0	Compara il valore 0 con il valore precedentemente copiato in quella posizione di memoria per ricavare lo ZF
Jz short loc_40102B	Effettua un salto condizionale in un indirizzo di memoria vicino se lo ZF è = 1
Push offset aSuccessInterne ; "Success Intern Connection\n"	Viene inserito sullo stack l'indirizzo di memoria per la stringa indicata nei doppi apici
Call sub_40117F	Viene chiamata la subroutine che si trova a quell'indirizzo
Add esp, 4	Somma 4 al valore contenuto nel registro esp
Mov eax, 1	Copia 1 nel registro eax
Jmp short loc_40103A	Salto non condizionato all'indirizzo di memoria indicato
Loc_40102B: ; "Error 1.1: No Internet\n"	All'indirizzo di memoria indicato è presente la stringa contenuta nei doppi apici
Push offset aError1_1NoInte	Inserisce nello stack l'indirizzo di memoria per la stringa indicata
Call sub_40117F	Viene chiamata la subroutine che si trova a quell'indirizzo
Add esp, 4	Aggiunge 4 al registro esp
Xor eax, eax	Esegue lo xor per il registro eax per azzerarlo
Loc_40103A:	Indirizzo di memoria
Mov esp, ebp	Copia il valore di ebp in esp
Pop ebp	Rimuove lo stack togliendo il valore
Retn	Equivale al return di una funzione
Sub_401000 endp	Indica la fine di una subroutine