S10 L3

Report esercizio “Malware Analysis: La memoria ed il linguaggio Assembly”

GiuliaSalani

TRACCIA

Nella lezione teorica del mattino, abbiamo visto i fondamenti del linguaggio Assembly. Dato il codice in Assembly per la CPU x86 allegato qui di seguito, identificare lo scopo di ogni istruzione, inserendo una descrizione per ogni riga di codice.

Ricordate che i numeri nel formato 0xYY sono numeri esadecimali. Per convertirli in numeri decimali utilizzate pure un convertitore online, oppure la calcolatrice del vostro computer (per programmatori).

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

INTRODUZIONE

L'assembly è un linguaggio di basso livello, legato strettamente all'architettura del processore, composto da istruzioni mnemoniche comprensibili dalla CPU. Contrariamente ai linguaggi di alto livello, è più vicino al linguaggio macchina e offre un controllo diretto sull'hardware.
Nel contesto della malware analysis, l'assembly è cruciale. Il codice sorgente di malware spesso non è disponibile, e il reverse engineering coinvolge l'analisi del codice binario. Gli analisti di malware utilizzano l'assembly per disassemblare e comprendere il comportamento del malware. Questo coinvolge l'identificazione delle funzioni, la comprensione dei flussi di controllo e la rivelazione delle tecniche evasive utilizzate dal malware per sfuggire alla rilevazione.
L'assembly è inoltre fondamentale per capire le vulnerabilità sfruttate dai malware e sviluppare contromisure di sicurezza. Conoscere approfonditamente assembly consente agli analisti di malware di individuare e comprendere le tattiche di attacco, aiutando nella protezione e nella mitigazione dei rischi associati alle minacce informatiche.

SVOLGIMENTO

0x00001141 <+8>: mov EAX,0x20	Spostamento	Assegna a EAX il valore 32
0x00001148 <+15>: mov EDX,0x38	Spostamento	Assegna a EDX il valore 56
0x00001155 <+28>: add EAX,EDX	Addizione	Aggiunge il valore di EDX a EAX
0x00001157 <+30>: mov EBP, EAX	Spostamento	Assegna a EBP il valore di EAX
0x0000115a <+33>: cmp EBP,0xa	Confronto	Confronta il valore contenuto nel registro EBP con il valore 10. Dopo questa istruzione, i flag del registro dei flag saranno impostati in base al risultato della sottrazione EBP – 10
0x0000115e <+37>: jge 0x1176 <main+61>	Salto condizionale	Se l'istruzione cmp ha confrontato EBP con il valore 10 e ha stabilito che EBP è maggiore o uguale a 10, l'istruzione jge 0x1176 salterà alla locazione di memoria 0x1176. Altrimenti, il flusso del programma continuerà normalmente con l'istruzione successiva.
0x0000116a <+49>: mov eax,0x0	Spostamento	Assegna a EAX il valore 0
0x0000116f <+54>: call 0x1030 <printf@plt>	Chiamata	Invoca la funzione printf