



S10/L4

Tecniche di ingegneria inversa

TRACCIA

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

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 a internet.

Consegna: 1. Identificare i costrutti noti (e s. while, for, if, switch, ecc.)

2. Ipotizzare la funzionalità – esecuzione ad alto livello.

BONUS: studiare e spiegare ogni singola riga di codice.



TRACCIA

```
♦ .text:00401000      push    ebp
♦ .text:00401001      mov     ebp, esp
♦ .text:00401003      push    ecx
♦ .text:00401004      push    0          ; dwReserved
♦ .text:00401006      push    0          ; lpdwFlags
♦ .text:00401008      call    ds:InternetGetConnectedState
♦ .text:0040100E      mov     [ebp+var_4], eax
♦ .text:00401011      cmp     [ebp+var_4], 0
♦ .text:00401015      jz      short loc_40102B
♦ .text:00401017      push    offset aSuccessInterne ; "Success: Internet Connection\n"
♦ .text:0040101C      call    sub_40105F
♦ .text:00401021      add     esp, 4
♦ .text:00401024      mov     eax, 1
♦ .text:00401029      jmp     short loc_40103A
♦ .text:0040102B      ; -----
♦ .text:0040102B
```

Ingegneria inversa

Come abbiamo visto in lezione teorica stamattina, l'**ingegneria inversa** è una competenza fondamentale per comprendere **come funziona un malware**. Non ci concentriamo sulla singola riga di codice, ma sull'insieme delle istruzioni che compongono le funzionalità. Questo processo ci permette di **ricostruire** ad alto livello il comportamento del malware, **riconoscendo costrutti comuni** come if, for e while. L'obiettivo è capire le intenzioni del creatore del malware e come queste si manifestano nel codice. In pratica, analizziamo il codice assembly per vedere come le istruzioni si combinano per creare funzionalità più complesse.

Ipotesi funzionalità

Grazie alle nozioni, ho cercato passo passo di **identificare** i vari **costrutti** del C, nel codice assembly fornito dalla traccia. Prima di esporre i costrutti individuati, faccio un'ipotesi sulla "**potenziale**" funzionalità del codice. E credo proprio che questo codice chiami la funzione **InternetGetConnectedState**, la quale **verifica se la macchina ha accesso a Internet**. Se la connessione risulta **attiva**, allora stampa un messaggio di successo. In caso contrario, non esegue nulla di visibile (a quanto pare). Ovviamente anche i **commenti** in questo caso hanno aiutato molto.

Cercando in rete quella funzione, credo ci sia un errore nella traccia. I parametri sono in realtà 2:

dwReserved e **lpdwFlags**

```
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_401028
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40105F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

Chiama la funzione che verifica
l'accesso a internet

Chiama la funzione sub_40105F,
presumibilmente quella che si occupa di
stampare il messaggio

Spiegazione codice

```
.text:00401000  
.text:00401001  
.text:00401003
```

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

push ebp: Salva il valore del registro base pointer (ebp) sullo stack. Questo è fatto per preservare il valore del registro prima di modificarlo.

mov ebp, esp: Imposta ebp per puntare all'inizio dello stack corrente. In pratica, ebp diventa il nuovo base pointer per la funzione **chiamante**, puntando all'inizio del frame della funzione **chiamata**.

push ecx: Salva il registro ecx sullo stack.

Spiegazione codice

```
.text:00401004  
.text:00401006
```

```
push 0  
push 0
```

push 0: Passa i parametri SULLO STACK, alla funzione InternetGetConnectedState. Viene passato 0 sia per **dwReserved** che per **lpdwFlags**. Vengono pushati prima della chiamata alla funzione.

```
.text:00401008
```

```
call ds:InternetGetConnectedState
```

call ds: Chiama la funzione **InternetGetConnectedState** per controllare lo stato della connessione Internet.

Spiegazione codice

```
.text:0040100E      mov     [ebp+var_4], eax
```

mov [ebp+var_4], eax: Salva il valore di ritorno della funzione InternetGetConnectedState (nel registro eax) nella variabile locale **var_4**.

```
.text:00401011      cmp     [ebp+var_4], 0
```

cmp [ebp+var_4], 0: Confronta il valore di **var_4** con 0.

```
.text:00401015      jz      short loc_40102B
```

jz short loc_40102B: Se var_4 è uguale a 0, salta all'etichetta loc_40102B.

Spiegazione codice

```
.text:00401017  
.text:0040101C  
.text:00401021
```

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"  
call    sub_40105F  
add     esp, 4
```

push offset aSuccessInterne: Spinge l'offset del messaggio di successo ("Success: Internet Connection\n") sullo stack.

call sub_40105F: Chiama una funzione (presumibilmente sub_40105F) per stampare il messaggio.

add esp, 4: Ripristina lo stack, rimuovendo il parametro aSuccessInterne.

```
.text:00401024  
.text:00401029  
mov     eax, 1  
jmp     short loc_40103A
```

mov eax, 1: Imposta il valore di ritorno a 1.

jmp short loc_40103A: Salta all'etichetta loc_40103A, terminando l'esecuzione.

Costrutti noti in C

Costrutto assembly	Descrizione Assembly	Costrutto C
call ds:InternetGetConnectedState	Chiamata alla funzione InternetGetConnectedState	InternetGetConnectedState(&dw Flags, 0);
mov [ebp+var_4], eax	Assegnazione del valore di EAX alla variabile locale var_4	var_4 = eax;
cmp [ebp+var_4], 0 e jz short loc_40102B	Confronto e salto condizionale (se zero)	if (var_4 == 0) { ... }
jmp short loc_40103A	Salto incondizionato	goto continueExecution;
push offset aSuccessInterne e call sub_40105F	Preparazione e chiamata a una funzione per l'output	printf("Success: Internet Connection\n");



GRAZIE

Flavio Scognamiglio