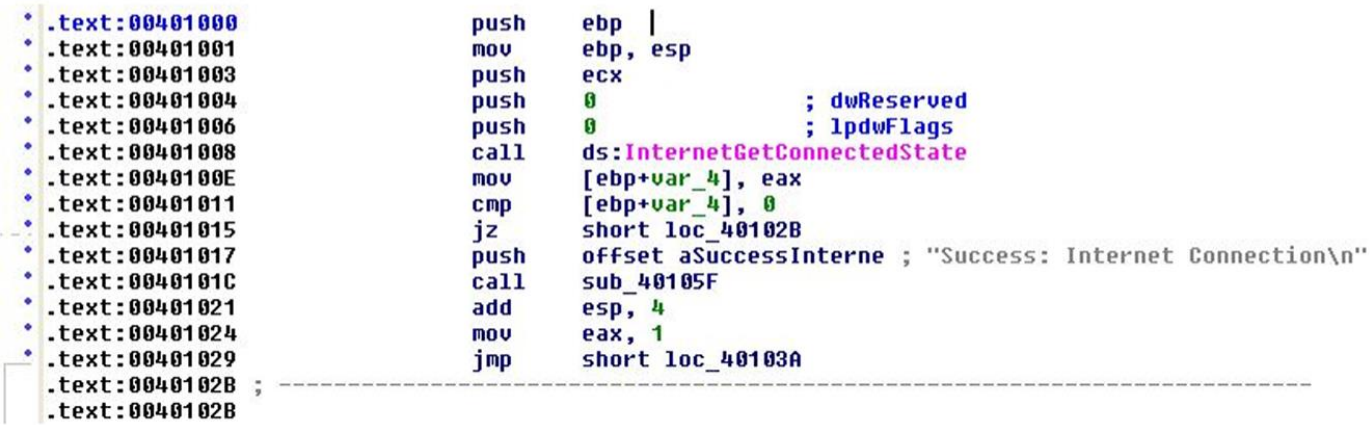
Report S10-L4
Costrutti C - Assembly X86

Il presente report riporta lo svolgimento dell'esercitazione che consiste nell'identificazione di costrutti noti in C nel codice Assembly fornito. Si riporta il tentativo di analizzare il funzionamento del malware.

Sommario

Traccia:

La figura seguente mostra un estratto del codice di un malware.
Identificare i costrutti noti visti durante la lezione teorica.



```
.text:00401000                push    ebp
.text:00401001                mov     ebp, esp
.text:00401003                push    ecx
.text:00401004                push    0               ; dwReserved
.text:00401006                push    0               ; lpdwFlags
.text:00401008                call    ds:InternetGetConnectedState
.text:0040100E                mov     [ebp+var_4], eax
.text:00401011                cmp     [ebp+var_4], 0
.text:00401015                jz      short loc_40102B
.text:00401017                push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                call    sub_40105F
.text:00401021                add     esp, 4
.text:00401024                mov     eax, 1
.text:00401029                jmp     short loc_40103A
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
```

Opzionale:
Provate ad ipotizzare che funzionalità è implementata nel codice assembly.

Hint:
La funzione internetgetconnectedstate prende in input 3 parametri e permette di controllare se una macchina ha accesso ad Internet.

Introduzione Teorica

Reverse Engineering

L'operazione che viene richiesta di effettuare nella traccia è detta Reverse Engineering, ovvero quel **processo di ricostruzione ad un più alto livello delle funzionalità di un malware a partire dall'analisi del codice Assembly**.

In altri termini, nel processo di Reverse Engineering si effettua la **decompilazione**, cioè la **"traduzione" del codice Assembly, di basso livello, in un linguaggio di programmazione di livello più alto**.

Si tratta di un'operazione fondamentale perché **il codice assembly di un malware**, essendo diretto a dare istruzioni direttamente all'Hardware, all'architettura del processore, della macchina, **è molto complesso e difficile da comprendere per gli analisti** che stiano cercando di capire le funzionalità del software dannoso.
Invece, **durante la decompilazione, il codice assembly viene convertito in un linguaggio** di programmazione di livello basso, come C o C++, che è, però, **più vicino al linguaggio umano di quanto non lo sia Assembly**.

Questo rende **più accessibile la comprensione del funzionamento del malware** agli analisti di sicurezza, poiché il codice risultante è più vicino alla struttura concettuale della programmazione tradizionale.

Assembly

Quindi, Assembly è il linguaggio di programmazione di basso livello più vicino al linguaggio macchina, è specifico per ciascuna architettura hardware e contiene set di istruzioni che vengono eseguite direttamente dalla CPU (Unità di Elaborazione centrale).

Quando un programma scritto in linguaggio assembly viene eseguito, le istruzioni assembly vengono convertite in linguaggio macchina, il linguaggio binario comprensibile direttamente dalla CPU.
La CPU interpreta e esegue queste istruzioni in modo sequenziale, manipolando i dati e controllando il flusso del programma. Quindi, ogni istruzione Assembly, eseguita direttamente dalla CPU, svolge un'operazione, e il complesso delle operazioni svolte dall'insieme delle istruzioni compone un programma completo.
In sostanza, il linguaggio assembly agisce come un'interfaccia tra il linguaggio di alto livello (più comprensibile agli sviluppatori) e il linguaggio macchina della CPU (più comprensibile alla macchina stessa), offrendo un modo comprensibile, per gli sviluppatori, di scrivere istruzioni che possono essere tradotte direttamente in istruzioni binarie eseguibili dalla CPU del computer.
Infatti, La CPU, a sua volta, interpreta ed esegue le istruzioni assembly convertendole in azioni concrete sulla macchina
Gli sviluppatori scrivono in linguaggio assembly per avere un maggiore controllo sulle operazioni effettuate dalla CPU, spesso per ottimizzare le prestazioni o implementare funzionalità specifiche a un livello di dettaglio più fine.
Infatti, Il linguaggio Assembly, offrendo accesso diretto all'hardware, consente un controllo molto preciso sulle risorse del computer, permettendo, quindi, agli sviluppatori si gestire direttamente la memoria, i registri e altri spetti dell'architettura del processore.

Svolgimento – identificazione costrutti noti di C a partire da Codice Malware in Assembly

Setup del Frame Pointer: **push ebp** seguito da **mov ebp, esp** sono istruzioni standard per la configurazione del frame pointer in una nuova funzione, che serve a salvare il contesto attuale dello stack prima di eseguire il resto del codice.

Push di argomenti per una chiamata di funzione: Le istruzioni **push 0** due volte stanno probabilmente mettendo due argomenti sullo stack.
Questo è tipico per una funzione che accetta argomenti, che in assembly vengono passati tramite lo stack.

Chiamata di funzione: **call ds:InternetGetConnectedState** è una chiamata a una funzione importata che, come suggerisce l'hint, verifica se la macchina ha accesso a internet.
In Windows, InternetGetConnectedState è una funzione dell'API di Windows che determina lo stato della connessione a internet.

Gestione del valore di ritorno: Dopo la chiamata alla funzione, il valore di ritorno viene confrontato con zero (**cmp [ebp+var_4], 0**).
Se la funzione restituisce zero (che significa nessuna connessione internet), il flusso del programma salta a una locazione etichettata (**jz short loc_40192B**), che è tipicamente usata per la gestione dell'errore o per eseguire codice condizionale basato sul risultato di una funzione.

Gestione dello stack post-chiamata: **add esp, 4** pulisce lo stack dopo la chiamata alla funzione, rimuovendo gli argomenti che erano stati passati.

Messaggio di successo: L'istruzione **push offset aSuccessInterne** sembra riferirsi al push di un messaggio di successo sulla pila, forse per essere usato in seguito per un output o un log, suggerendo che il codice successivo (non mostrato nell'immagine) gestirà il caso di connessione internet riuscita.

Conclusione

Basandosi su questi elementi e sull'hint fornito, si può ipotizzare che questo frammento di codice è progettato per **verificare se il sistema su cui il malware è eseguito ha una connessione internet attiva**.

Se la connessione è presente, il malware potrebbe proseguire con la sua attività malevola, come connettersi a un server di comando e controllo o scaricare ulteriori payload malevoli.

Nel caso in cui la connessione non sia attiva, potrebbe saltare alcune funzionalità o tentare di stabilire una connessione in altri modi.