S10 L4

Report esercizio “Malware Analysis”

GiuliaSalani

TRACCIA

La figura seguente mostra un estratto del codice di un malware.
Identificare i costrutti noti visti durante la lezione teorica.

```
push ebp |
mov ebp, esp
push ecx
push 0                          ; dwReserved
push 0                          ; lpdwFlags
call ds:InternetGetConnectedState
mov [ebp+var_4], eax
cmp [ebp+var_4], 0
jz short loc_40102B
push offset aSuccessInterne ; “Success: Internet Connection\n”
call sub_40105F
add esp, 4
mov eax, 1
jmp short loc_40103A
```

Opzionale: Provate ad ipotizzare che funzionalità è implementata nel codice assembly.
Hint: La funzione internetgetconnectedstate prende in input 3 parametri e permette di controllare se una macchina ha accesso ad internet.

INTRODUZIONE

L'assembly è un linguaggio di basso livello, legato strettamente all'architettura del processore, composto da istruzioni mnemoniche comprensibili dalla CPU. Contrariamente ai linguaggi di alto livello, è più vicino al linguaggio macchina e offre un controllo diretto sull'hardware.
Nel contesto della malware analysis, l'assembly è cruciale. Il codice sorgente di malware spesso non è disponibile, e il reverse engineering coinvolge l'analisi del codice binario. Gli analisti di malware utilizzano l'assembly per disassemblare e comprendere il comportamento del malware. Questo coinvolge l'identificazione delle funzioni, la comprensione dei flussi di controllo e la rivelazione delle tecniche evasive utilizzate dal malware per sfuggire alla rilevazione.
L'assembly è inoltre fondamentale per capire le vulnerabilità sfruttate dai malware e sviluppare contromisure di sicurezza. Conoscere approfonditamente assembly consente agli analisti di malware di individuare e comprendere le tattiche di attacco, aiutando nella protezione e nella mitigazione dei rischi associati alle minacce informatiche.

FUNZIONALITÀ DEL CODICE

Prologo della funzione (creazione dello stack):

```
push ebp |
mov ebp, esp
push ecx
```

Inserimento nello stack dei parametri richiesti dalla funzione e successiva chiamata della funzione:

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

salvataggio del risultato nella variabile [ebp+var_4]:

```
mov [ebp+var_4], eax
```

confronto :

```
cmp [ebp+var_4], 0
```

salto condizionale se la condizione è vera:

```
jz short loc_40102B
```

se la condizione è falsa il programma procede con il blocco successivo:

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

chiamata:

```
call sub_40105F
```

aggiunta di 4 byte ad esp per rimuovere il parametro dalla pila:
add esp, 4

spostamento del valore 1 in eax:
mov eax, 1

salto incondizionato:
jmp short loc_40103A

OPZIONALE

Al centro del codice vi è la funzione InternetGetConnectedState che, come abbiamo letto nella traccia, permette di controllare se una macchina ha accesso a internet.

Il codice verifica infatti lo stato della connessione a Internet attraverso la chiamata a InternetGetConnectedState. Se la connessione non è attiva (il risultato è zero), il programma salta a una parte di codice non condivisa (loc_40102B) che probabilmente gestisce il caso in cui la connessione non è disponibile. Se la connessione ha successo (il risultato è diverso da zero), il programma procede eseguendo la stampa di un messaggio che conferma la connessione a Internet. Successivamente, chiama un'altra funzione (non specificata nel codice fornito), ripulisce lo stack e salta a un'altra parte del codice (loc_40103A).

Il cuore della parte di codice condivisa nella traccia è una struttura if-else, che dà istruzioni al programma su come procedere nel caso in cui la connessione sia attiva e nel caso in cui non lo sia.