Marzo 2024

CYBER SECURITY SPECIALIST

S10L5

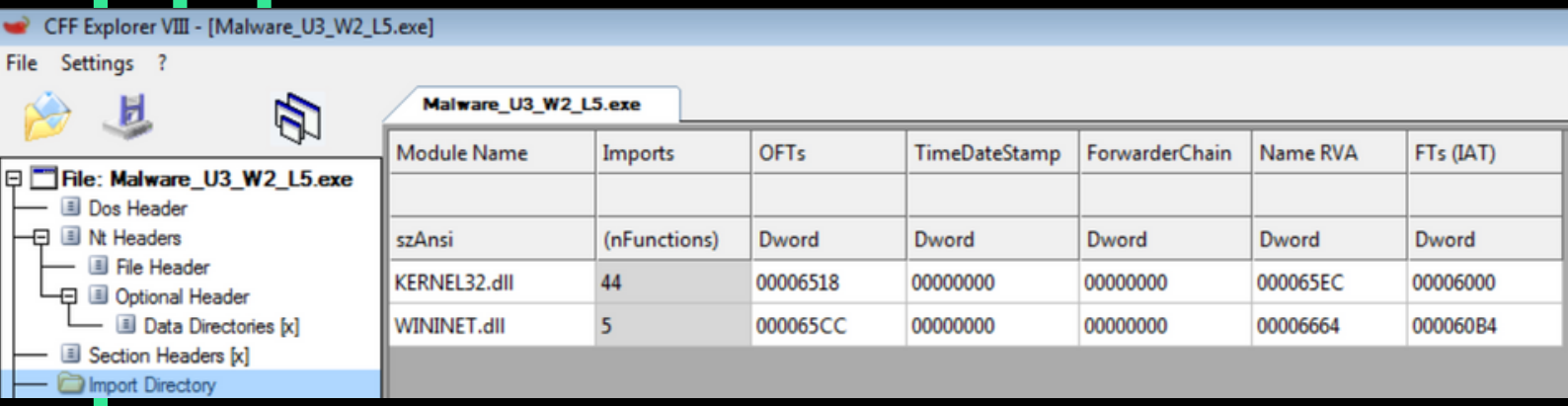
Jacopo Trovato

Malware Analysis

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 a pagina 4, rispondere 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. Per sapere le librerie[1] che vengono importate dal file eseguibile, è necessario usare il tool CFF Explorer[2],importante il malware che si desidera analizzare. Andare sulla sezione "**Import Directory**" e le librerie che vengono importate in questo caso sono:

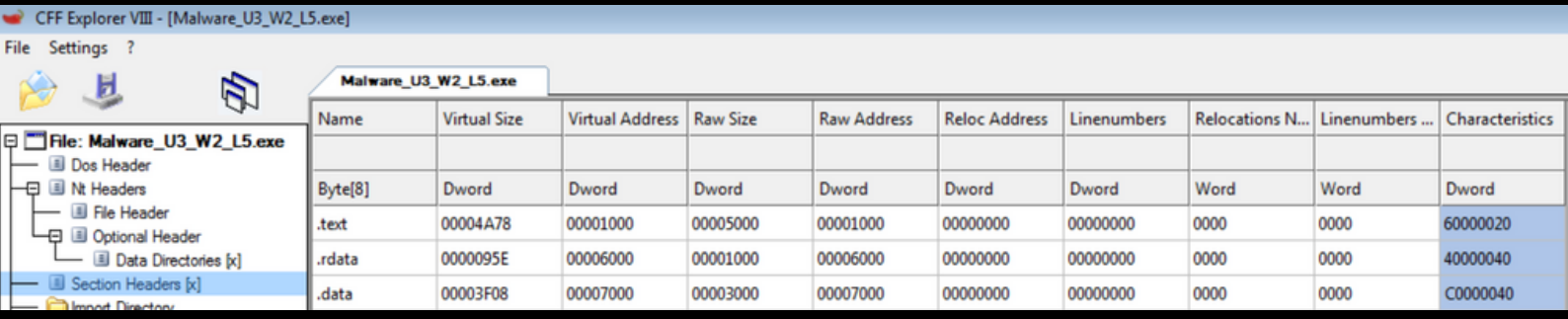
- KERNELL32.dll = Contiene funzioni essenziali per la gestione della memoria. Le operazioni di input/output e varie altre attività di sistema, come la gestione della memoria, le informazioni di sistema e le operazioni su file.
- WININET.dll = Fornisce funzionalità per la comunicazione via Internet e la gestione delle risorse di rete. Alcune funzionalità sono, Navigazione Web, Download e Upload.



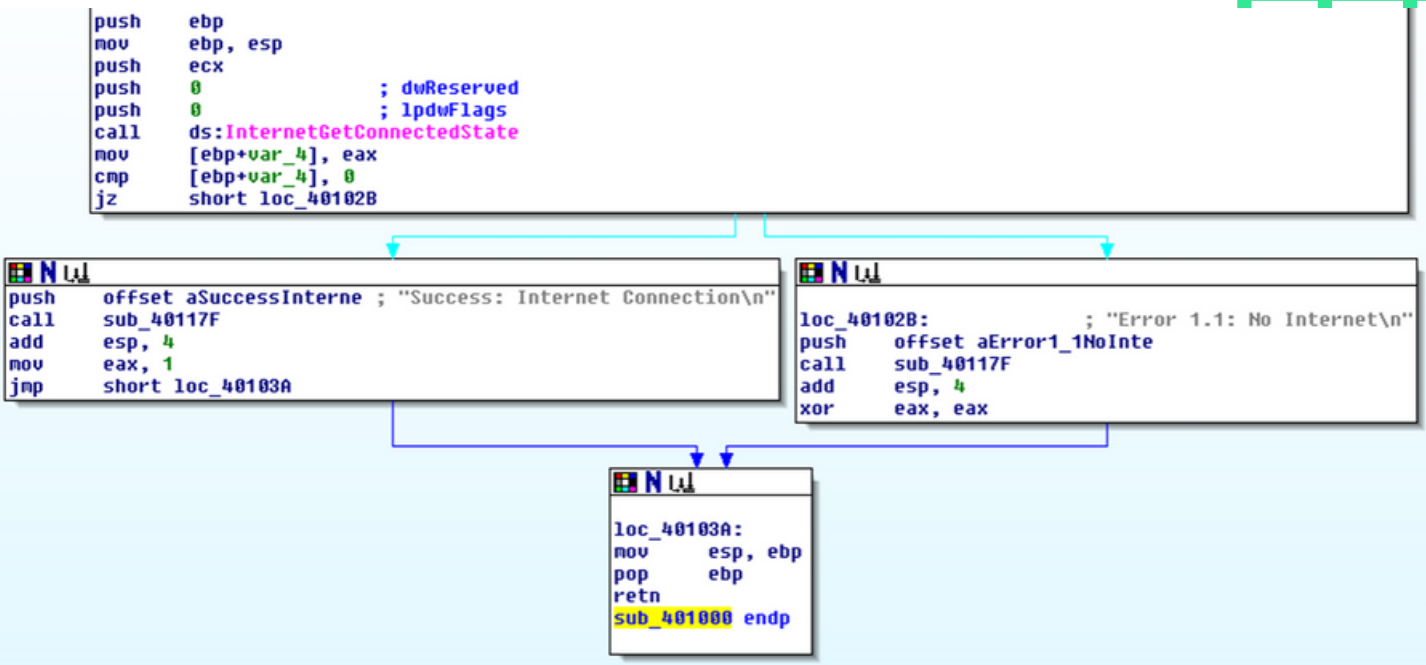
Pagina di Import Directory di CFF Directory
con le relative librerie.

2. Per conoscere le sezioni[3] di cui si compone il file, andare, sempre su CFF explorer, nella pagina di "**Section Headers**", le sezioni in questo caso sono:

- .text = Contiene dati di sola lettura, come stringhe, costanti e altre informazioni di sola lettura utilizzate dal programma durante l'esecuzione, questi dati sono statici e non possono essere modificati durante l'esecuzione del programma.
- .rdata = Contiene dati di sola lettura, come stringhe e costanti, utilizzati dal programma durante l'esecuzione, questi dati sono statici e non possono essere modificati.
- .data = Contiene dati modificabili utilizzati dal programma durante l'esecuzione, questi dati includono variabili globali, dati inizializzati e altri dati dinamici e possono essere modificati durante l'esecuzione del programma.



Pagina di Section Headers di CFF Directory
con le relative sezioni.



3. I costrutti noti sono:

push ebp mov ebp, esp	**Viene creato lo stack**[4]
push ecx push 0 ; dwReserved push 0 ; lpdwFlags call ds : InternetGetConnectedState	**Viene chiamata la funzione**[5] **"InternetGetConnectedState"**
cmpi [ebp+var_4], 0 jz short loc_40102B	**IF, una comparazione tra due valori, utilizza il Jz ed effettua un salto se produce un risultato nullo.**
loc_400103A: mov esp, ebp pop esp	**Viene chiuso lo stack**

4. Il codice è una funzione scritta in linguaggio assembly x86[6] per gestire lo stato della connessione Internet. Inizia impostando il frame del registro base "ebp" e lo stack pointer "esp". Successivamente, prepara gli argomenti per la chiamata alla funzione "InternetGetConnectedState", mettendo dei valori sullo stack. Dopo aver ottenuto il risultato della chiamata, controlla se la connessione è attiva confrontando il risultato con 0. Se la connessione è attiva, il programma stampa un messaggio tramite la funzione "sub_40105F", altrimenti passa alla gestione degli errori, chiamando la funzione "sub_40117F". Infine, ripristina lo stack pointer e il registro base, finendo la funzione.

ISTRUZIONE	SPIEGAZIONE
push ebp	mette il valore del registro base “ebp” nello stack
mov ebp, esp	imposta il registro base ”ebp” con il valore dello stack pointer “esp”
push ecx	Memorizza il valore attuale nello stack
push 0	mette il valore 0 nello stack
call ds: InternetGetConnectedState	chiama la funzione “GetConnectedState”dall'indirizzo di memoria “ds”
mov [ebp+var_4], eax	assegna il valore di “eax” alla variabile “var_4”
cmp [ebp+var_4], 0	confronta il valore della variabile “var_4” con 0
jz Short loc_40102B	salta fino a “loc_40102B” se il confronto precedente ha prodotto 0
push offset aSuccessInterne	mette l'indirizzo “aSuccessInterne” nello stack
call sub_404117F	chiama la funzione “sub_404117F”
add esp, 4	incrementa il registro “esp” di 4
mov eax, 1	carica il valore 1 in “eax”
jmp Short loc_40103A	salta fino a “loc_40103A”
push offset aError1_1NoInte	mette l'indirizzo “aError1_1NoInte” nello stack
call sub_40117F	chiama la funzione “sub_40117F”
add esp, 4	incrementa il registro “esp” di 4
xor eax, eax	azzera il registro “eax” utilizzando “xor”
mov esp,ebp	copia il valore di “ebp” in “esp” rispristinando quest’ultimo
pop ebp	estrae il valore superiore dello stack e lo copia in“ebp” ripristinandolo
retn	ritorna il controllo al chiamante

Glossario

1. **Libreria** = insieme di codice predefinito che contiene funzioni e routine comuni, scritte in linguaggio assembly. Queste funzioni possono essere utilizzate da altri programmi assembly per eseguire operazioni specifiche, come la manipolazione delle stringhe. Le librerie semplificano lo sviluppo del software, consentendo ai programmatori di evitare di dover scrivere da zero le stesse operazioni di base ogni volta che sviluppano un nuovo programma.

2. **CFF Explorer** = software di reverse engineering specializzato nell'analisi dei file eseguibili (PE) Windows. Questo strumento consente agli utenti di esaminare gli attributi dei file PE, visualizzare le dipendenze, disassemblare il codice, e accedere ai dati binari e alle risorse dei programmi.

3. **Sezione** = porzione del codice sorgente che organizza i dati e le istruzioni del programma. Le sezioni permettono una chiara organizzazione del codice e dei dati, facilitando la comprensione e la gestione del programma.

4. **Stack** = regione di memoria utilizzata per gestire le chiamate a funzioni e conservare dati locali. Si usa per passare dati tra le funzioni e mantenere traccia delle istruzioni di ritorno.

6. **Assembly x86** = linguaggio di programmazione a basso livello utilizzato per scrivere codice direttamente comprensibile dai processori x86, si riferisce a una famiglia di architetture di processori a 32-bit e 64-bit, questo linguaggio permette di manipolare direttamente i registri, la memoria e le istruzioni del processore, consentendo di ottenere un controllo preciso sulle operazioni del computer.