PROGETTO S10-L5

SCOPO: rispondere con riferimento a 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 ai seguenti quesiti:

1-) Quali librerie vengono importate dal file eseguibile?

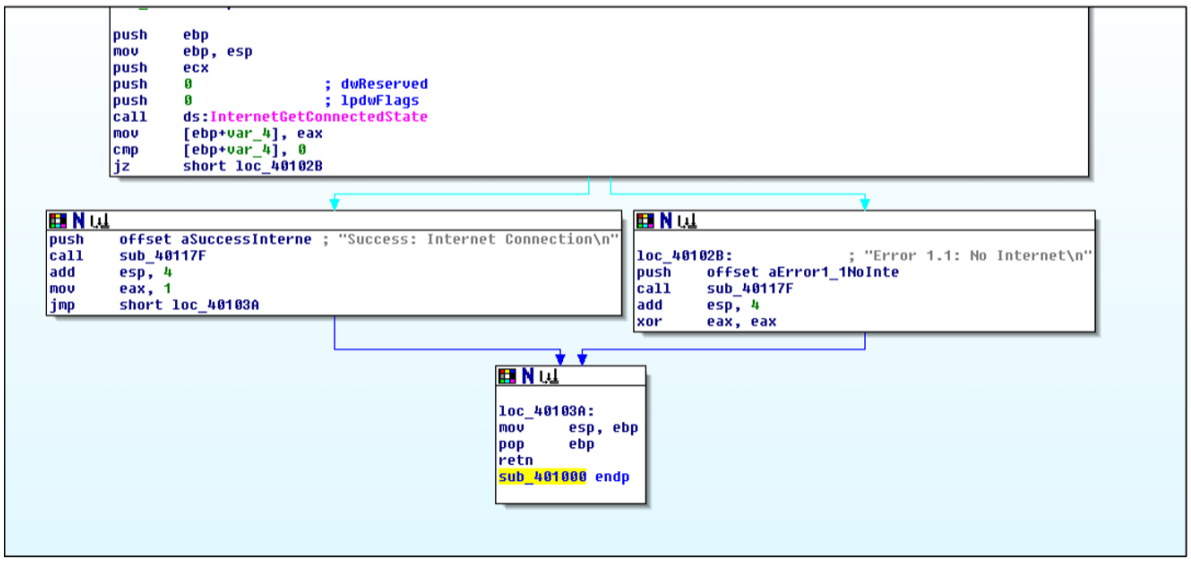
2-) Quali sono le sezioni di cui si compone il file eseguibile del malware?

***con riferimento alla figura sotto:**

3-)Identificare i costrutti noti(creazione dello stack, eventuali cicli, altri costruitti).

4-)Ipotizzare il comportamento della funzionalità implementatà

5-)Bonus fare tabella con significato delle singole righe di codice assembly.

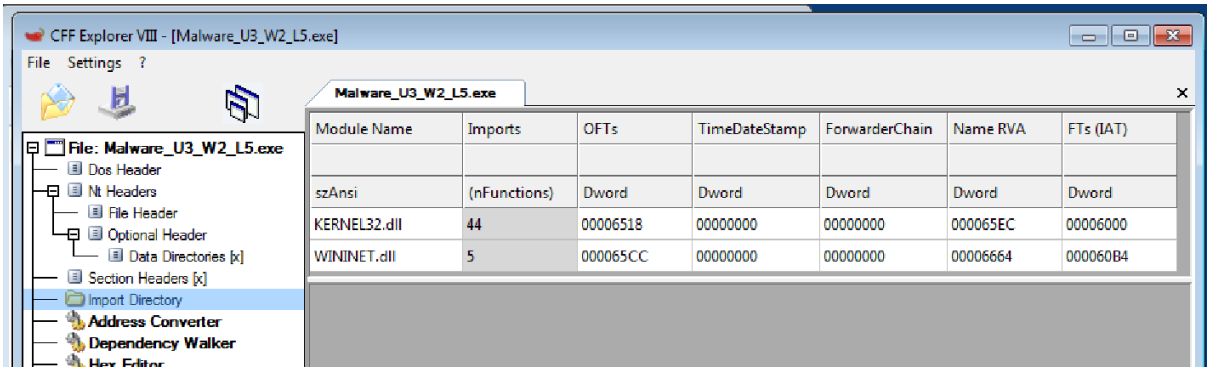


1-) librerie che vengono importate dal file eseguibile

Una libreria : è una raccolta di funzioni, classi o moduli predefiniti e riutilizzabili che possono essere inclusi nei programmi per eseguire determinate operazioni o compiti senza doverli riscrivere da zero.

Come possiamo vedere sull'imagine sotto dopo aver importato le librie con CFF EXPLORER. Otteniamo l'imagine sotto:

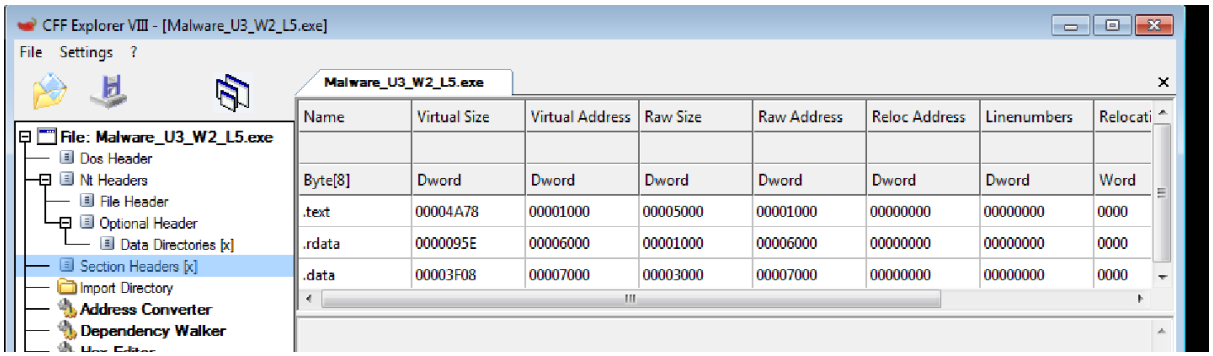
PROGETTO S10-L5



Possiamo vedere che le due librerie importate sono KERNEL32.dll e WININET.dll .
-KERNEL32.dll è una libreria che contiene le funzioni principali per interagire con il sistema operativo.
Come per esmpio maipolazione dei file e gestione della memoria.

-WININET.dll è una libreria che contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP,NTP.

2-) sezioni di cui si compone il file eseguibile del malware



Come possiamo vedere; sull'imagine sopra abbiamo tre sezioni .text , .rdata , .data .

.text : è una sezione che contiene le istruzioni che la CPU eseguirà una volta che il software sarà avviato.

.rdata : include generalmente le informazioni circa le librerie e le funzioni importate ed esportate.

PROGETTO S10-L5

.data : contiene i dati ,le variabili globali del programma eseguibile, che devono essere disponibili da parte del programma.

3-)Identificare i costrutti noti(creazione dello stack, eventuali cicli, altri costruitti).

- come creazione dello stack abbiamo queste righe di codice in assembly

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

Questo perche la creazione dello stack è evidenziato con l'operazione di inserimento **push.**

-come eventuali cicli abbiamo; il ciclo "FOR" che può essere evidenziato come sulla figura sotto.

```
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

I cicli spesso coinvolgono istruzioni di salto condizionale che determinano se continuare o uscire dal ciclo.

-abbiamo anche il ciclo if qua sotto.

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

-come altri costrutti abbiamo la chiamata di funzione.i parametri sono passati sullo stack tramite le istruzioni push.

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

4-)Ipotizzare il comportamento della funzionalità implementatà

In base a alcuni indicazione sulla funzionalità implementatà come per esempio: " **internetGetconnectedState**" ,"**succes internet connexion**", "**no internet**" possiamo ipotizare che questo codice sembra gestire due flussi di esecuzione alternativi per la connessione a internet, uno per il caso

PROGETTO S10-L5

di successo e uno per il caso di errore, e termina pulendo lo stack e restituendo il controllo al chiamante.

5-)Bonus fare tabella con significato delle singole righe di codice assembly.

push ebp: Questa istruzione spinge il valore del registro base della pila (EBP) nello stack.

mov ebp, esp: Questa istruzione sposta il valore corrente dello stack pointer (ESP) nel registro base della pila (EBP).

push ecx: Questa istruzione spinge il contenuto del registro ECX nello stack.

push 0: Questa istruzione spinge il valore 0 nello stack.

push 0: Questa istruzione spinge nuovamente il valore 0 nello stack. Anche questo potrebbe essere un parametro o un valore di inizializzazione.

call ds:intrnetGetConnectedState: Questa istruzione effettua una chiamata di funzione a intrnetGetConnectedState, utilizzando il segmento dati (DS) per recuperare l'indirizzo della funzione. Prima di questa chiamata, sono stati messi nello stack i parametri necessari per la funzione (due valori 0). Dopo la chiamata, il valore di ritorno della funzione viene posto nel registro EAX.

PROGETTO S10-L5

mov [ebp+var_4], eax: Questa istruzione memorizza il valore di ritorno della funzione intrnetGetConnectedState nella variabile locale [ebp+var_4].

cmp [ebp+var_4], 0: Questa istruzione confronta il valore memorizzato nella variabile [ebp+var_4] con 0.

jz short loc_40102B: Questa istruzione esegue un salto condizionale corto (jz, Jump if Zero) alla locazione loc_40102B se il confronto precedente ha dato come risultato zero, cioè se [ebp+var_4] è uguale a zero.

push offset aSuccessinternet: Questa istruzione mette nello stack l'indirizzo (offset) di una stringa denominata "aSuccessinternet". Questa stringa potrebbe essere un messaggio di successo o un identificatore di qualche tipo.

call sub_40117F: Viene effettuata una chiamata alla sotto-routine (subroutine) situata all'indirizzo sub_40117F.

add esp, 4: Questa istruzione aggiunge 4 byte allo stack, probabilmente per ripulire i parametri dalla chiamata di funzione precedente.

mov eax,1: Questa istruzione carica il valore 1 nel registro EAX. Questo valore potrebbe essere utilizzato per indicare un esito positivo o un codice di errore specifico.

jmp short loc_40103A: Questa istruzione esegue un salto incondizionato alla locazione loc_40103A.

PROGETTO S10-L5

loc_401020: Questa etichetta segna l'inizio di un blocco di codice.

push offset aError1_1Nointe: Questa istruzione mette nello stack l'indirizzo (offset) di una stringa denominata "aError1_1Nointe".

call sub_40117F: Viene effettuata una chiamata alla sotto-routine situata all'indirizzo sub_40117F.

add esp, 4: Come prima, questa istruzione ripulisce i parametri dalla chiamata di funzione precedente.

xor eax, eax: Questa istruzione esegue un'operazione di XOR tra il registro EAX e se stesso, effettivamente reimpostando il registro EAX a zero.

loc_40103A: Questa etichetta segna un punto comune di uscita per il flusso di esecuzione.

mov esp, ebp: Questa istruzione ripristina il valore dello stack pointer (ESP) con il valore del registro base della pila (EBP), ripristinando così lo stack al suo stato precedente alla funzione corrente.

pop ebp: Questa istruzione ripristina il valore originale del registro base della pila (EBP), rimuovendolo dalla cima dello stack.

PROGETTO S10-L5

retn: Questa istruzione termina la funzione e restituisce il controllo al chiamante.