PROGETTO 29 MARZO S10L5

L'obiettivo di oggi, 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, e quello di 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 Fig.4, 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

1) Librerie importante dal file eseguibile

Controllare quali sono le librerie e funzioni richieste dall'eseguibile contenute nell'header PE è fondamentale per capire lo scopo del malware. Oltre alle funzioni importate, il file eseguibile può mettere a disposizione di altri programmi o dell'utente delle funzioni da "chiamare": l'header del formato PE contiene un elenco delle funzioni esportate da un eseguibile. Per controllare le funzioni importate ed esportate dal malware, nello specifico del file eseguibile **Malware_U3_W2_L5.exe**; ci serviremo di un tool di Windows: **CFF Explorer**. In Fig. 2 possiamo vedere le Librerie importate dal file eseguibile.

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

Fig.2

Andiamo ad analizzare ogni libreria:

KERNEL32.DLL: Questa libreria è una componente critica del sistema operativo Windows, che fornisce un'interfaccia tra il software in esecuzione e il nucleo del sistema operativo (il kernel). Gestisce molteplici funzionalità fondamentali relative alla gestione della memoria, alle operazioni di input/output, alla creazione e gestione dei processi e dei thread, alle funzioni di sincronizzazione, e alla manipolazione di file e dispositivi

WININET.dll: Questa libreria fornisce un'interfaccia di programmazione delle applicazioni (API) ad alto livello per la gestione delle comunicazioni Internet. Offre funzionalità che permettono alle applicazioni di interagire con il protocollo HTTP, FTP (e altre funzionalità). E' essenziale quindi per tutte quelle applicazioni Windows che richiedono funzionalità di rete

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

Sempre grazie a CFF Explorer, possiamo anche vedere le sezioni con cui si compone il file eseguibile del malware (Fig.3)

Malware_U3_W2_L5.exe

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

Fig.3

Andiamo ad analizzare le sezioni:

.text: Nel sistema operativo Windows XP, così come in altri sistemi operativi Windows, la sezione .text di un file eseguibile svolge un ruolo fondamentale. La sua funzione principale è contenere il codice macchina del programma, ossia le istruzioni che il processore esegue direttamente. Tra le sue funzioni principali abbiamo l'esecuzione del codice, la protezione della memoria, l'ottimizzazione delle prestazioni ed il caricamento della memoria.

.rdata: Questa sezione dati di sola lettura svolge un ruolo importante all'interno del sistema operativo Windows. Contiene dati costanti (read only) usati dal programma, che non devono essere modificati durante l'esecuzione. Alcuni aspetti chiave di questa sezione sono la conservazioni di costanti e di dati immutabili, le import table, il supporto per il debugging, l'ottimizzazione delle prestazioni e la conservazione dei puntatori a funzioni.

.data: Questa sezione del file eseguibile contiene i dati inizializzati globali e statici del programma, i quali possono essere modificati durante l'esecuzione del programma. Alcuni aspetti chiave di questa sezione sono la conservazione dei dati inizializzati, i permessi di lettura e scrittura, le variabili globali e statiche conservate dal programma, la gestione della memoria e il dimensionamento dinamico delle variabili.

Con riferimento alla figura in Fig.4 , risponde ai seguenti quesiti:

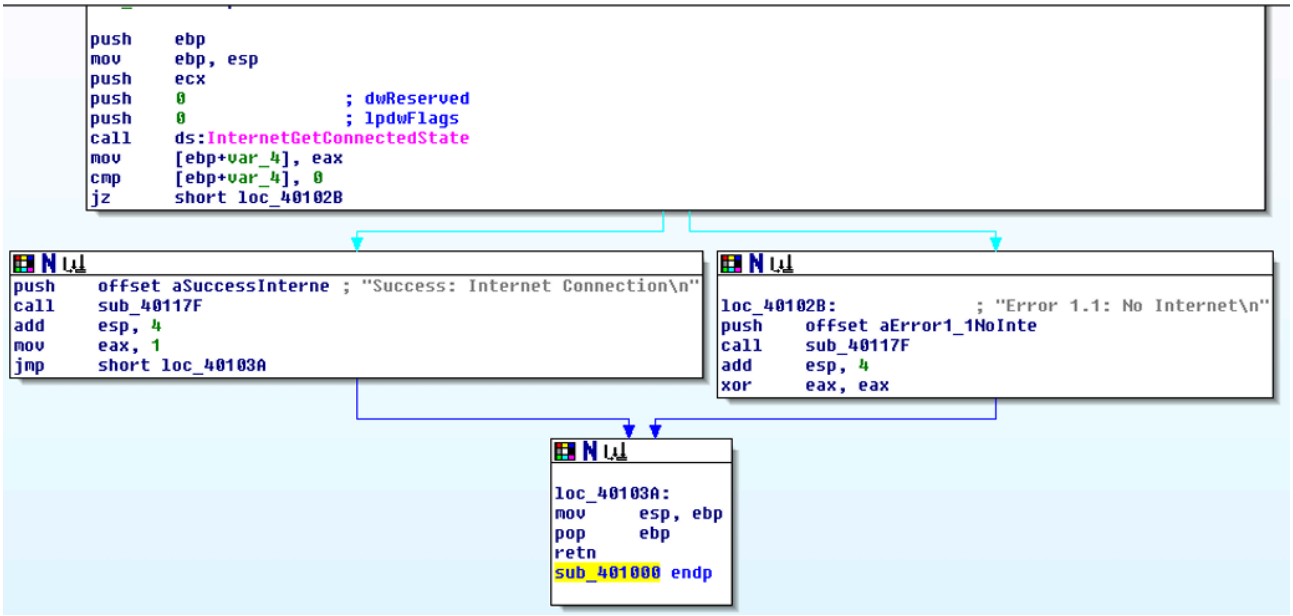

Fig.4

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

Ho identificato 3 costrutti principali:

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

Creazione dello Stack: Queste due istruzioni insieme stabiliscono un nuovo frame dello stack per la funzione che sta per eseguire il suo codice. Il frame dello stack è importante perché fornisce un'area isolata nella stack memory dove la funzione può salvare i dati temporanei senza influenzare altre parti del programma.

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

Gestione delle condizione, costrutto condizionale IF: Le istruzioni cmp e jz sono usate per verificare il risultato di '**internetGetConnectedState'** e decidere il flusso del programma in base al risultato del cmp. Se i due valori confrontati sono uguali (risultato = 0), il programma salta alla loc_40102B, in caso contrario continua con l'istruzione push offset aSuccessInterne

```
mov     esp, ebp
pop     ebp
```

Rimozione dello stack: Viene ripristinato il valore del registro Stack Pointer al valore del base pointer, successivamente viene rimosso il valore corrente dalla cima dello stack e viene salvato nel registro EPB. Poiché il valore di EPB era stato salvato nello stack all'inizio della funzione, questo ripristina EPB al suo stato precedente prima che la funzione fosse chiamata

4) Ipotizzare il comportamento della funzionalità implementata

Questo frammento di codice controlla lo stato della connessione a internet della macchina attraverso la funzione Windows API **InternetGetConnectedState.** Attraverso la gestione del flusso (costrutto IF) viene controllato se il parametro restituito dalla funzione sopracitata è uguale a 0

- Se il parametro ritorno è diverso da 0, la funzione scrive a schermo "Success: Internet Connection", quindi c'è connessione internet (Fig.5)

```
loc_40102B:                             ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
add     esp, 4
xor     eax, eax
```

Fig.5

- Se il parametro di ritorno è uguale a 0, la funzione scrive a schermo "No Internet", quindi non c'è connessione a Internet. (Fig.6)

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

Fig.6

5) Fare una tabella con significato delle singole righe di codice assembly

ISTRUZIONE	DESCRIZIONE
push ebp	Salva il valore corrente di Base Pointer (ebp) sullo stack
mov ebp, esp	Imposta il Base Pointer (ebp) al valore attuale dello Stack pointer (esp)
push ecx	Salva il valore corrente del registro ecx sullo stack
push 0	Pusha 0 nello stack per il parametro dwReserved della funzione InternetGetConnectedState
push 0	Pusha 0 nello stack per il parametro lpdwFlags della funzione InternetGetConnectedState
call ds:InternetGetConnectedState	Chiama la funzione InternetGetConnectedState, i parametri sono presi dallo stack
mov [ebp+var_4], eax	Sposta il valore di ritorno della funzione (eax) in una variabile locale alla funzione (ebp+var_4)
cmp [ebp+var_4], 0	Compara il valore contenuto nella variabile locale con 0
jz short loc_40102B	Se il valore è zero (cioè se cmp ha stabilito che [ebp+var_4] è uguale a 0), salta all'indirizzo loc_40102B
push offset aSuccessInterne	Mette sullo stack l'indirizzo della stringa "Success: Internet Connection\n”
call sub_40117F	Chiama la subroutine all'indirizzo sub_40117F
add esp, 4	Pulisce lo stack incrementando di 4 il valore dello Stack Pointer (ESP), rimuovendo così l'argomento precedentemente inserito sullo stack (standard calling convention)

ISTRUZIONE	DESCRIZIONE
mov eax, 1	Sposta il valore 1 nel registro EAX
jmp short loc_40103A	Esegue un salto incondizionato a un'altra posizione nel codice (loc_40103A), per continuare l'esecuzione da quel punto
loc_40102B ; "Error 1.1: No Internet\n”	Identifica una posizione specifica nel codice, il commento suggerisce che non c’è internet
push offset aError1_1NoInte	Mette sullo stack l'indirizzo della stringa di errore "Error 1.1: No Internet\n”
call sub_40117F	Chiama la subroutine all'indirizzo sub_40117F
add esp, 4	Aggiusta lo stack pointer (ESP) dopo la chiamata alla funzione, rimuovendo il parametro che era stato passato
xor eax, eax	Azzera il registro EAX impostando il suo valore a 0
loc_40103A	identifica una posizione specifica nel codice.
mov esp, ebp	Ripristina il valore dello Stack Pointer (ESP) al valore di Base Pointer (EBP), che era stato impostato all'inizio della funzione.
pop ebp	Recupera il valore originale del Base Pointer (EBP) dallo stack.
retn	Ritorna al chiamante, facendo il pop dell'indirizzo di ritorno dallo stack e saltando a quel punto nel codice.
sub_401000 endp	Indica la fine della subroutine con l'etichetta 'sub_401000'.