MALWARE ANALYSIS

1. Obiettivo

Con riferimento al file Malware_U3_W2_L5, rispondere ai seguenti quesiti.

- Quali librerie vengono importate dal file eseguibile?
- Quali sono le sezioni di cui si compone il file eseguibile del malware?

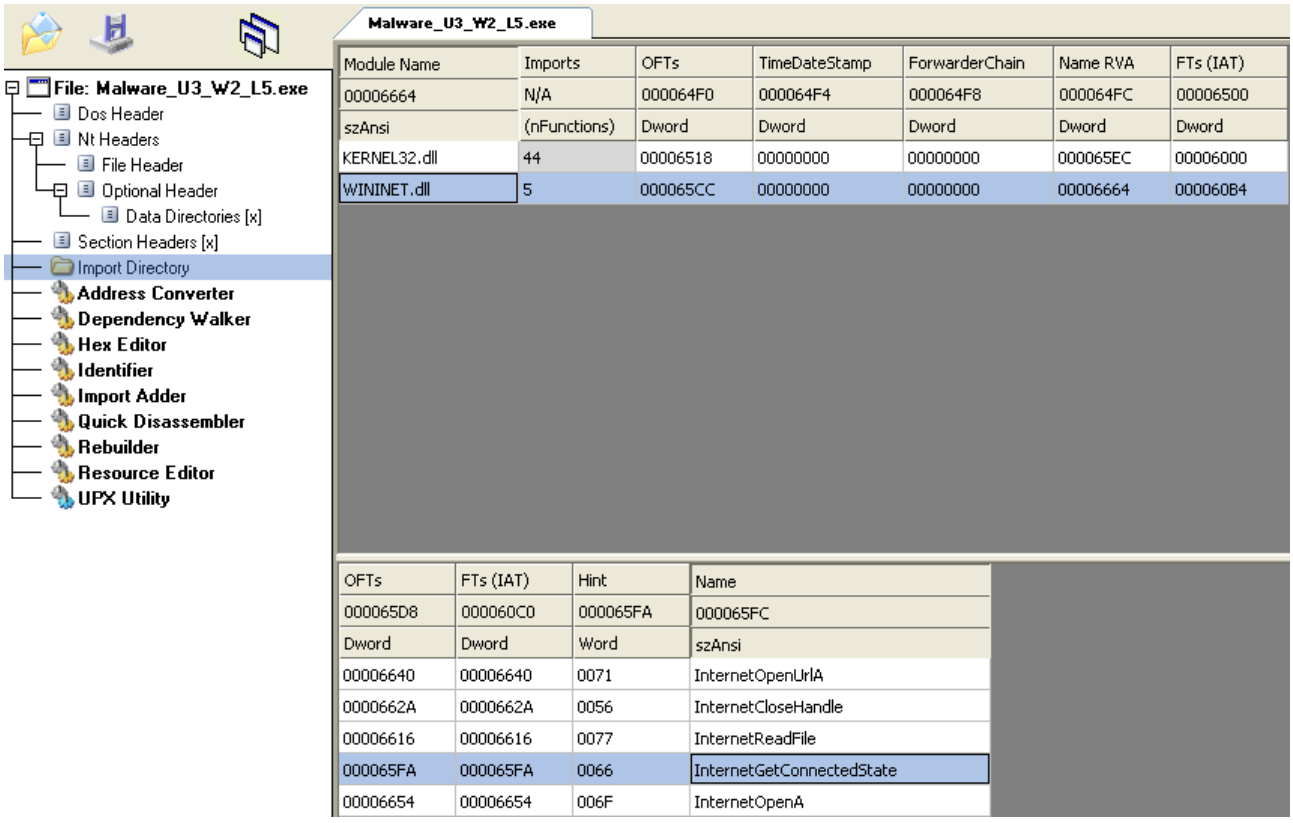
Con riferimento alla figura in slide 3, risponde ai seguenti quesiti.

- Identificare i costrutti noti (creazione dello stack, eventuali cicli, costrutti).
- Ipotizzare il comportamento della funzionalità implementata.
- BONUS: spiegare il significato delle singole righe di codice assembly.

2. Svolgimento

Nella prima parte dell'esercizio di oggi procediamo con un'analisi statica basica del malware per capire quali librerie vengono importate e di quali sezioni si compone il programma.

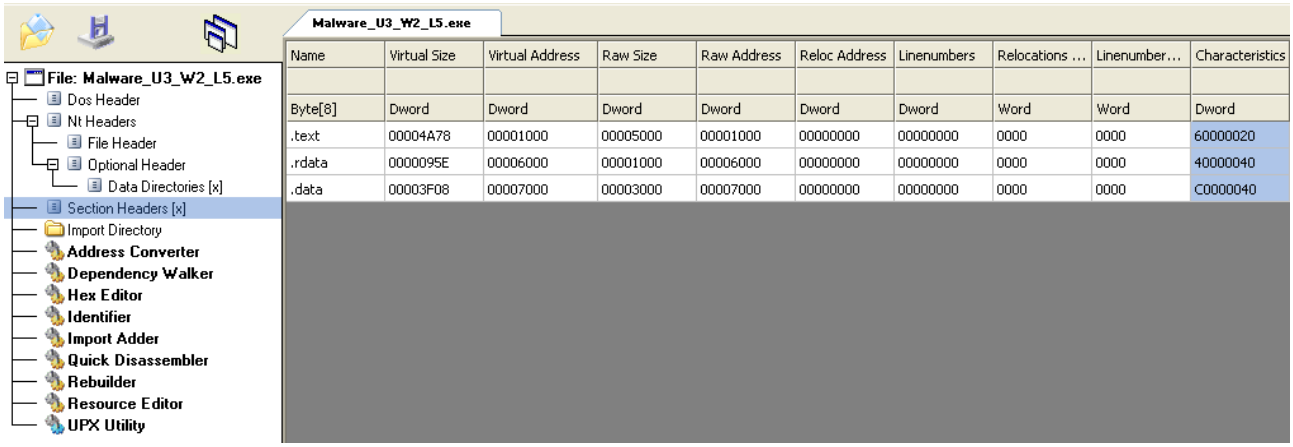
Usiamo il tool CFF Explorer, in cui importiamo il file eseguibile del malware. Nella sezione "Import Directory" possiamo vedere quali sono le librerie importate dal programma. Nel nostro caso, vengono importate due librerie, KERNEL32.dll e WININET.dll.



KERNEL32 è un'importante libreria di Windows che permette di interfacciarsi con il sistema operativo. Un attaccante potrebbe interagire in molti modi con la macchina attaccata attraverso questa libreria, ad esempio potrebbe modificare le chiavi diregistro, creare nuovi processi o terminarne di esistenti, leggere e modificare file, allocare memoria, etc.

La libreria WININET invece contiene funzioni per la comunicazione di rete in Windows. Permette di aprire o chiudere una connessione, inviare dati, gestire le credenziali d'accesso a siti web, etc. In particolare possiamo vedere tra la sue funzioni InternetGetConnectedState, che incontriamo nel codice assembly della seconda parte dell'esercizio e permette di verificare lo stato di connessione alla rete.

Nella sezione "Section Headers" possiamo invece trovare le sezioni di cui si compone il malware. Possiamo vedere che esso è composto da tre sezioni: .text, .rdata e .data.



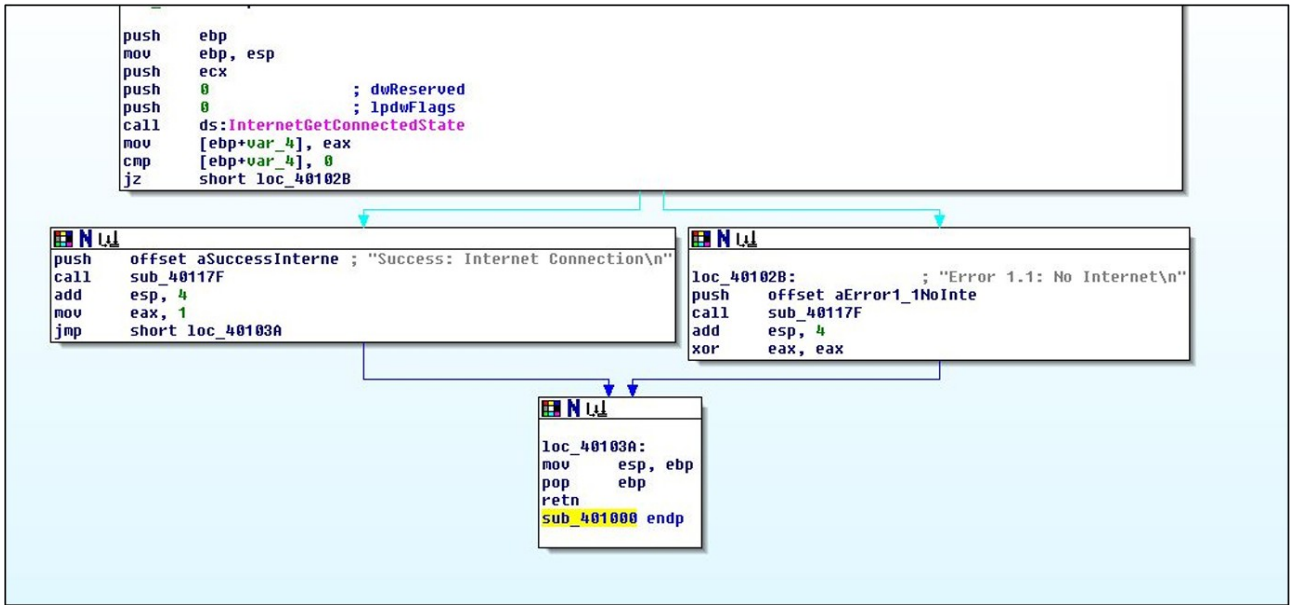
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000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

.text è la parte del programma che porta il codice, quindi le istruzioni che dovrebbe eseguire la CPU una volta lanciato il malware.

.rdata contiene informazioni sulle librerie importate ed esportate dal programma, nonché costanti, valori di inizializzazione delle variabili, tabelle di dati.

.data contiene le variabili globali e statiche del programma necessarie per permettere a questo di funzionare.

Nella seconda parte dell'esercizio andiamo ad analizzare il codice assembly del malware.



Per prima cosa andiamo a vedere quali costrtutti sono presenti nel codice.

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

Queste prime due righe servono per creare lo stack su cui andrà a lavorare la seguente parte di codice.

```
push    ecx
push    0                   ; dwReserved
push    0                   ; lpdwFlags
call    ds:InternetGetConnectedState
```

Questo blocco introduce sullo stack i parametri necessari alla funzione InternetGetConnected per essere chiamata e poi chiama la funzione stessa.

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

Qui viene introdotto un ciclo if else. Viene sottratto 0 al valore di [ebp+var_4] per verificare se il suo valore è 0 e a seconda del risultato la funzione salta ad un'altro indirizzo di memoria della RAM oppure prosegue.

```
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

Quest'ultimo blocco elimina lo stack una volta che la funzione è terminata.

Andiamo ora a vedere cosa fa nello specifico ogni riga di codice.

push ebp	Inserisce sullo stack il pointer EBP, che indica la base dello stack.
mov ebp, esp	Copia il valore del registro ESP (stack pointer) in EBP. Insieme all'istruzione precedente contribuisce a creare un nuovo stack.
push ecx	Inserisce sullo stack il registro ECX.
push 0 ; dwReserved	Inserisce un nuovo parametro che riservato alla funzione InternetGetConnetcedState.
push 0 ; lpdwFlags	Inserisce un nuovo parametro che sarà usato dalla funzione InternetGetConnetcedState come una variabile flag.
call ds:InternetGetConnectedState	Chiama la funzione InternetGetConnectedState.
mov [ebp+var_4], eax	Copia il registro EAX (che contiene il valore di ritorno della funzione InternetGetConnectedState) nella variabile locale var_4
cmp [ebp+var_4], 0	Compara la variabile var_4 con 0
jz short loc_40102B	Salto conditionale: se il valore della comparazione precedente è uguale a 0, il programma salta all'indirizzo di memoria 40102B.
push offset aSuccessInterne	Inserisce sullo stack la stringa "Success: Internet Connection\n".
call sub_40117F	Chiama la funzione sub_40117F, che mostra in output la stringa.
add esp, 4	Aggiunge il valore 4 (per 4 byte) allo stack pointer per restaurarlo al suo stato originale.
mov eax, 1	Assegna valore 1 al registro EAX, significa che l'operazione ha avuto successo.
jump short loc_40103A	Salta all'indirizzo di memoria 40103A.
loc_40102B	Punto di ancoraggio dell'indirizzo di memoria 40102B (a cui si arriva dopo il jump della riga 9).
push offset aError1_1NoInte	Inserisce sullo stack la stringa "Error 1.1: No Internet\n".
call sub_40117F	Chiama la funzione sub_40117F, che mostra in output la stringa.
add esp, 4	Aggiunge il valore 4 (per 4 byte) allo stack pointer per restaurarlo al suo stato originale.

xor eax, eax	Operazione XOR che risetta a 0 il registro EAX.
loc_40103A	Punto di ancoraggio dell'indirizzo di memoria 40103A (a cui si arriva dopo il jump della riga 14).
mov esp, ebp	Copia il valore di EBP in ESP. Questo restaura il valore di ESP precedente all'inizio della funzione.
pop ebp	Toglie dallo stack il registro EBP; insieme all'istruzione precedente rimuove lo stack.
retn	Ritorno dalla funzione sub_401000 .
sub_401000 endp	Chiude la funzione; in questo modo sia lo stack che la funzione chiamata vengono rimossi, avendo terminato il loro compito.

Da quest'analisi possiamo capire che questa il malware verifica lo stato della connessione di rete della macchina su cui viene lanciato. Lo fa tramite la funzione InternetGetConnectedState: a seconda del valore di ritorno dato dalla funzione, il progrmma capisce se c'è connessione (il valore di ritorno è diverso da 0) oppure no (il valore è uguale a 0) e riporta in output il risultato.