

## COSTRUTTI C – ASSEMBLY x86

Identificazione di costrutti noti in C ed analisi delle funzionalità di un programma

```

* .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:0040102D

```

Push ebp

Mov ebp, esp

Push ecx

Push 0 ; dwReversed

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

## Creazione dello stack

```
Push ebp
```

```
Mov ebp, esp
```

Le prime due righe di codice servono a creare uno stack per le variabili locali: notiamo la presenza dei due puntatori **EBP** (Extended Base Pointer) ed **ESP** (Extended Stack Pointer) che puntano rispettivamente alla base ed alla cima dello stack. Notiamo anche che non viene immediatamente definito lo spazio dedicato alle variabili locali.

**Stack**: tipologia di memoria che viene utilizzata per le **variabili locali** ed i parametri delle funzioni, e per supportare il flusso del programma. Lo stack è molto interessante per l'analisi dei malware: è una pila di piatti dove possiamo aggiungere o rimuovere un piatto dalla cima secondo struttura LIFO (last in first out), azione di aggiungere piatto si chiama **push**, togliere un piatto = **pop**.

## Funzione InternetGetConnectedState

```
Push ecx
```

```
Push 0 ; dwReversed
```

```
Push 0 ; lpdwFlags
```

```
Call ds: InternetGetConnectedState
```

Con le prime tre operazioni push vengono inserite in cima allo stack tre parametri (ecx, 0 e 0) di variabili, che saranno poi dati in pasto alla funzione **InternetGetConnectedState**, che ha lo scopo di verificare se una macchina ha accesso ad Internet.

## IF Statement

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

```
Jz short loc_40102B
```

Notiamo per prima cosa la presenza di un'istruzione cmp (= compare), che si occuperà di confrontare – facendo la differenza – la variabile scritta nel registro EBP+var\_4 e 0. Se il risultato di questa operazione dà come valore 0, la ZF (Zero Flag) si aggiorna con il valore booleano di 1 e la

condizione jz (Jump Zero) viene confermata: viene dunque fatto un “salto” breve alla locazione di memoria con indirizzo 40102B. Se invece il risultato dell’operazione fosse diverso da 0, la ZF assume valore 0 ed il programma continuerebbe ad eseguire le righe di codice successive, fino ad arrivare al salto non condizionale all’indirizzo di memoria 40103A.

#### **push ebp**

Con questa Istruzione viene "pushato" l'extended base pointer sulla cima dello stack

#### **mov ebp, esp**

Qui invece viene assegnato il valore del registro dell'Extended Stack pointer al registro dell'Extended Base Pointer

#### **push ecx**

Qui, tramite l'istruzione push, viene posto il valore inserito nel registro "ecx" in cima allo stack

#### **Push 0 ; dwReserved**

Crea un buffer vuoto di 4 byte sullo stack. Dal commento (“dwReserved”) si può notare che questo parametro, che verrà utilizzato dalla successiva funzione chiamata, è riservato e deve essere 0

#### **push 0 ; lpdwFlags**

Come sopra, questa istruzione crea un buffer vuoto di 4 byte. Dal commento accanto ho visto che è relativo a un puntatore che riceve la descrizione della connessione tramite la funzione chiamata dalla riga di comando immediatamente successiva.

#### **call ds: InternetGetConnectedState**

Viene chiamata la funzione "InternetGetConnectedState" che recupera lo stato connesso del sistema locale

#### **mov [ebp+var\_4], eax**

Viene copiato il valore contenuto nel registro "eax" nel registro "ebp+var\_4", cioè nel registro a distanza di 4 byte verso la cima dello stack

#### **cmp [ebp+var\_4], 0**

Con questa istruzione viene attuata una sottrazione tra il parametro contenuto nel registro "ebp+4\_var" e 0 andando a modificare la zero flag del registro. Questo valore sarà fondamentale per la successiva istruzione di jump

#### **jz short loc\_40102B**

Con questa istruzione di jump viene controllata la zero flag ottenuta dalla precedente istruzione "cmp". Se questa risulterà uguale a 1 verrà effettuato uno "short jump" all'indirizzo di memoria "40102B", altrimenti verranno eseguite normalmente le successive righe di codice

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

Con questa istruzione viene inserita la stringa "aSuccessInterne" in un registro in cima allo stack

#### **call sub\_40105F**

Viene chiamata una funzione inserita nell'indirizzo di memoria "40105F"

#### **add esp, 4**

Con questa istruzione viene effettuata una somma tra il valore inserito all'interno del registro "esp"

#### **mov eax, 1**

Viene sostituito il valore contenuto all'interno del registro "eax" con il valore 1

#### **jmp short loc\_40103A**

Viene effettuato un salto all'indirizzo di memoria "40103A"