PROGETTO SETTIMANA 10

Parte 1

Dato un malware individuare:

1) Quali sono le librerie che vengono importate dal file eseguibile?
2) Quali sono le sezioni di cui si compone il file eseguibile del malware?

Quesito 1 – Librerie

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

Utilizzando il programma **CFF Explorer** ci dirigiamo nella sezione **import directory** possiamo vedere quali librerie vengono importate. In questo caso vengono importate:

- **KERNEL32.DLL**: contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria.
- **WININET.DLL**: contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP.

Quesito 2 – Sezioni

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

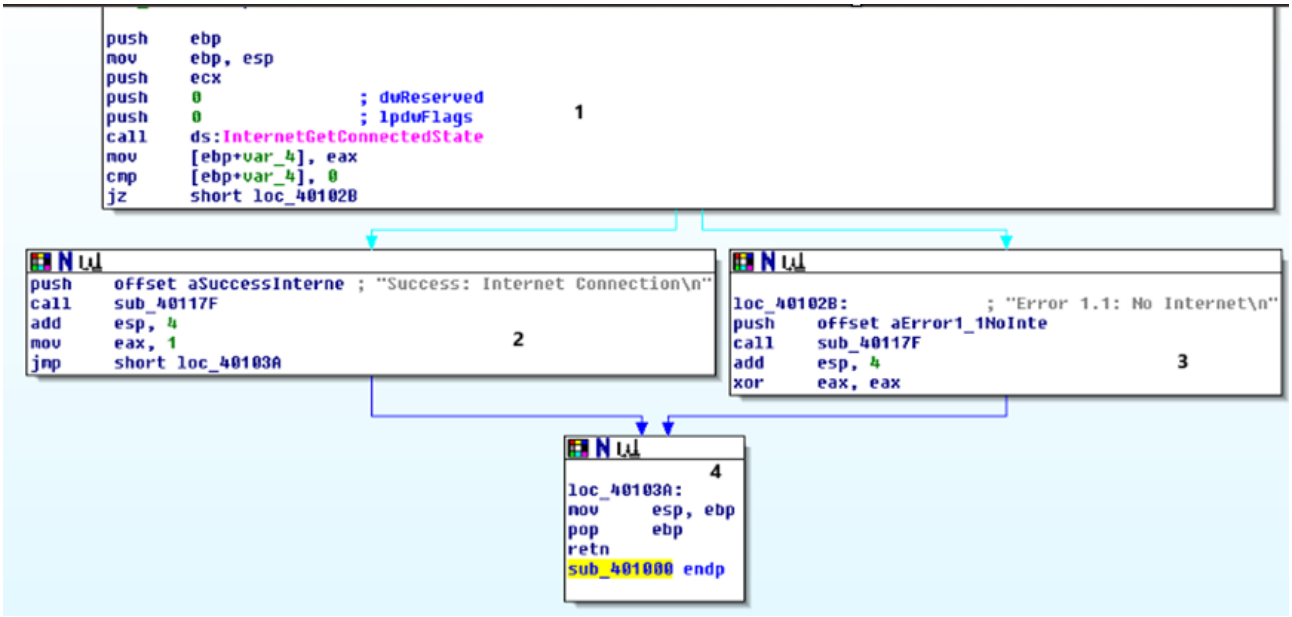
Sempre utilizzando **CFF Explorer** ci spostiamo sella parte **section headers** in modo da vedere le sezioni di cui si compone il file. Questo eseguibile è formato dalle sezioni:

- **.text**: contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
- **.rdata**: include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile.
- **.data**: contiene tipicamente i dati / le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma.

Parte 2

Dato il diagramma di flusso seguente, rispondere ai quesiti:

3) Identificare i costrutti noti.
4) Ipotizzare il comportamento della funzionalità implementata.
5) BONUS -> Spiegare ogni riga del codice.



Quesito 3 – Costrutti

In questo codice assembly possiamo identificare due costrutti:

- **Creazione di uno stack**:
  push: Utilizzato per mettere valori nello stack.
  pop: Utilizzato per estrarre valori dallo stack.
  mov: Utilizzato per spostare dati da una posizione all'altra.

- **Ciclo if**:
  cmp: Compara due operandi.
  jz: Salta all'etichetta specificata se l'ultimo confronto ha prodotto zero (il che significa che i due operandi erano uguali).
  jmp: Salta incondizionatamente all'etichetta specificata.

Quesito 4 – Comportamento funzionalità

Osservando il diagramma e il codice si può dedurre che:

questo codice effettua un controllo della connessione Internet tramite la funzione '**InternetGetConnectedState' .** Il controllo che effettua questa funzione restituisce un valore diverso da zero che indicherà che la connessione a internet è attiva.

In base all'esito della risposta il programma si comporterà in maniera diversa:

- Se la connessione internet è attiva verrà stampato "Success: Internet Connection".
- Se la connessione internet non è attiva verrà stampato "Error 1.1: No Internet".

Dopo aver stampato il messaggio, il programma imposta il registro **eax** su 1 se la connessione è attiva, 0 se non c'è connessione.

Quesito 5 BONUS – Spiegazione

Andando più nel dettaglio esaminiamo cosa fa ogni riga del codice:

Primo blocco di codice:

1) **push epb**: Salva il valore del registro base dello stack (**EBP)** nello stack.
2) **mov epb, esp**: Imposta il registro base dello stack (**EBP**) uguale al registro stack pointer (**ESP**) per creare un frame di stack per una funzione.
3) **push ecx**: Salva il valore del registro ECX nello stack.
4) **push 0**: Mette il valore 0 nello stack.
5) **push 0**: Mette il valore 0 nello stack.
6) **call ds:InternetGetConnectedState**: Chiama la funzione InternetGetConnectedState per controllare lo stato della connessione a Internet. Il risultato viene memorizzato in [**epb+var_4**].
7) **cmp [epb+var_4], 0**: Compara il valore memorizzato in [**epb+var_4**] con **0**.
8) **jz short loc_40102B**: Salta a **loc_40102B** se il risultato della comparazione è uguale a zero (cioè, se non c'è connessione a Internet).

Secondo blocco di codice:

Se la connessione a Internet è attiva il programma esegue le seguenti operazioni:

1) Chiama una funzione (**sub_40117F**) per stampare un messaggio "Success: Internet Connection".
2) **mov eax, 1**: Carica il valore 1 nel registro eax.
3) Salta a **loc40103A.**

Terzo blocco di codice:

Se la connessione a Internet non è attiva, il programma esegue le seguenti operazioni:

1) Chiama la stessa funzione **(sub_40117F**) per stampare un messaggio di errore "Error 1.1: No Internet".
2) **xor eax, eax**: Assegna 0 al registro eax (questo probabilmente indica un errore).
3) Salta a **loc40103A**.

Quarto blocco d codice:

1) **loc40103A**: Ripristina lo stato dello stack e ritorna dalla funzione.

- **mov esp, epb**: Ripristina il registro stack pointer (ESP) al valore del registro base dello stack (EBP).
- **pop epb**: Esegue il pop del valore dallo stack e lo memorizza nel registro base dello stack (EBP).
- **retn**: Questa istruzione termina la subroutine e ritorna al punto di chiamata.