Floriana Feminò

11/01/2023

ASSEMBLY x86

Task:

Analisi ed identificazione dello scopo di istruzioni scritte in linguaggio Assembly

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

Questa istruzione ha lo scopo di spostare tramite l'operazione ***mov*** il valore 0x20 – espresso in numeri esadecimali – nel registro di tipo general purpose EAX; il valore 0x20 corrisponde al numero decimale 32.

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

Questa istruzione ha lo scopo di spostare tramite l'operazione *mov* il valore esadecimale 0x38 nel registro EDX; il valore 0x38 corrisponde al numero decimale 56, di conseguenza al registro EDX viene assegnato il valore 56.

0x00001155 <+28>: add EAX, EDX

Questa istruzione ha lo scopo di sommare i valori dei registri EAX ed EDX, aggiornando poi il valore del registro EAX con la somma dell'operazione aritmetica svolta. Sappiamo che il valore di EAX in numeri decimali è **32** e quello di EDX è **56**, pertanto sommando i due valori otteniamo il nuovo valore **88**, che viene assegnato al registro EAX.

0x00001157 <+30>: mov EBP, EAX

Tramite questa istruzione viene spostato tramite l'operazione *mov* il valore di EAX all'interno del registro EBP, che assumerà quindi il valore di 88.

0x0000115a <+33>: cmp EBP, 0xa

In questa riga viene introdotta l'istruzione "**cmp**". Essa si comporta in maniera similare all'istruzione "sub", usata per sottrarre i valori di 2 registri, senza però andare a modificare i due operandi. Tuttavia questa operazione va a modificare lo **zero flag (ZF)** ed il **carry flag (CF)**, che si usa per gestire eventuali riporti in un'operazione aritmetica. Questi sono degli **status flag**, valori che possono avere solo valori 1 o 0 in base alle seguenti casistiche:

Destinazione = sorgente → ZF 1 CF 0

Se la sorgente è uguale alla destinazione, si avrà una sottrazione tra due numeri uguali es. sub 5, 5 = 0. Visto che il risultato è 0, lo ZF viene settato a 1

Destinazione < sorgente → ZF 0 CF 1

Se la destinazione è minore della sorgente, si avrà una sottrazione come sub 2, 5 = -3 che verrà gestito come un numero con "prestito" (riporto), quindi CF è 1 e ZF è 0

Destinazione > sorgente → ZF 0 CF 0

Se la destinazione è maggiore della sorgente, sia ZF che CF sono 0

In questa specifica riga di comando, l'operazione è **88 (EBP) – 10 (0xa) =78**. Ne consegue che ZF e CF assumono entrambi valore 0 in quanto 88 > 10.

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

In questa riga viene introdotta l'istruzione "**conditional jump**" (= salto condizionale). In questo caso l'istruzione consiste nel salto alla locazione di memoria 0x1176 nel caso in cui la destinazione abbia un valore maggiore o uguale al valore dell'istruzione "cmp" contenuta nella riga vista precedentemente.

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

In questa riga, tramite l'istruzione *mov* viene assegnato il valore esadecimale 0x0, che in sistema decimale equivale a 0, al registro EAX.

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

Nel linguaggio assembly una funzione viene chiamata con l'istruzione **CALL**: la funzione chiamante passa l'esecuzione del programma alla funzione chiamata per la quale verrà creato un nuovo stack.

In questa riga, l'istruzione ***call*** esegue 2 operazioni:

1. Applica un push dell'indirizzo immediatamente successivo (0x1030) sullo stack
2. Cambia l'Extended Instruction Pointer (EIP) nella destinazione di chiamata. In questo modo viene trasferito il controllo al target dell'istruzione "call" e comincia l'esecuzione da lì. Il <printf@plt> consiste in un piccolo stub che – alla fine – chiama la vera funzione printf, che è mappata in una posizione arbitraria in un dato processo (spazio di indirizzi virtuale), così come il codice che sta tentando di chiamarla. Uno **stub** è una porzione di codice utilizzata per simulare il comportamento di funzionalità software (come una routine su un sistema remoto) o l'interfaccia COM e può fungere anche da temporaneo sostituto di codice ancora da sviluppare. Sono pertanto utili durante il porting di software, l'elaborazione distribuita e in generale durante lo sviluppo di software e il software testing.

Fonti:

https://stackoverflow.com/questions/5469274/what-does-plt-mean-here

https://it.wikipedia.org/wiki/Stub_(informatica)

https://www.aldeid.com/wiki/X86-assembly/Instructions/call