S10-L4

(Bonaldi Cristian)

Costrutti C - Assembly x86

Traccia:

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

```
.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
```

Provate ad ipotizzare che funzionalità è implementata nel codice assembly.

1. Identificare i costrutti noti (e s. while, for, if, switch, ecc.)
2. Ipotizzare la funzionalità – esecuzione ad alto livello
3. BONUS: studiare e spiegare ogni singola riga di codice

Esercizio e sviluppo:

Nell'esercizio di oggi si identificano i costrutti noti in C (while,for,if,switch etc), ipotizzando anche la funzionalità del codice Assembly dato dalla traccia.

Si procede quindi nella lettura e nell'interpretazione del codice Assembly riga per riga, per comprenderne appieno il funzionamento.

Come mostrato in figura, questo rappresenta la creazione di uno stack. In sostanza **push ebp** salva il valore del registro EBP (Base Pointer) nello stack, dopodiché con **mov ebp, esp** si va a copiare il valore dello stack pointer, ovvero **esp** nel registro ebp.

L'istruzione **push ecx** mette il valore corrente di **ecx** nello stack, In questo modo, salvandolo nello stack, garantiamo che il suo valore originale non venga accidentalmente modificato durante

l'esecuzione di altre istruzioni o funzioni. In questo modo, possiamo recuperare il valore esatto di **ecx** in seguito, quando ne avremo di nuovo bisogno, assicurandoci che non si perda nessuna informazione importante.

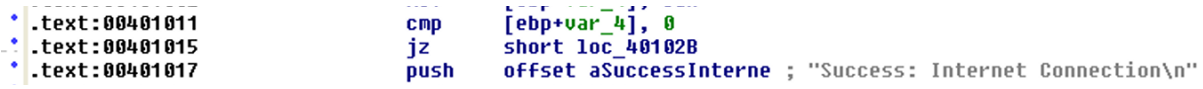
Le due istruzioni **push 0** nello stack servono per passare gli argomenti alla funzione InternetGetConnectedState, rispettivamente 0 e 0.
Mentre **call ds:InternetGetConnectedState** è l'istruzione che chiama la funzione InternetGetConnectedState, che verifica lo stato della connessione a Internet.

IN C

chiamata della funzione **InternetGetConnectedState(0, 0);** dove vengono passati due argomenti (0 e 0).

---

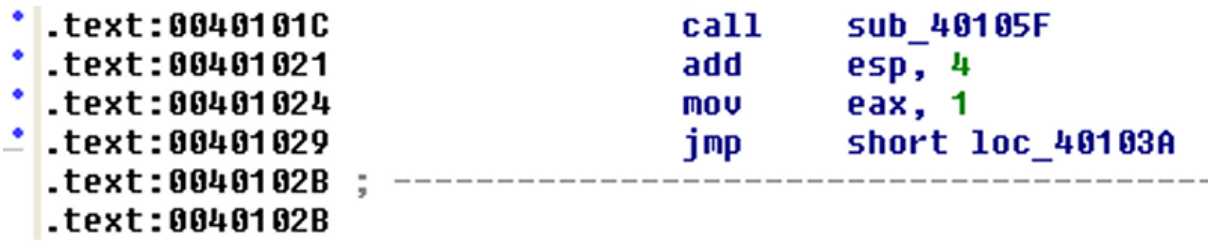
mov [ebp+var_4], eax va a memorizzare il valore di ritorno della funzione (che è in eax) in una variabile locale **(var_4).**



Dopodichè, tramite il cmp, confronta il valore della variabile locale con 0.
Se il valore confrontato è uguale a 0, e quindi ZF (Zero Flag) = 1, fa un jump alla locazione loc_40102B (infatti jz = jump if zero).
Con l'istruzione **push offset aSuccessInterne** mette l'indirizzo della stringa "*Success: Internet Connection\n*" nello stack, rendendo questo testo disponibile come argomento per la funzione che verrà chiamata subito dopo.

IN C

Questo è un costrutto **if**, che verifica se una condizione è vera (in questo caso, se la funzione InternetGetConnectedState restituisce 0). Se la condizione è vera, il codice all'interno del blocco if viene saltato, altrimenti il programma prosegue con le istruzioni successive.



Adesso **call** va a chiamare una funzione (sub_40105F, probabilmente un **printf**) per stampare la stringa passata come argomento.

Si procede quindi con l'istruzione **add esp, 4** che serve a ripulire lo stack dopo aver chiamato la funzione precedentemente. Infatti, quando passiamo un argomento alla funzione, mettendolo quindi nello stack, vengono occupati **4 byte** di spazio, dato che un registro su un'architettura a 32 bit è lungo 4 byte. Dopo aver usato l'argomento, dobbiamo rimuoverlo per ripristinare l'ordine nello stack, ed è per questo che aggiungiamo **4** a esp, che è il puntatore dello stack.
Infine, **mov eax, 1** imposta il valore di ritorno della funzione a 1 e **jmp short loc_40103A** salta alla fine della funzione, concludendo il blocco condizionale.

IN C

Questo potrebbe essere l'equivalente di **return 1;** in C.

Spiegazione del flusso e funzionalità:

Il codice utilizza la funzione InternetGetConnectedState per determinare se il computer è connesso a Internet. Vengono passati due argomenti a questa funzione, entrambi con valore 0 e il valore di ritorno di questa funzione viene memorizzato nel registro **eax**, che è comunemente usato per restituire valori dalle funzioni.
Il codice confronta il valore restituito da InternetGetConnectedState con zero usando l'istruzione **cmp**. Questo confronto determina se c'è una connessione attiva. Se il valore in EAX è maggiore o uguale a zero (e quindi che è stata rilevata una connessione) il codice esegue un jump e procede a chiamare la funzione **printf**, passando una stringa contenente il messaggio: "*Success: Internet Connection\n*".
Dopo la chiamata a printf, il codice ripulisce lo stack per assicurarsi che non ci siano argomenti residui. Viene aggiunto un valore a esp, che è il puntatore

dello stack (il valore in questione è 4) per rimuovere gli argomenti passati, mantenendo così lo stack in uno stato coerente.