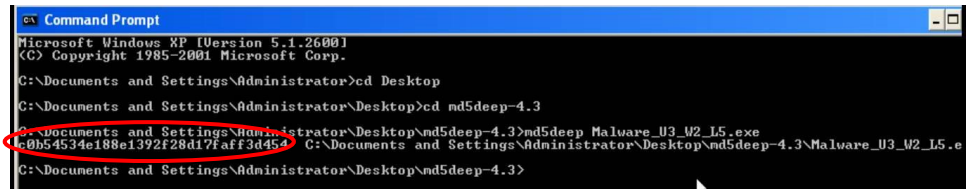
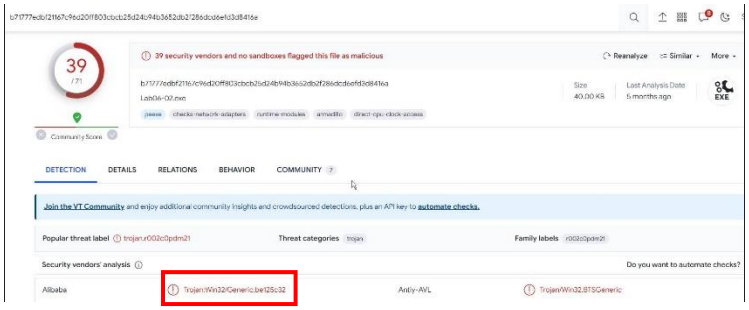
INTRO

Il progetto odierno ha lo scopo di effettuare un'analisi del Malware dato così da ipotizzare il comportamento, capire quali librerie andrà ad importare l'eseguibile e quali sezioni compongono il Malware.
A tal proposito andremo ad utilizzare VirusTotal ed il tool CFFExplorer sulla nostra macchina XP per l'analisi Malware.

TASK 1. LIBRERIE IMPORTATE

Il Malware è un eseguibile in formato PE (Portable Executable) che contiene le informazioni necessarie al sistema operativo per gestire il codice tra cui le librerie. All'interno di esse sono presenti funzioni importanti che il programma andrà a 'chiamare' nel momento in cui ne avrà bisogno. Possiamo andare ad identificarle possiamo estrapolare il codice hash dell'eseguibile con 'md5deep' per poi analizzarlo con VirusTotal come in figura:





Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	19064	20480	6.37	4b8aaeb128744c00b1f9b29dd120616e	196535.5
.rdata	24576	2398	4096	3.66	e5e39acc53e64c50fa5a35693a911478	304856
.data	28672	16136	12288	0.7	305514f6ece00473b7ff8bc023f57e15	2765274

Imports

+ KERNEL32.dll
+ WININET.dll

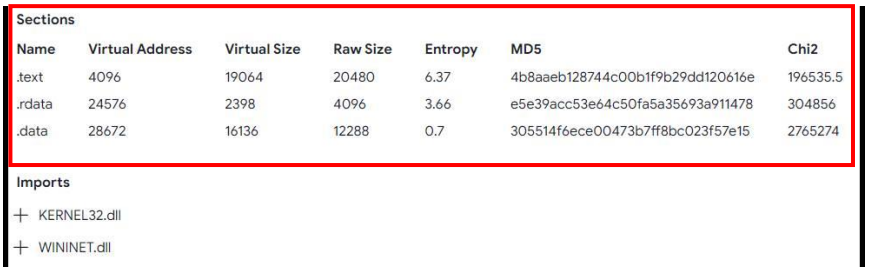
Le librerie importate sono:

- KERNEL32.dll: è una libreria che contiene tutte quelle funzioni che, in generale, servono per interagire con il sistema operativo (gestione della memoria, manipolazione di file etc.);
- WININET.dll: è una libreria che contiene le funzioni principali per l'implementazione dei protocolli HTTP, FTP, NPT.

Oltre alle librerie, VirusTotal ci fornisce una descrizione del comportamento del Malware: in questo caso è identificato come Trojan.

TASK 2. SEZIONI

Un'altra importante informazione che ci fornisce VirusTotal è riguardante le sezioni che compongono il file eseguibile ed ognuna di esse ha un preciso scopo.



Sections

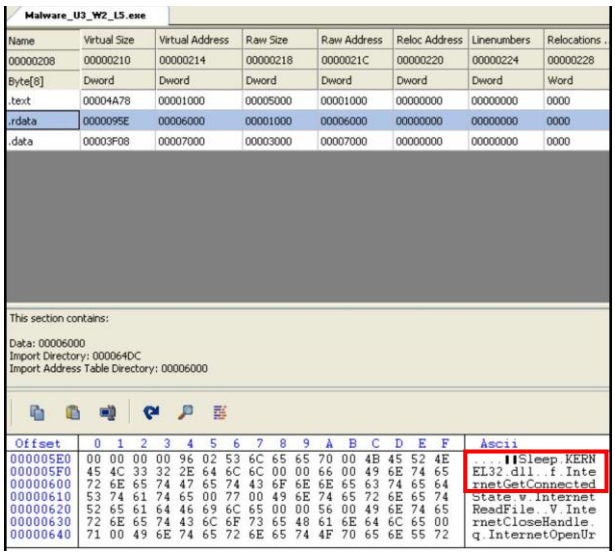
Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	19064	20480	6.37	4b8aaeb128744c00b1f9b29dd120616e	196535.5
.rdata	24576	2398	4096	3.66	e5e39acc53e64c50fa5a35693a911478	304856
.data	28672	16136	12288	0.7	305514f6ece00473b7ff8bc023f57e15	2765274

Imports

+ KERNEL32.dll
+ WININET.dll

Le sezioni che compongono il file sono:

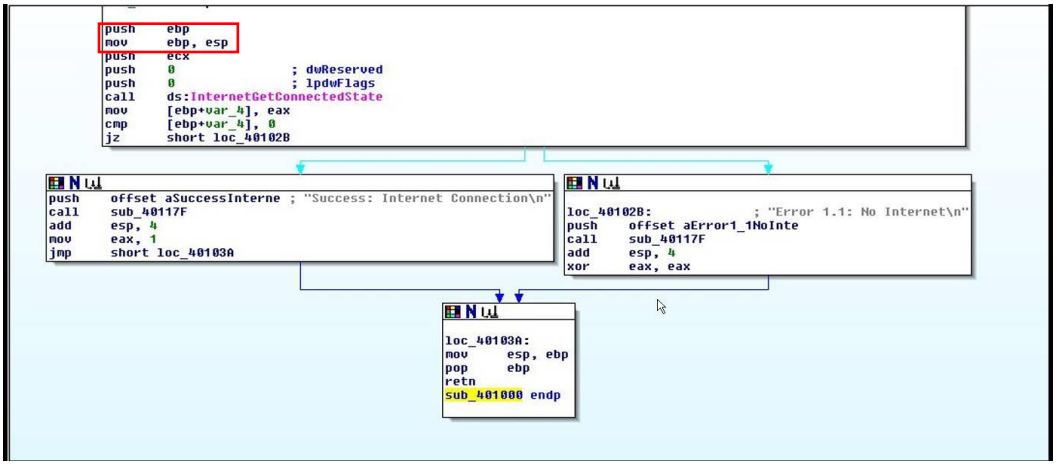
- .text: è la sezione che in genere viene proprio eseguita dalla CPU, poiché contiene le righe di codice da eseguire una volta avviato il file;
- .rdata: include tutte quelle informazioni relative alle librerie importate e, per ricavarle nel dettaglio, abbiamo bisogno di CFFExplorer come in figura sotto;
- .data: la sezione contiene i dati delle variabili/funzioni globali che sono di conseguenza accessibili da qualsiasi funzione all'interno dell'eseguibile.



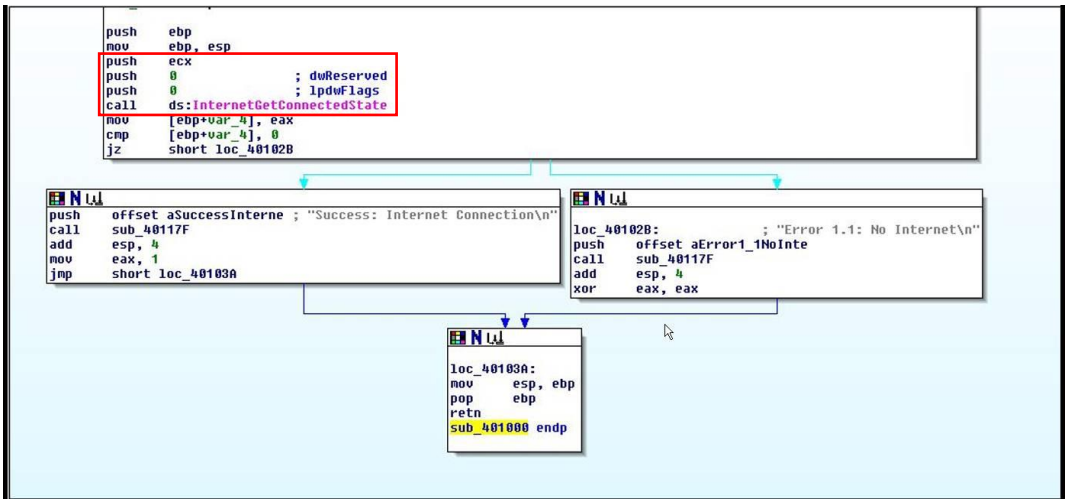
TASK 3. ASSEMBLY E COSTRUTTI NOTI

Facendo riferimento alla figura sottostante, andiamo ad identificare i costrutti noti del codice Assembly.
Il linguaggio Assembly è univoco per una data architettura di un PC e la conoscenza di tale linguaggio servirà per tradurre le istruzioni eseguite dalla CPU in formato leggibile dall'uomo.

Il primo costrutto che possiamo notare è dato dalle prime due righe di codice che va a creare uno stack per le variabili locali. Possiamo anche notare la presenza dei due puntatori EBP (Extended Base Pointer) ed ESP (Extended Stack Pointer) che puntano rispettivamente alla base ed alla cima dello stack:

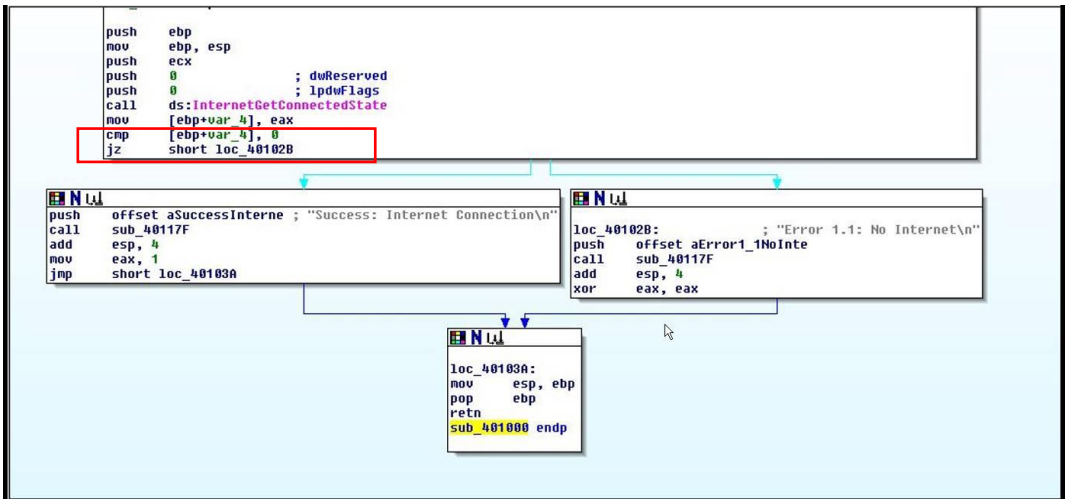


Il secondo rappresenta la chiamata di funzione. I parametri sono passati sullo stack tramite le istruzioni push. Ciò è il modo in cui la funzione chiamante invia i parametri necessari alla funzione chiamata per poter svolgere il suo compito (parametri 'pushati' sullo stack):

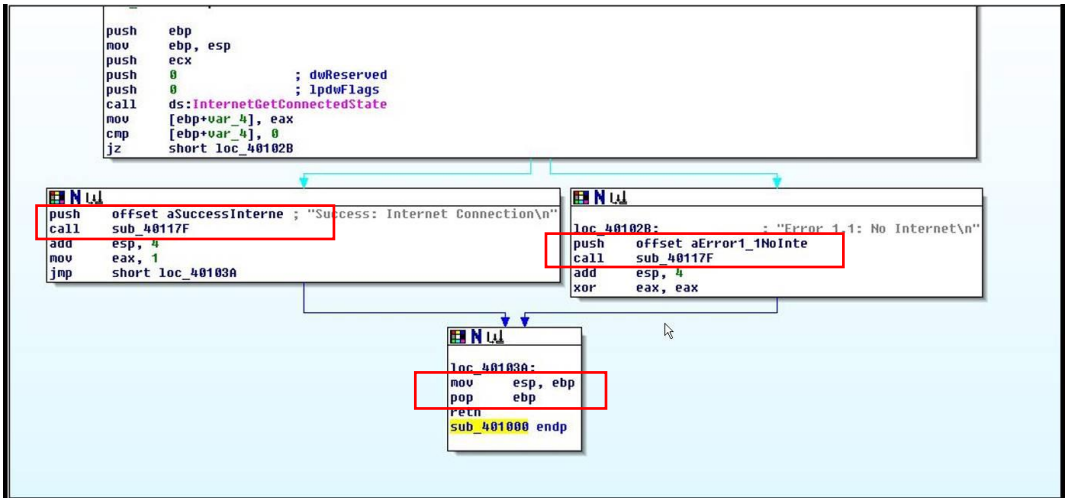


Il terzo blocco che possiamo identificare è un ciclo if. È composto da un 'cmp' e un jump:

- Con cmp compara i valori e se la variabile è uguale a 0, allora salterà all'indirizzo di memoria segnato;
- Se il valore di ritorno della funzione è diverso da 0, allora vuol dire che c'è una connessione attiva.



Infine abbiamo altre chiamate di funzione fino alla rimozione dello stack finale con l'istruzione 'pop':



TASK 4. COMPORTAMENTO DELLA FUNZIONALITA' IMPLEMENTATA

A seguito della lettura del linguaggio Assembly, possiamo ipotizzare il comportamento della funzionalità implementata. Una volta che l'eseguibile malevolo chiamerà la funzione indicata (InternetGetConnectedState), andrà ad eseguire un controllo con il ciclo if e, come detto sopra, se il valore di ritorno è diverso da 0 allora ci sarà connessione e andrà avanti. In caso contrario salterà alla locazione di memoria indicata e darà in OUTPUT l'errore di connessione. Il Malware potrebbe, nel primo caso, andare ad utilizzare la connessione internet per eseguire varie operazioni tra cui il download di un secondo eseguibile malevolo oppure l'installazione di una backdoor.

TASK BONUS. RIGHE DI CODICE ASSEMBLY

- Primo blocco:

push EBP	"Pusha" l'EBP sulla cima dello stack
mov EBP, ESP	copia il valore del registro dell'ESP nel registro EBP
push ECX	mette il valore del registro ECX sullo stack
push 0; dwReserved	mette il valore indicato nello stack
push 0; lpdwflags	mette il valore indicato nello stack
call ds: InternetGetConnectedState	chiamata sulla funzione nota per verificare la connessione
mov [EBP+var_4], EAX	copia il valore del registro dell'EAX nel registro [EBP+var_4]
cmp [EBP+var_4], 0	sottrazione ntra il registro [EBP+var_4] e 0
jz short loc_40102B	controllo della precedente ZF, se =1 ci sarà uno short jump all'indirizzo indicato

- Secondo blocco:

push offset aSuccessinterne: "Success intenet connection\n"	inserimento della stringa
call sub_40105F	chiamata di una funzione in quell'indirizzo di memoria
add esp, 4	somma tra il valore di registro di ESP e 4
mov eax, 1	copia 1 nel registro EAX

- Terzo blocco:

push offset aError1_1NoInte: "Error 1.1: No Internet\n"	inserimento della stringa
call sub_40117F	chiamata di una funzione in quell'indirizzo di memoria
add ESP, 4	somma tra il valore di registro di ESP e 4
xor EAX, EAX	azzera il valore del registro EAX

- Quarto blocco:

mov ESP, EBP	copia il valore di registro dell'EBP nel registro ESP
pop EBP	pulizia dello stack
retn	ritorno da una chiamata di funzione
sub_401000 endp	fine della funzione (end point)