Analisi di un codice Assembly x86

1. 0x00001141 <+8>: mov EAX,0x20
2. 0x00001148 <+15>: mov EDX,0x38
3. 0x00001155 <+28>: add EAX, EDX
4. 0x00001157 <+30>: mov EBP, EAX
5. 0x0000115a <+33>: cmp EBP,0xa
6. 0x0000115e <+37>: jge 0x1176 <main+61>
7. 0x0000116a <+49>: mov eax,0x0
8. 0x0000116f <+54>: call 0x1030 <printf@plt>

Identifico lo scopo di ogni istruzione:

1. **mov EAX,0x20**: Carica il valore esadecimale 0x20 nel registro EAX.
2. **mov EDX,0x38**: Carica il valore esadecimale 0x38 nel registro EDX.
3. **add EAX,EDX**: Somma il contenuto dei registri EAX e EDX e memorizza il risultato in EAX.
4. **mov EBP, EAX**: Copia il valore corrente di EAX nel registro EBP.
5. **cmp EBP,0xa**: Compara il valore nel registro EBP con il valore esadecimale 0xA
6. **jge 0x1176 <main+61>**: Salta all'indirizzo 0x1176 se il risultato della comparazione precedente è maggiore o uguale (jump if greater or equal).
7. **mov eax,0x0**: Carica il valore esadecimale 0x0 nel registro EAX.
8. **call 0x1030 <printf@plt>**: Chiama la funzione printf dalla posizione 0x1030.

In base a questa parte di codice assembly x86, possiamo presumere che questo codice stia eseguendo alcune operazioni aritmetiche e un controllo condizionale.

In sostanza, il codice confronta se la somma di 0x20 e 0x38 è maggiore o uguale a 10. Se è così, salta a un'etichetta non specificata (0x1176).

Se la condizione è falsa, imposta EAX a 0 e chiama la funzione printf, probabilmente per stampare qualche messaggio a schermo.

1) 0x20=32 in decimale EAX=32

2) 0x38=56 in decimale EDX=56

3) 32+56=88 in decimale EAX+EDX=EAX(Aggiunto)

4) EBP=88 in decimale EBP=EAX

5) confronto tra 88(des) – 10(sor)=Modifica flag (ZF0,CF0)

6) condizione 4470 >= flag78 else <main+61> indirizza ad un’altra linea di memoria

7) carica il valore esadecimale 0x0=0 in decimale nel registro EAX+0=EAX(Aggiunto)

8) chiama la funzione <printf@ptl> dalla posizione 0x1030=4144 in decimale