**Traccia:**

La figura seguente mostra un estratto del codice di un malware:

* - 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 : 004010011 cmp [ebp+var\_4], 0
* - text : 004010015 jz short loc\_40102B
* - text : 004010017 push offset aSuccessInterne ; “Success: Internet Connection\n”
* - text : 00401001C call sub\_40105F
* - text : 004010021 add esp, 4
* - text : 004010024 mov eax, 1
* - text : 004010029 jmp short loc\_40103A
* - text : 00401002B ; -------------------------------------------------------------------------------------------------------------------------------------------
* - text : 00401002B

Ai fini dell’esercizio:

1. Identificare i costrutti noti (es. while, for, if, switch, ecc.);
2. Ipotizzare la funzionalità/esecuzione ad alto livello;
3. BONUS: studiare e spiegare ogni singola riga di codice.
4. -----------------------------------------------------------------------------------------------------------------------------------------------------------|

 🡪 Costrutto IF

-----------------------------------------------------------------------------------

 🡪 Chiave GO TO

Di seguito sono elencati alcuni costrutti noti presenti nel codice con il corrispettivo in linguaggio C:

* **Push e Pop**: Istruzioni utilizzate per spingere e popolare valori sullo stack. In C può essere ottenuto usando operazioni sullo stack o su strutture dati come *array o liste*.
* **Mov**: Istruzione utilizzata per muovere dati tra registri, variabili o memoria. In C può essere rappresentata assegnando un *valore a una variabile o copiando il contenuto di una variabile in un'altra*.
* **Call**: Utilizzata per chiamare una funzione o un sotto-programma. L'equivalente in C sarebbe la chiamata di una funzione tramite la sua *firma*.
* **Cmp**: Utilizzata per confrontare due valori. Un'equivalente in C sarebbe l'uso degli *operatori di confronto come* ***==****,* ***<****,* ***>****,* ecc.
* **Jz (Jump if Zero)**: Salto condizionale che avviene se il confronto precedente ha dato come risultato zero (Z set). In C, il corrispondente sarebbe l'istruzione *if* (in figura) che verifica se una condizione è vera, se la condizione è zero, salta all'etichetta specificata.
* **Offset**: Utilizzato per ottenere l'indirizzo di memoria di un'etichetta o una variabile. In C, si possono ottenere gli indirizzi utilizzando l'operatore *&*.
* **Add**: Utilizzato per aggiungere due operandi. L'equivalente in C sarebbe l'operatore di *addizione* ***+***.
* **Jmp (Jump)**: Salto incondizionato a un'altra parte del codice. In C è equivalente alla parola *chiave go to* (in figura), che permette di saltare a un'etichetta specifica nel codice.
* **Etichette**: Punti di riferimento nel codice, spesso utilizzati come destinazioni per i salti condizionali o incondizionati. In C, le etichette sono utilizzate per definire *punti di ingresso o uscita da cicli o per facilitare il salto a parti specifiche del codice*.
* **Variabili locali**: Accesso alle variabili locali tramite l'uso di ebp come base del frame di stack. In C, le variabili locali sono dichiarate *all'interno di una funzione e sono accessibili solo all'interno di quella funzione*.
* **Commenti**: Parti del codice che forniscono informazioni aggiuntive per facilitare la comprensione del codice. In C, i commenti vengono utilizzati per documentare il codice e non vengono tradotti in istruzioni di linguaggio macchina. I commenti possono essere aggiunti al codice utilizzando ***//*** *per commenti su una singola riga o* ***/\* \*/*** *per commenti su più righe*.

1. -----------------------------------------------------------------------------------------------------------------------------------------------------------|

Questa porzione di codice sembra controlli la connessione Internet e stampi un messaggio di successo se la connessione è attiva.

Potrebbe essere una parte di un malware che necessita di una connessione attiva per eseguire determinate azioni o comunicare con un server remoto.

Provando a fornire un’interpretazione ad alto livello potremmo dire che:

* Il codice sembra iniziare con l'inizializzazione e il controllo della connessione Internet (funzione **InternetGetConnectedState**);
* Il valore di ritorno viene memorizzato in **[ebp+var\_4]**;
* Successivamente, viene effettuato un confronto per vedere se il valore memorizzato è *uguale a zero*;
* Se il risultato del controllo della connessione è *positivo* (**diverso da zero**), viene stampato un messaggio di successo utilizzando la funzione **sub\_40105F**;
* Alla fine, il programma restituisce un valore di uscita. Se la connessione è stata verificata con successo, viene impostato il valore di ritorno su **1**; altrimenti, il flusso di esecuzione salta direttamente a **loc\_40102B**, che potrebbe essere una sezione per gestire casi in cui la connessione non è disponibile.

1. -----------------------------------------------------------------------------------------------------------------------------------------------------------|

Seguendo le richieste della traccia bonus, analizziamo ogni riga singolarmente:

1. **push ebp**: Questa istruzione mette il valore corrente del registro base del puntatore (EBP) nello stack.
2. **mov ebp, esp**: Questa istruzione copia il valore corrente dello stack pointer (**esp**) nel registro di base (**ebp**).
3. **push ecx**: Mette il valore corrente del registro ECX (uno dei registri generali, utilizzato per contare cicli e per altre operazioni) nello stack.
4. **push 0**: Mette un valore di 0 nello stack. Questo valore potrebbe essere usato come parametro per una chiamata di funzione successiva.
5. **push 0**: Mette un altro valore di 0 nello stack.
6. **call ds:InternetGetConnectedState**: Questa è una chiamata di funzione che utilizza il valore di un'etichetta globale denominata **InternetGetConnectedState** dalla sezione dei dati (DS). Questa funzione sembra essere utilizzata per verificare lo stato della connessione Internet.
7. **mov [ebp+var\_4], eax** : Sposta il valore di ritorno della chiamata di funzione (contenuto in EAX) nella variabile locale **[ebp+var\_4]**.
8. **cmp [ebp+var\_4], 0**: Compara il valore della variabile **[ebp+var\_4]** con 0.
9. **jz short loc\_40102B**: Salta all'etichetta **loc\_40102B** se il risultato della comparazione precedente è zero, il che indica che non c'è connessione Internet.
10. **push offset aSuccessInterne**: Mette l'indirizzo di una stringa nella pila. Questa stringa sembra essere un messaggio di successo.
11. **call sub\_40105F**: Chiama una subroutine chiamata **sub\_40105F**, che sembra essere responsabile della gestione del messaggio di successo.
12. **add esp, 4**: Aggiunge 4 byte all'indicatore dello stack per "ripulire" i parametri dalla pila.
13. **mov eax, 1**: Mette il valore 1 nel registro EAX.
14. **jmp short loc\_40103A**: Salta all'etichetta **loc\_40103A**.
15. **/ 16. loc\_40102B**: Le righe che iniziano con ";", invece, sono solitamente commenti nel codice assembly che forniscono informazioni aggiuntive o annotazioni per facilitare la comprensione del codice.