REPORT ASSEMBLY

FRANCESCO PERSICHETTI

L'esercizio di oggi consiste nell'identificare lo scopo di ogni istruzione sotto riportata:

- 0x00001141 <+8>: mov EAX,0x20:
  Con l'operatore "mov" imposto il valore immediato al registro EAX che sarebbe 0x20 in esadecimale; in decimale equivale a 32

- 0x00001148 <+15>: mov EDX,0x38:
  Con l'operatore "mov" imposto il valore immediato al registro EDX che sarebbe 0x38 in esadecimale; in decimale equivale a 56

- 0x00001155 <+28>: add EAX, EDX:
  Con l'operatore "add" vado appunto a fare l'addizione tra EAX e EDX. Visto che nei due punti precedenti abbiamo già calcolato il valore decimale, la somma sarà (EAX = 32) + (EDX = 56) = 88

- 0x00001157 <+30>: mov EBP, EAX:
  In questo caso con "mov" imposto il valore di registro di EAX a EBP

- 0x0000115a <+33>: cmp EBP,Oxa:
  Con il comando "cmp" come abbiamo visto equivale all'operazione "sub" quindi vado a sottrarre senza modificare gli operandi

- 0x0000115e <+37>: jge 0x1176 <main+61>:
  Con l'operatore "jge" si fa un salto alla locazione di memoria specificata se la destinazione è maggiore o uguale della sorgente nell'istruzione "cmp"

- 0x0000116a <+49>: mov EAX,0x0:
  Con l'operatore "mov" imposto il valore immediato al registro EAX che sarebbe 0x0 in esadecimale; in decimale equivale a 0

- 0x0000116f <+54>: call 0x1030 <printf@plt>:
  L'istruzione "CALL" viene utilizzato per chiamare una funzione, ed eseguire due operazioni:
  "Push" l'indirizzo di ritorno 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 un piccolo stub, cioè porzione di codice che simula il comportamento di funzionalità software, che chiama la vera funzione printf. La vera funzione printf è mappata in una posizione arbitraria in un dato processo, così come il codice che cerca di chiamarla