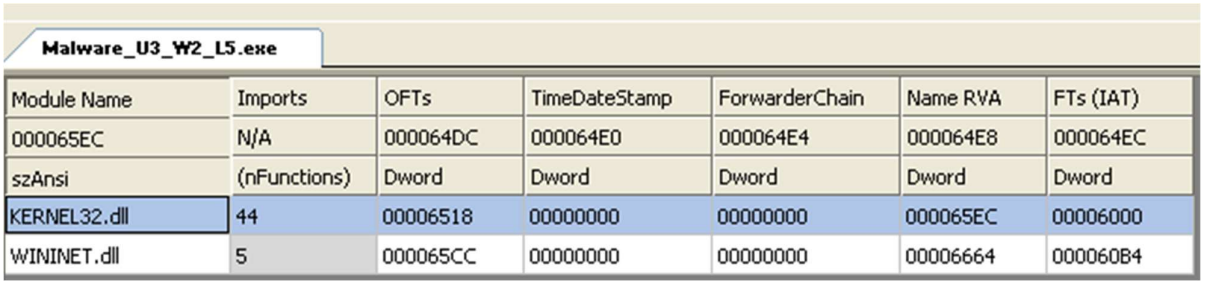
PROGETTO SETTIMANALE S10-L5

ANALISI STATICA BASICA

In riferimento al file eseguibile “Malware_U3_W2_L5” presente nella macchina virtuale dedicata all’analisi dei malware ho eseguito un’analisi statica basica, rispondendo ai quesiti.

QUALI LIBRERIE VENGONO IMPORTATE DAL FILE ESEGUIBILE?

In un’analisi statica basica si cerca di determinare se il file è malevolo o meno senza eseguirlo. Ci sono diversi metodi per analizzare un virus, ma in questo specifico caso, per vedere quali librerie vengono importate utilizzo il programma “CFF Explorer”. Una volta caricato il file clicco sulla sezione “import directory” e vedrò le relative librerie.


Malware_U3_W2_L5.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064E0	000064E4	000064E8	000064EC
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

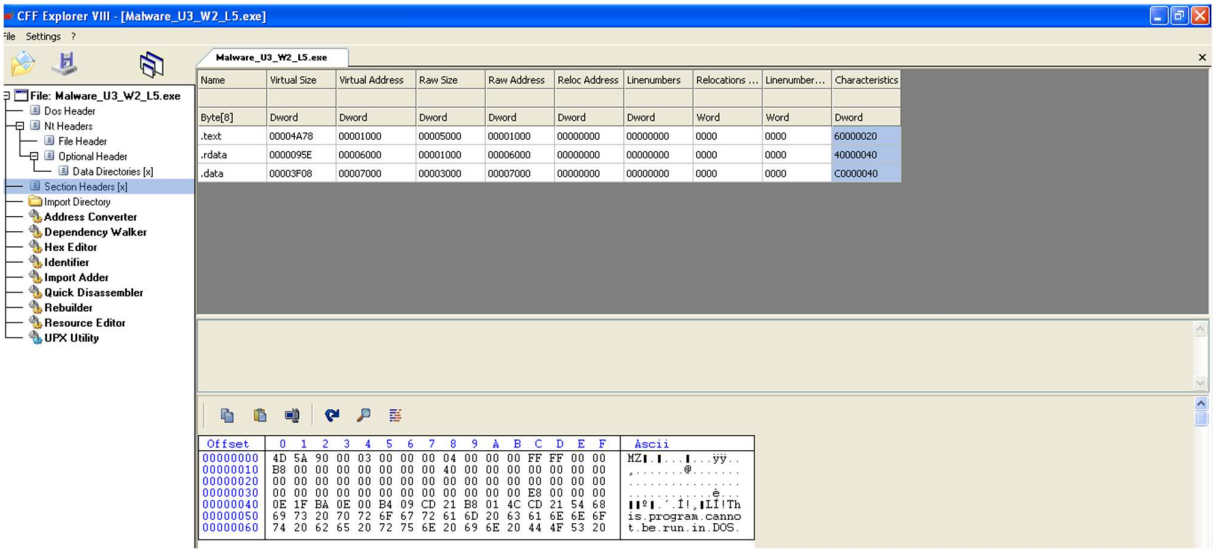
In questo caso le librerie sono due:

- **Kernel32.dll**: si utilizza per interagire con il sistema operativo ad esempio: manipolazione dei file, gestione della memoria e così via.
- **Wininet.dll**: contiene funzioni per implementare alcuni protocolli di rete, ad esempio HTTP, FTP, NTP.

QUALI SONO LE SEZIONI DI CUI SI COMPONE IL FILE ESEGUIBILE?

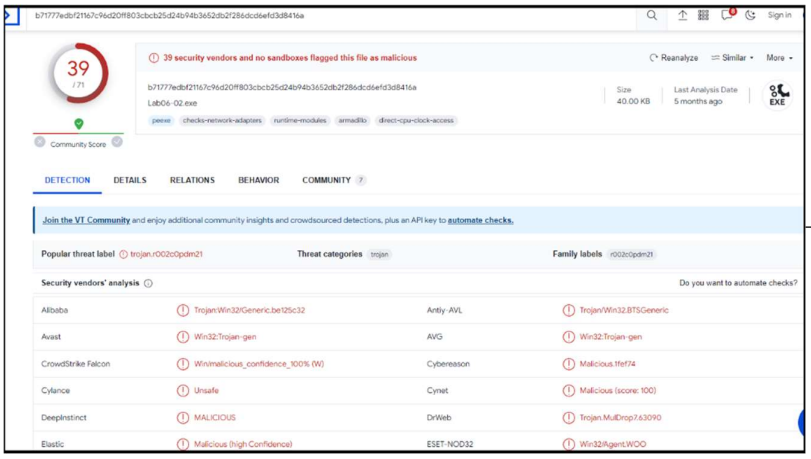
Le sezioni di cui si compone il file eseguibile si possono vedere sempre sul programma “CFF Explorer”.

In questo caso, le sezioni sono le seguenti:



- **.text**: contiene le righe di codice che la CPU eseguirà una volta che il software verrà avviato. Di solito è l’unica sezione di un file eseguibile dalla CPU.
- **.rdata**: include informazioni sulle librerie e funzioni importate ed esportate dall’eseguibile.
- **.data**: contiene dati e variabili globali del programma eseguibile.

Per avere maggiore conferma che effettivamente si tratti di un malware si può estrapolare l’hash del file e caricarlo su “Virus Total” (tool che ci permette di controllare la reputazione del file in questione). L’hash si può vedere sempre tramite “CFF Explorer”, oppure calcolarlo con “md5deep”.

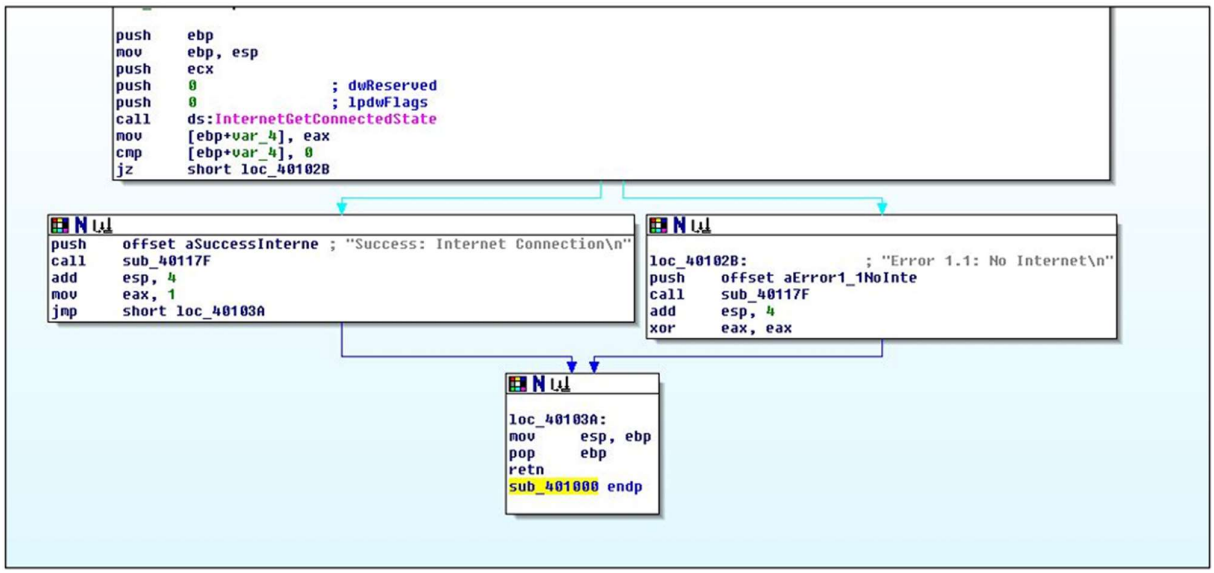


Il tool ha confermato che si tratti di un malware, in particolare di un trojan, un malware che si nasconde in file innocui e che si attiva quando la vittima apre il file.

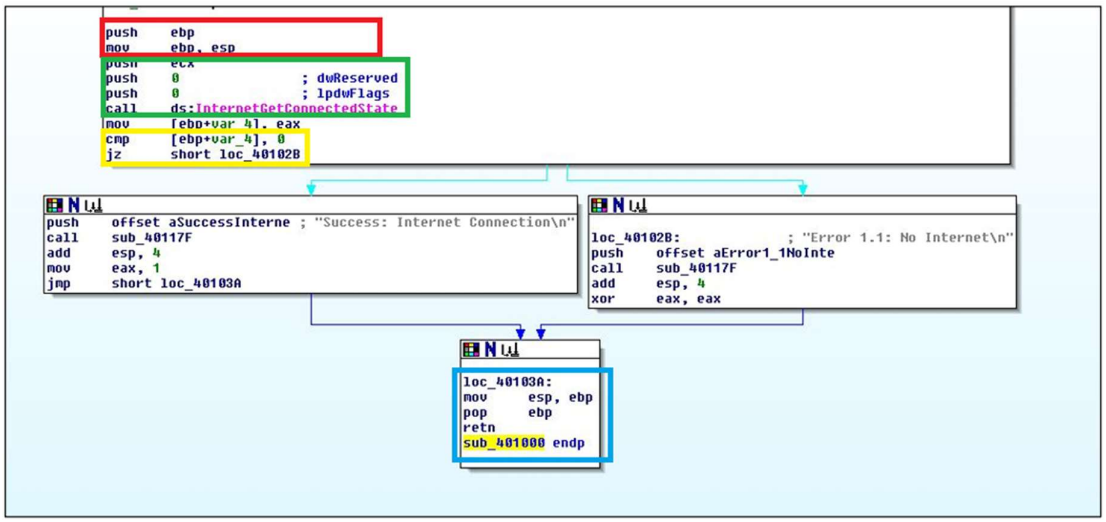
CODICE ASSEMBLY

L'assembly è un linguaggio utile per l'analisi statica avanzata. I programmi che traducono il linguaggio macchina in assembly si chiamano disassembler.

In questo caso il compito da svolgere è quello di identificare i costrutti noti e ipotizzare il comportamento della funzionalità implementata con riferimento alla seguente figura:



1. IDENTIFICARE I COSTRUTTI NOTI



- Il riquadro in **rosso** prevede la creazione dello stack
- Il riquadro in **verde** prevede la chiamata di funzione
- Il riquadro in **giallo** prevede il costrutto IF
- Il riquadro in **azzurro** prevede la chiusura dello stack

2. IPOTIZZARE IL COMPORTAMENTO DELLA FUNZIONALITA' IMPLEMENTATA

Questo estratto di codice controlla lo stato della connessione Internet. Quindi si verifica una condizione simile a un'istruzione IF in linguaggi di programmazione ad alto livello. Questo lo possiamo capire dall'istruzione "jz short loc_40102B" la quale fa un salto condizionale alla posizione di memoria indicata se il risultato della comparazione (cmp [ebp+var_4], 0) è zero, indicando così che la condizione è soddisfatta.

Nel caso in cui la condizione sia soddisfatta il blocco di codice successivo viene eseguito. Questo blocco gestisce il caso in cui la connessione Internet è attiva e stampa un messaggio di successo. In caso contrario, passa al secondo blocco e stampa un messaggio di errore.

In tutti e due i casi il programma esegue la chiusura dello stack per ripristinare lo stato precedente e tornando al chiamante.

3. SIGNIFICATO DELLE SINGOLE RIGHE DI CODICE

push ebp	Salva il valore di “ebp” nello stack.
mov ebp, esp	Imposta “ebp” con il valore corrente dello stack.
push ecx	Salva il valore “ecx” nello stack.
push0; dwReserved	Mette 0 nello stack, cioè il valore “dwReserved” che verrà passato alla funzione “InternetGetConnectedState”.
push0; lpdwFlags	Mette un altro 0 nello stack, questa volta il valore del parametro lpdwFlags.
Callds: InternetGetConnectedState	Chiama la funzione.
mov [ebp+var_4], eax	Memorizza il valore di ritorno della funzione “eax” nella variabile locale [ebp+var_4].
cmp [ebp+var_4], 0	Compara il valore memorizzato in [ebp+var_4] con 0.
jz short loc_40102B	Esegue un salto condizionale alla locazione indicata se il confronto precedente è vero.
Push offset aSuccessInterne; "Success:InternetConnection\n	Mette l'indirizzo della stringa "Success: Internet Connection\n" nello stack. Un messaggio di successo che verrà stampato o utilizzato.
call sub_40117F	Chiama la funzione che ha l'indirizzo sub_40117F.
add esp, 4	Aggiunge 4 al registro “esp”. Questo è probabilmente un modo per "ripulire" lo stack dopo la chiamata della funzione.
mov eax, 1	Mette il valore 1 nel registro eax.
jmp short loc_40103A	Salto incondizionato (jump) alla locazione loc_40103A.
loc_40102B	inizio di un blocco di istruzioni che gestisce il caso in cui non c'è connessione a Internet.
push offset aError1_NoInte	Mette l'indirizzo della stringa "Error 1.1: No Internet\n" nello stack.
call sub_40117F	Chiama la funzione che ha l'indirizzo sub_40117F.
add esp, 4	Aggiunge 4 al registro “esp”. Come detto precedentemente è un modo per “ripulire” lo stack dopo la chiamata della funzione.
xor eax, eax	Imposta “eax” a 0.
loc_40103A	Posizione specifica nel codice.
mov esp, ebp	Muove il valore corrente di 2ebp” nel registro “esp”.
pop ebp	Estrae il valore superiore dello stack e lo colloca nel registro ebp. Questa operazione si fa per ripristinare il valore originale di “ebp”.
retn	Restituisce il controllo al chiamante.
sub_401000 endp	Indica la fine della procedura.