**Alaimo Alessandro**
**11/01/2023**

# Report Assemby x86

0x00001141 <+8>: mov EAX, 0x20

- Tramite il comando mov andiamo ad impostare come valore immediato 0x20 in esadecimale oppure 32 in decimale al registro EAX. L'equivalente di questa stringa in C sarebbe: a=32;

0x00001148 <+15>: mov EDX, 0x38

- Tramite il comando mov andiamo ad impostare come valore immediato 0x38 in esadecimale oppure 56 in decimale al registro EDX. L'equivalente di questa stringa in C sarebbe: d=56;

0x00001155 <+28>: add EAX, EDX

- Tramite il comando add andiamo a sommare il registro EDX in EAX andando ad aggiornarne il valore che adesso è 88. Il suo equivalente in C sarebbe: a=a+d;

0x00001157 <+30>: mov EBP, EAX

- Tramite il comando mov andiamo ad impostare il valore del registro EAX al registro EBX. L'equivalente di questa stringa in C sarebbe: b=a;

0x0000115a <+33>: cmp EBP, Oxa

- Tramite il comando cmp andiamo a fare un sub, il quale equivale alla sottrazione, senza aggiornare il valore di EBP. Questo comando serve per verificare se il numero è 0, ha avuto un riporto oppure nessuno dei due. Nel nostro caso abbiamo: EBP (88) – 10 = 78. Ergo, in questo caso, abbiamo la Zero Flag a 0 e la Carry Flag a 0;

0x0000115e <+37>: jge 0x1176 <main+61>

- Tramite il comando jge andiamo a fare un salto nell'indirizzo di memoria 0x1176 solo nel caso in cui la destinazione sia di valore maggiore o uguale alla sorgente nell'istruzione cmp;

0x0000116a <+49>: mov EAX, 0x0

- Tramite il comando mov andiamo ad impostare come valore immediato 0x0 in esadecimale oppure 0 in decimale al registro EAX. L'equivalente di questa stringa in C sarebbe: a=0;

0x0000116f <+54>: call 0x1030 <printf@plt>

- L'istruzione call viene utilizzata per chiamare una funzione.
  L'istruzione call esegue due operazioni:

“Spinge” l'indirizzo di ritorno (indirizzo immediatamente successivo all'istruzione CALL) sullo stack.
Cambia l'EIP nella destinazione della chiamata. In questo modo si trasferisce il controllo alla destinazione della chiamata e si avvia l'esecuzione.

Printf@plt è in realtà un piccolo stub,una porzione di codice utilizzata per simulare il comportamento di funzionalità software o l'interfaccia COM e può fungere anche da temporaneo sostituto di codice ancora da sviluppare, che (alla fine) chiama la vera funzione printf, modificando le cose durante il percorso per rendere più veloci le chiamate successive.

La vera funzione printf è mappata in una posizione arbitraria in un dato processo (spazio degli indirizzi virtuali), così come il codice che cerca di chiamarla.

Fonti : https://www.aldeid.com/wiki/X86-assembly/Instructions/call
https://stackoverflow.com/questions/5469274/what-does-plt-mean-here