EPICODE

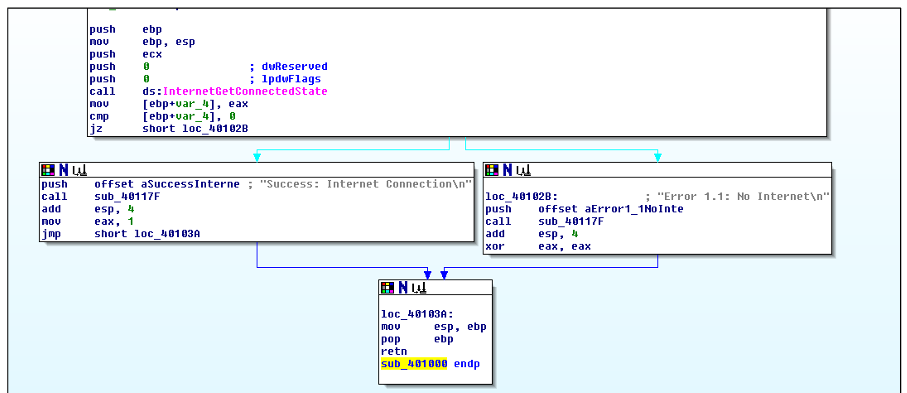
Progetto S10 – L5

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 sottostante, 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



Segue soluzione

SOLUZIONE

1. Quali librerie vengono importate dal file eseguibile?

Come è possibile notare dalla figura sottostante, le librerie importate dal malware sono:  
- KERNEL32.dll  
- WININET.dll

KERNEL32.dll è la libreria che fa parte del sistema Windows. Contiene le funzioni principali per interagire con il sistema operativo, come per esempio la gestione della memoria, la manipolazione dei file.

WININET.dll contiene le funzioni per l’implementazione di alcuni protocolli di rete. Come HTTP, FTP, NTP.

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

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

Le sezioni importate dal malware sono:  
- .text  
- .rdata  
- .data

.text è la sezione che contiene il codice (le istruzioni) che la cpu eseguirà una volta avviato il malware.

.rdata è la sezione che, generalmente, contiene le informazioni riguardanti le importazioni e le esportazioni delle librerie e delle funzioni.

.data è la sezione che, tipicamente, contiene le variabili globali. Ovvero quelle variabili non definite all’interno di un contesto di una funzione, ma dichiarate globalmente e accessibili da qualsiasi funzione.

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

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

Si identificano tre costrutti noti:

1. Creazione di uno stack: In questa situazione lo stack viene creato con “push ecx”. Serve a creare spazio per una variabile locale senza preoccuparsi del valore di ecx. Per esempio, il compilatore Visual C++ usa spesso “push ecx” e “pop ecx” come alternative a “sub esp, 4” e “add esp, 4”.

2. Si tratta di un “if”, in quanto in base al risultato della funzione salterà, o meno, ad una locazione diversa.

3. Rimozione dello stack: Questo costrutto serve ad eliminare il frame della funzione corrente dallo stack ed a ritornare alla funzione chiamante.

Immagine che contiene testo, schermata, schermo, linea

Descrizione generata automaticamente

4. Ipotizzare il comportamento della funzionalità implementata

Il comportamento è così riassumibile:

- Crea lo stack.

- Dichiara i 2 parametri necessari e chiama la funzione per verificare la connessione ad Internet.

- Se c’è connessione stampa la conferma di connessione, altrimenti stampa l’errore di connessione.

- Rimuove lo stack.

5. BONUS fare tabella con significato delle singole righe di codice assembly

Prima parte del codice

push ebp:

Salva il valore di ebp nella pila. ebp è il registro che indica il puntatore di base della funzione.

mov ebp, esp:

Copia il valore di esp in ebp. esp è il registro che indica il puntatore di pila, cioè la posizione “in alto” in cui inserire o estrarre i dati dalla pila.

push ecx :

Salva il valore di ecx nella pila e crea spazio per una variabile locale.

push 0 ; dwReserved:

Passa 0 come secondo parametro alla funzione InternetGetConnectedState. Questo parametro è riservato e deve essere sempre 0.

push 0 ; lpdwFlags:

Passa 0 come primo parametro alla funzione InternetGetConnectedState. Questo parametro è un puntatore a una variabile che riceve il tipo di connessione a Internet.

call ds:InternetGetConnectedState:

Chiama la funzione InternetGetConnectedState, che verifica lo stato della connessione a Internet del sistema locale e restituisce un valore booleano (vero o falso) in eax. eax è il registro che viene usato spesso per memorizzare il valore di ritorno di una funzione.

mov [ebp+var\_4], eax:

Salva il valore restituito dalla funzione (vero 1 o falso 0) in una variabile locale.

cmp [ebp+var\_4], 0:

Confronta il valore della variabile locale con 0.

jz short loc\_40102B:

Salta a loc\_40102B se il valore della variabile locale è 0, cioè se non c'è connessione a Internet. Questo salto dipende dal valore di ZF (zero flag), che viene impostato a 1 se il confronto precedente ha dato come risultato 0 (in quanto 0 = 0).

Ipotesi 1: c’è connessione

push offset aSuccessInterne ; "Success: Internet Connection\n":

Passa l'indirizzo di una stringa "Success: Internet Connection\n" come parametro alla funzione sub\_40117F.

call sub\_40117F:

Chiama la funzione sub\_40117F, che è una funzione definita dall'utente e non fa parte delle funzioni standard di assembly. Per sapere con certezza cosa fa, bisognerebbe analizzare il codice della funzione. Tuttavia, è molto probabile che la funzione serva a stampare la stringa passata.

add esp, 4:

Ripristina il puntatore di pila dopo aver passato il parametro, sommando 4 byte al valore di esp. Questo serve per mantenere la coerenza della pila e per evitare di sovrascrivere i dati salvati in precedenza.

mov eax, 1:

Assegna 1 al registro eax come valore di ritorno della funzione corrente (connessione stabilita).

jmp short loc\_40103A:

Salta incondizionatamente a loc\_40103A, che potrebbe essere la fine della funzione corrente o il punto in cui il malware esegue altre operazioni.

Ipotesi 2: non c’è connessione

loc\_40102B ; "Error 1.1: No Internet\n":

Etichetta della locazione loc\_40102B, lascia un commento.

push offset aError1\_1NoInte:

Passa l’indirizzo della stringa "Error 1.1: No Internet\n" alla funzione che segue sub\_40117F.

call sub\_40117F:

Chiama la funzione sub\_40117F, che è una funzione definita dall'utente e non fa parte delle funzioni standard di assembly. Per sapere con certezza cosa fa, bisognerebbe analizzare il codice della funzione. Tuttavia, è molto probabile che la funzione serva a stampare la stringa passata.

add esp, 4:

Ripristina il puntatore di pila dopo aver passato il parametro, sommando 4 byte al valore di esp. Questo serve per mantenere la coerenza della pila e per evitare di sovrascrivere i dati salvati in precedenza.

xor eax, eax:

Azzera il registro eax.

Parte finale del codice

loc\_40103A:

Etichetta della locazione loc\_40103A

mov esp, ebp:

Copia il valore del registro ebp nel registro esp, per ripristinare lo stack pointer.

pop ebp:

Recupera il valore del registro ebp dallo stack, per ripristinare il base pointer.

retn:

Recupera l’indirizzo di ritorno dallo stack e salta a quella locazione, riprendendo l’esecuzione del programma principale.

sub\_401000 endp:

Indica la fine della funzione sub\_401000.