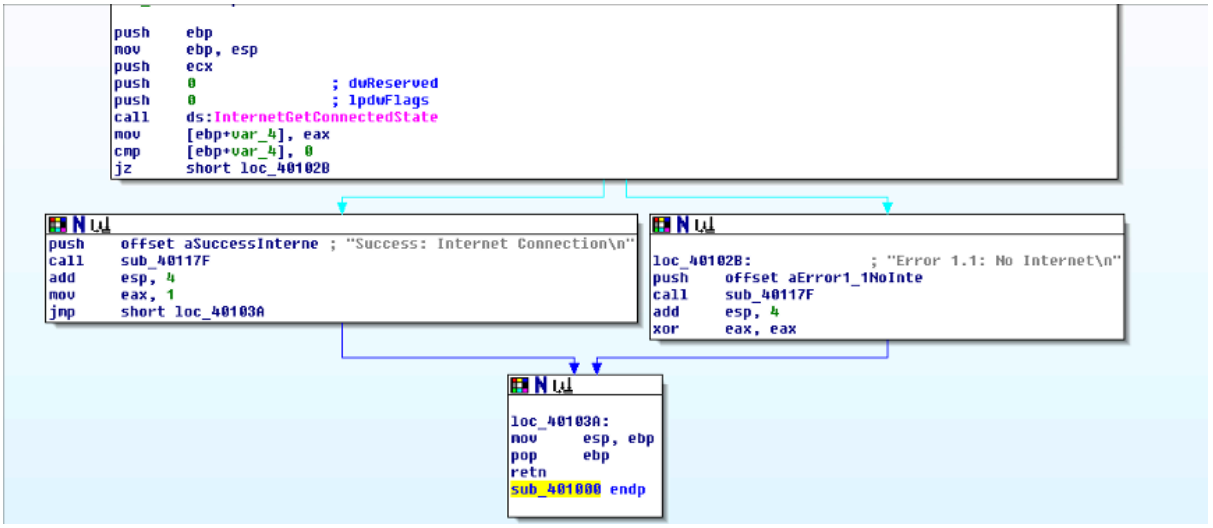
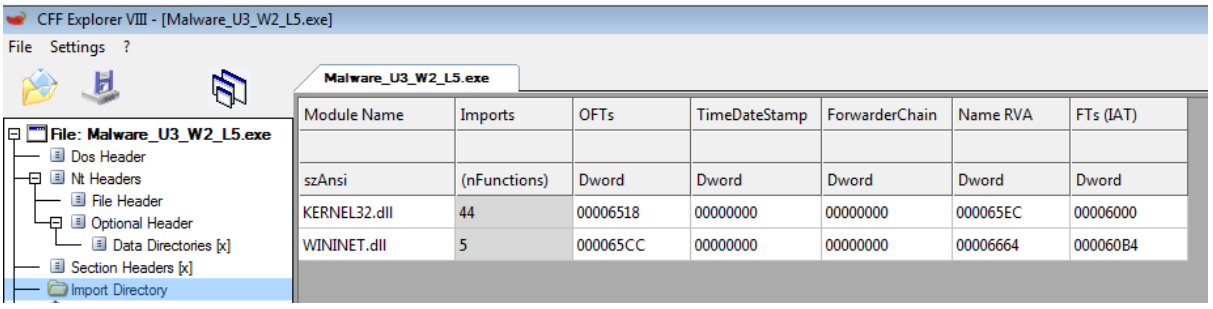
PROGETTO S10/L5

Traccia: con riferimento al file Malware_U3_W2_L5 presente all'interno della cartella «Esercizio_Pratico_U3_W2_L5» sul desktop della macchina virtuale dedicata per l'analisi dei malware, rispondere ai seguenti quesiti:
1. Quali librerie vengono importate dal file eseguibile?
2. Quali sono le sezioni di cui si compone il file eseguibile del malware?
Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:
3. Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti).
4. Ipotizzare il comportamento della funzionalità implementata.
5. BONUS fare tabella con significato delle singole righe di codice assembly.



Svolgimento:
1.



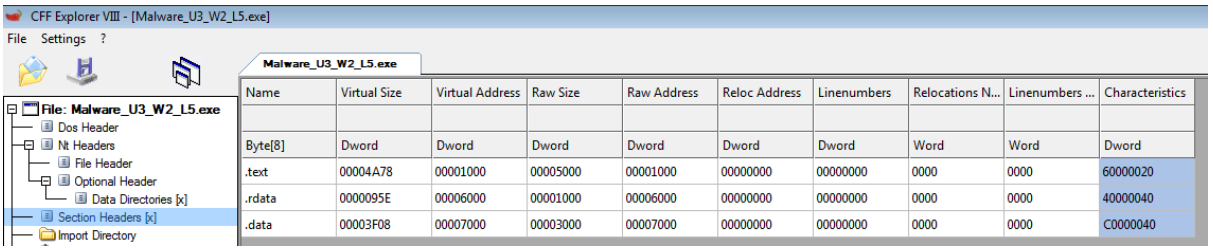
Ci sono due librerie importate da questo file eseguibile:

KERNEL32.dll

WININET.dll

KERNEL32.dll è una libreria di sistema essenziale di Windows che contiene centinaia di funzioni per la gestione di operazioni di basso livello come la gestione della memoria, la creazione e gestione dei processi e dei thread, e altre funzioni di base del sistema operativo.
WININET.dll fornisce funzioni di alto livello per l'interazione con il protocollo HTTP, FTP e Gopher. Viene comunemente utilizzata per creare client che inviano richieste a server web.
La presenza di queste due librerie indica che il file eseguibile potrebbe essere coinvolto nella gestione di processi di base e nella comunicazione di rete.

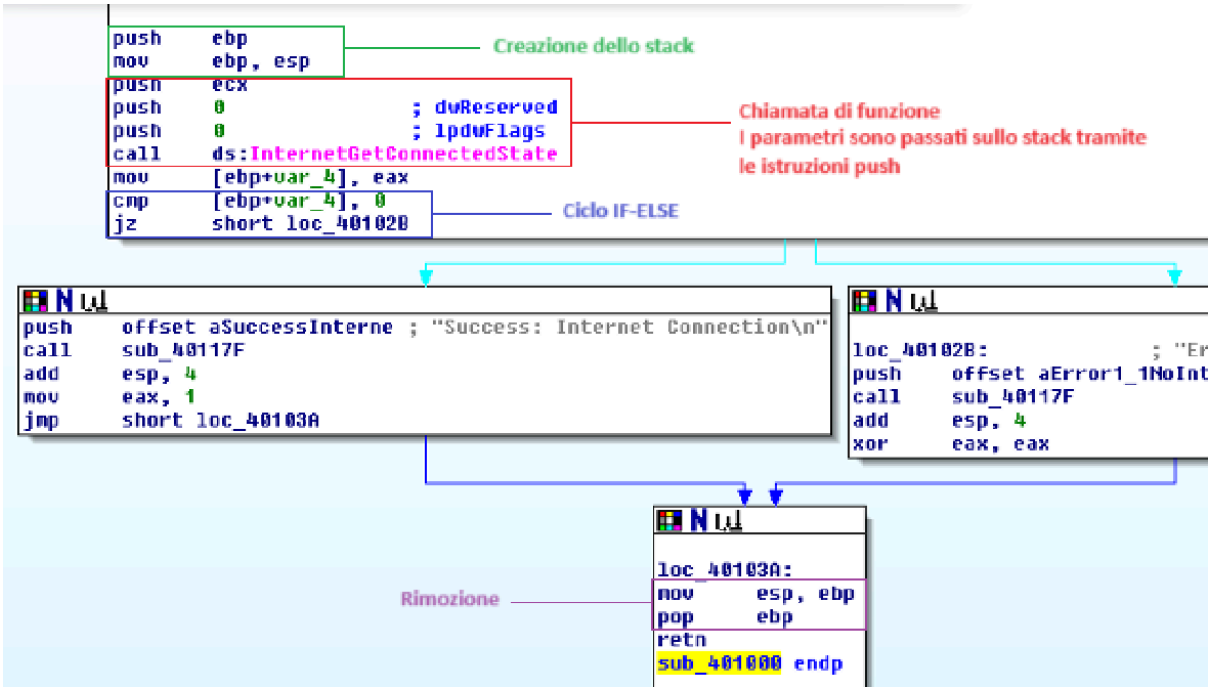
2.



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

Il file eseguibile contiene le seguenti sezioni:
- **.text**: questa sezione contiene il codice eseguibile del programma. È in genere dove risiede il codice macchina che verrà eseguito dal processore.
- **.rdata**: questa sezione include dati leggibili ma non scrivibili, spesso usata per memorizzare le costanti e i dati di sola lettura.
- **.data**: la sezione `.data` è utilizzata per variabili globali e statiche che sono inizializzate dall'editore di collegamenti.
Ognuna di queste sezioni ha diversi attributi che determinano le caratteristiche dei dati che contengono. Ad esempio, la sezione `.text` è di solito contrassegnata come eseguibile e leggibile, ma non scrivibile, per prevenire l'auto-modifica del codice, che è una tecnica di sicurezza comune per prevenire attacchi dannosi. La sezione `.data` è leggibile e scrivibile perché deve contenere dati che possono cambiare durante l'esecuzione del programma. La sezione `.rdata` è solo leggibile e contiene dati di sola lettura.

3 (più Bonus 5). I costrutti noti che ho individuato sono i seguenti:



Questo codice assembly, che è un linguaggio di programmazione a basso livello strettamente correlato al codice macchina, è una funzione che verifica la presenza di una connessione Internet. Ecco una ripartizione dei costrutti e del comportamento:
Creazione del frame dello stack:
push ebp: questa istruzione salva il puntatore base della funzione precedente nello stack.
mov ebp, esp: imposta il frame dello stack spostando il puntatore dello stack nel puntatore base.

push ecx: salva il registro ecx nello stack, probabilmente usato più avanti in questa funzione.
Prologo della funzione:
push 0: questo potrebbe essere l'invio di un argomento a una chiamata di funzione, probabilmente un flag o un parametro specifico.
push offset aSuccessIntern ; "Success: Internet Connection\n": invia l'indirizzo di memoria della stringa "Success: Internet Connection" nello stack, probabilmente per utilizzarlo dopo la chiamata di funzione come messaggio.
push offset aError1_1NoInter ; "Error 1.1: No Internet\n": invia l'indirizzo di memoria del messaggio di errore nello stack, probabilmente da utilizzare in caso di fallimento della successiva chiamata di funzione.
Chiamata di funzione e branching:
call ds:InternetGetConnectedState: chiama una funzione esterna, possibilmente una funzione dell'API di Windows per verificare lo stato della connessione internet.
cmp [ebp+var_4], 0: confrontare una variabile locale con 0 per verificare il risultato della precedente chiamata di funzione.
jz short loc_401028: salta al codice di gestione degli errori se il flag zero è impostato (il che significa che il confronto era vero, e Internet non è connesso).
Esecuzione condizionale:
Se la funzione InternetGetConnectedState restituisce un valore diverso da zero (internet è connesso), non effettua il salto e il codice seguente viene eseguito:
push offset aSuccessIntern: invia l'indirizzo del messaggio di successo nello stack.
call sub_40117F: probabilmente chiama una funzione per stampare o gestire il messaggio di successo.
add esp, 4: pulisce lo stack di 4 byte, la dimensione dell'indirizzo inserito.
nop: No Operation – non fa nulla, spesso utilizzato per allineamento o attesa.
jmp short loc_40103A: salto incondizionato alla fine della funzione.
Gestione degli errori:
loc_401028: etichetta per la gestione degli errori se internet non è connesso.
push offset aError1_1NoInter: mette l'indirizzo del messaggio di errore nello stack.
call sub_40117F: probabilmente chiama la stessa funzione di sopra per stampare o gestire il messaggio di errore.
add esp, 4: pulisce nuovamente lo stack della dimensione dell'indirizzo.
xor eax, eax: imposta il registro eax a 0, spesso usato per restituire un valore di 0 da una funzione.
Epilogo della funzione:
loc_40103A: etichetta per la fine della funzione.
mov esp, ebp: ripristina il puntatore dello stack, rimuovendo effettivamente le variabili locali.
pop ebp: ripristina il puntatore di base al suo valore precedente.
retn: ritorna dalla funzione.

4. Sulla base dell'analisi del codice assembly fatta al punto precedente, la funzione sembra comportarsi nel seguente modo: la funzione inizia *preparando il frame dello stack* per le proprie operazioni. Questo include la memorizzazione del contesto attuale e la preparazione per le variabili locali.
La funzione poi chiama una API di Windows, presumibilmente `InternetGetConnectedState`, la quale *determina lo stato della connessione Internet del computer*. Questa funzione API prende probabilmente dei parametri, uno dei quali è stato spinto nello stack (`push 0`).

Dopo la chiamata all'API, la funzione *controlla il valore di ritorno*. In assembly, il risultato di una funzione di solito viene restituito nel registro `eax`, e le operazioni come `cmp` possono essere utilizzate per confrontare il valore del registro con un altro valore.
Se il risultato del confronto indica che *non c'è connessione Internet* (vale a dire, il valore è zero), il codice eseguirà un salto (`jz`, jump if zero) a una sezione che gestisce questo caso, mostrando o registrando un messaggio di errore ("Error 1.1: No Internet").
Se *c'è connessione a Internet* (il valore non è zero e quindi il salto condizionale non viene eseguito), la funzione procede a una sezione che gestisce questo caso di successo, mostrando o registrando un messaggio di successo ("Success: Internet Connection").
Dopo la gestione del successo o dell'errore, la funzione *pulisce lo stack* (rimuove gli indirizzi dei messaggi che erano stati spinti nello stack) e poi ripristina lo stato dello stack di come era prima della chiamata della funzione (`mov esp, ebp` e `pop ebp`).
Infine, la *funzione termina e ritorna al chiamante* con l'istruzione `retn`. Se il percorso di errore è stato preso, il registro `eax` è stato impostato a zero, che potrebbe indicare un fallimento. Se il percorso di successo è stato preso, il valore di `eax` non è stato esplicitamente impostato nel frammento fornito, ma potrebbe essere stato impostato dalla funzione `sub_40117F` chiamata precedentemente.
In sintesi, il comportamento atteso della funzione è di *verificare la presenza di una connessione Internet* e *segnalare il risultato attraverso un messaggio di successo o errore*.