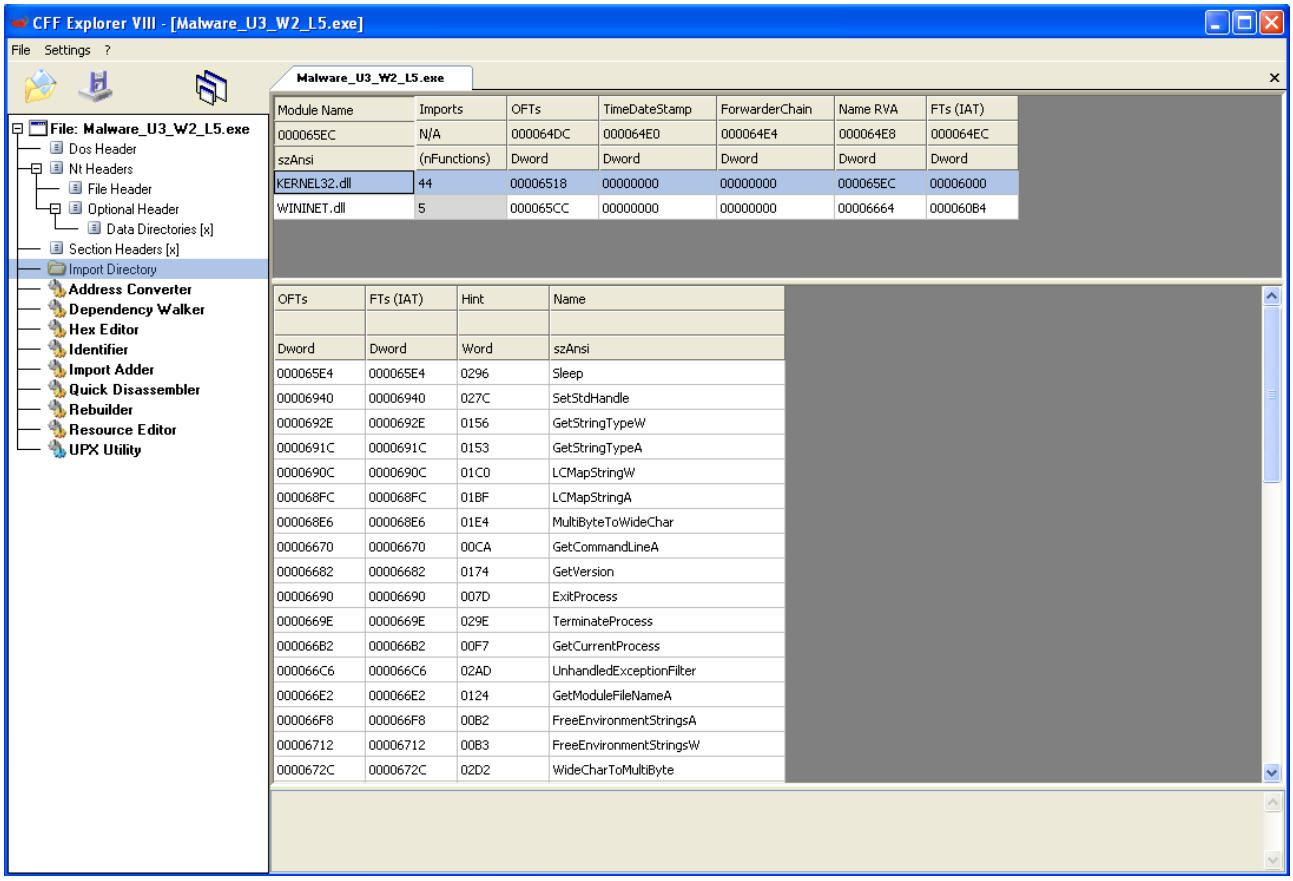
Analisi Statica Malware e Analisi Codice Assembly

Nell'esercizio di oggi andremo ad effettuare **l'analisi statica basilare** di un malware, denominato Malware_U3_W2_L5, per determinarne le **librerie** importate e le **componenti**, oltre ad analizzare un frammento di **codice Assembly** per determinarne i costrutti noti ed ipotizzarne la funzione.

Analisi statica di base del malware

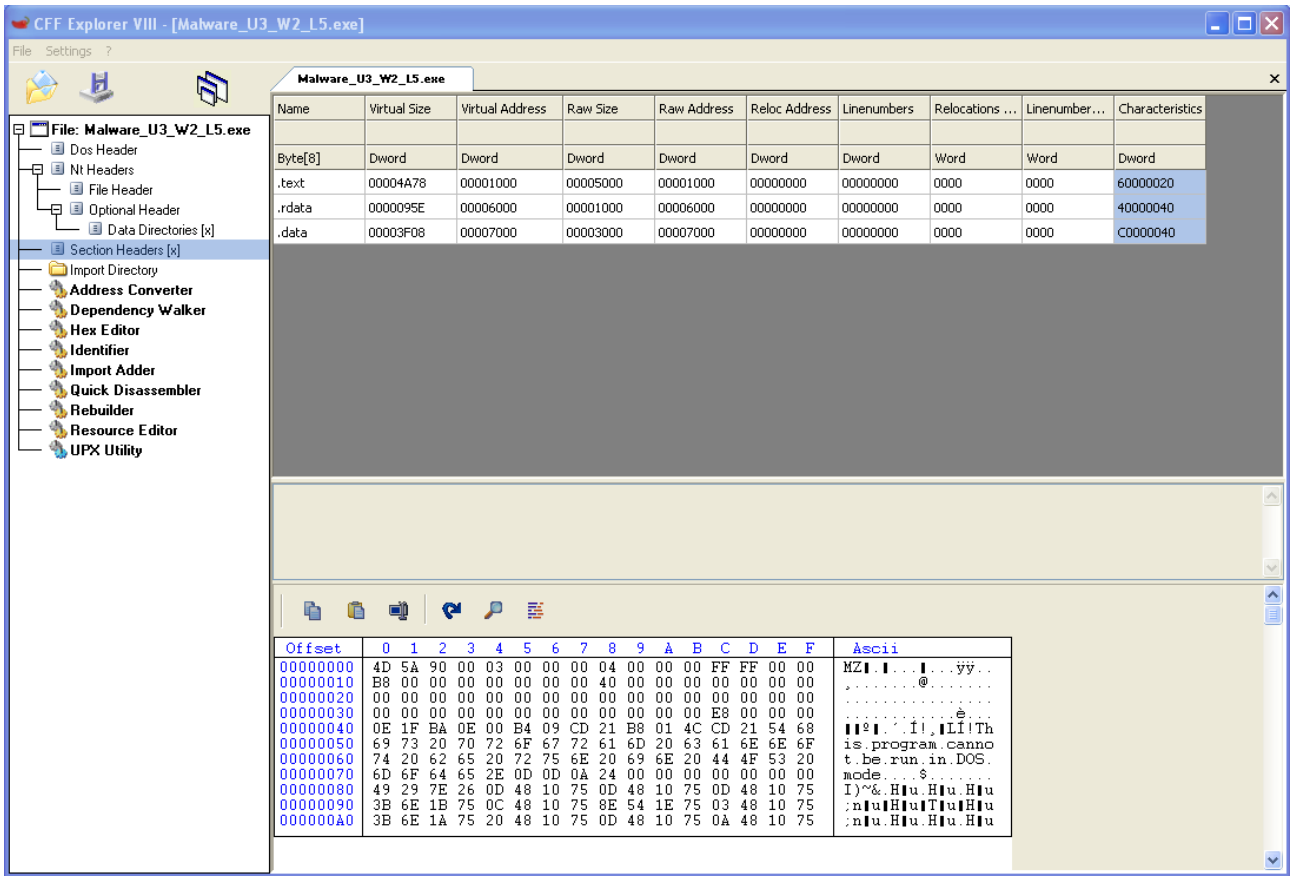
Innanzitutto, analizziamo il malware in questione utilizzando il software **CFF Explore**, il quale ci permette di analizzare la struttura di un file **PE** (portable executable, non è raro trovare malware in questo formato): il risultato dell'analisi al netto delle **librerie importate** è riportato nell'immagine seguente.



Notiamo come il malware importi due librerie, in particolare:

- **kernel32.dll** : una delle librerie di Windows più usata, si occupa di regolare tra le altre cose la gestione della memoria, le operazioni di input ed output, la creazione di processi, in generale ciò che permette ad un programma di poter essere eseguito sul sistema operativo;
- **wininet.dll** : si occupa di gestire l'utilizzo di protocolli di rete quali ad esempio HTTP, HTTPS, FTP; viene quindi utilizzata dalle applicazioni che richiedono un accesso ad internet.

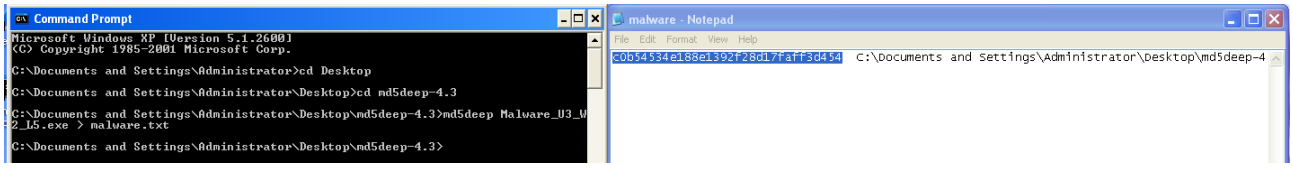
Utilizzando sempre CFF Explorer, possiamo inoltre controllare le **sezioni** di cui il malware si **compone**, come illustrate nella successiva immagine.

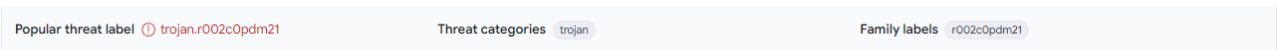


In particolare, le sezioni importate sono:

- .**text** : contiene le istruzioni che la CPU eseguirà una volta che il software sarà avviato;
- .**rdata** : include le informazioni riguardo le librerie e le funzioni importate ed esportate dall'eseguibile;
- .**data** : contiene le variabili globali dell'eseguibile, ossia quelle che devono essere accessibili da qualsiasi parte del programma.

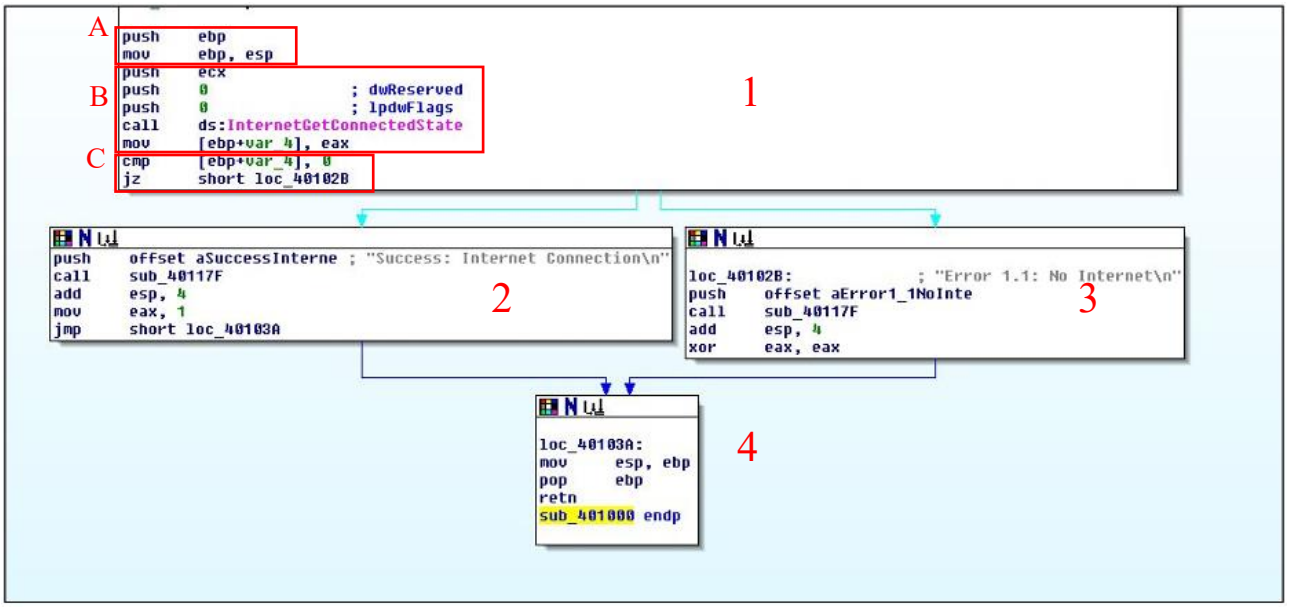
In base all'analisi effettuata finora, possiamo ipotizzare che il malware preso in esame sia un **downloader**, il quale una volta avviato andrebbe ad **instaurare una connessione internet** con un dominio al fine di scaricare ulteriori software dannosi. Per averne un'ulteriore conferma, ricaviamo l'**hash** dell'eseguibile con **md5deep** e la confrontiamo sulla piattaforma online **VirusTotal**, il quale conferma l'ipotesi specificando che si tratta di un **trojan**, come mostrato nell'immagine seguente.





Analisi codice Assembly

Di seguito analizzeremo un frammento di codice **Assembly** identificandone i **costrutti** ed ipotizzandone la **funzione**. Il codice preso in esame è mostrato qui di seguito.



1. **Crea lo stack** (A) e presenta un **costrutto if** (C), per cui il risultato della **funzione richiamata** (B) "indirizza" il flusso dell'esecuzione, dopo un paragone: se la risposta è positiva, sarà eseguito il codice della parte 2, altrimenti sarà eseguita la sezione 3.
2. Viene richiamata una stringa di testo nello stack e successivamente viene chiamata una **funzione**, probabilmente una **printf** per stampare la stringa indicante la presenza di una connessione ad internet. L'indicatore di **stack** viene **aggiornato** e si effettua un salto verso la parte finale della funzione.
3. Come al punto 2, viene probabilmente stampata a schermo la stringa indicata, contenente un messaggio di errore, e lo **stack aggiornato**; eax viene impostato a 0.
4. La **funzione viene chiusa** e lo **stack eliminato**.

Ipoteticamente, la porzione di codice presa in esame serve a **verificare la presenza o meno di una connessione ad internet**, **stampando** a schermo una **stringa testuale** recante la **conferma** di tale connessione (e settando il valore di ritorno della funzione a 1) o un **messaggio di errore** qualora la connessione non vi sia (settando il valore di ritorno della funzione a 0).

Come esercizio addizionale, andremo ad analizzare ogni riga del codice presentato nella precedente immagine:

- **push ebp** - salva l'attuale Extended Base Pointer nello stack;
- **mov ebp, esp** - copia l'indicatore dello stack nell'indicatore della base dello stack. Considerando entrambi i passi, viene quindi aperto un nuovo stack.
- **Push ecx** - viene introdotto il registro ecx
- **push 0 ; dwReserved**
  push 0 ; lpdwFlags
  call ds:InternetGetConnectedState - viene chiamata la funzione ds:InternetGetConnectedState. Le due variabili introdotte nello stack sono necessarie al funzionamento della funzione, così come il registro ecx precedentemente introdotto.

- **mov [ebp+var_4], eax** - copia il valore del registro eax (ossia il risultato della funzione precedente: 1 per connessione presente, 0 per connessione assente) nel registro all'indirizzo indicato.
- **cmp [ebp+var_4], 0** - effettua un paragone tra 0 e il registro all'indirizzo indicato. Se c'è connessione, i due valori saranno diversi (1 per l'argomento nella destinazione, 0 per la sorgente), il valore restituito sarà 1 e la **Zero Flag NON** sarà quindi **settata** (**ZF = 0**). Se non c'è connessione, i valori sono uguali (0 per entrambi gli argomenti), l'output sarà 0 e la **Zero Flag** quindi **è settata** (ossia **ZF = 1**). Ciò sarà importante nel prossimo passaggio.
- **jz short loc_40102B** - analogo del ciclo if, **jz** ci dice di saltare all'indirizzo di memoria indicato se la Zero Flag è settata (ZF = 1), altrimenti il salto non avviene.

Se c'è connessione (ZF = 0):

- **push offset aSuccessInterne ; "Success: Internet Connection\n"** - introduce nello stack l'indirizzo della stringa.
- **call sub_40117F** - chiama una subroutine non specificata, probabilmente una di stampa per la stringa precedente.
- **add esp, 4** - ripulisce lo stack dopo la chiamata, spostando l'esp di 4 byte (ossia 32 bit, la grandezza di un registro standard nei sistemi 32 bit appunto).
- **mov eax, 1** - imposta eax come 1, ossia la funzione ritornerà come risultato 1.
- **jmp short loc_40103A** - effettua un salto non condizionale all'indirizzo di memoria indicato.

Se non c'è connessione (ZF = 1):

- **loc_40102B:** - indica l'indirizzo di memoria a cui si trova la funzione seguente ed introduce nello stack la stringa.
- **push offset aError1_NoInte ; "Error 1.1: No Internet\n"** - inserisce nello stack la stringa indicata.
- **call sub_40117F** - richiama la stessa subroutine di prima, probabilmente per stampare la stringa richiamata alla riga sopra.
- **add esp, 4** - come precedentemente visto, ripulisce lo stack dopo la chiamata della funzione.
- **xor eax, eax** – utilizzando questo operatore logico e la stessa destinazione e sorgente, il registro eax viene azzerato; diversamente da quanto visto prima verrà quindi restituito dalla funzione 1.

Chiusura della funzione:

- **loc_40103A:** - indica l'indirizzo di memoria del codice che segue.
- **mov esp, ebp** – copia l'indicatore della base dello stack nell'indicatore dello stack, andando di fatto a "scartare" e "chiudere" lo stack che abbiamo visto fino ad ora
- **pop ebp** – scarta ebp attuale, andando a ripristinare quello originale.
- **retn** – termina la funzione, restituendo il controllo al punto del programma dove la funzione è stata richiamata.

Giuseppe Pariota