

PROGETTO S10-L5

PARTECIPANTI:
GABRIELE ARCELLI,
STEFANO CESARONI &
VALERIO ZAMPONE

CONSEGNA

L'esercizio settimanale ci chiede di analizzare il malware denominato
"Malware_U3_W2_L5"

In particolare ci viene chiesto di:

- analizzare le librerie importate dal malware
- analizzare le sezioni di cui si compone il malware

In secondo luogo, ci viene chiesto di:

- identificare i costrutti nella slide 3
 - ipotizzare il comportamento della funzionalità implementata
- 

ANALISI LIBRERIE

Di seguito vediamo quali sono le librerie importate dal malware e quali sono i comportamenti ad esse associati

Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

Utilizzando il programma “**CFF explorer**” e spostandoci nella sezione “**import directory**”, possiamo controllare le librerie importate dal file eseguibile.

ANALISI LIBRERIE

Inoltre, cliccando sulla libreria **WININET.dll** otteniamo maggiori informazioni sulle funzionalità associate alla libreria

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

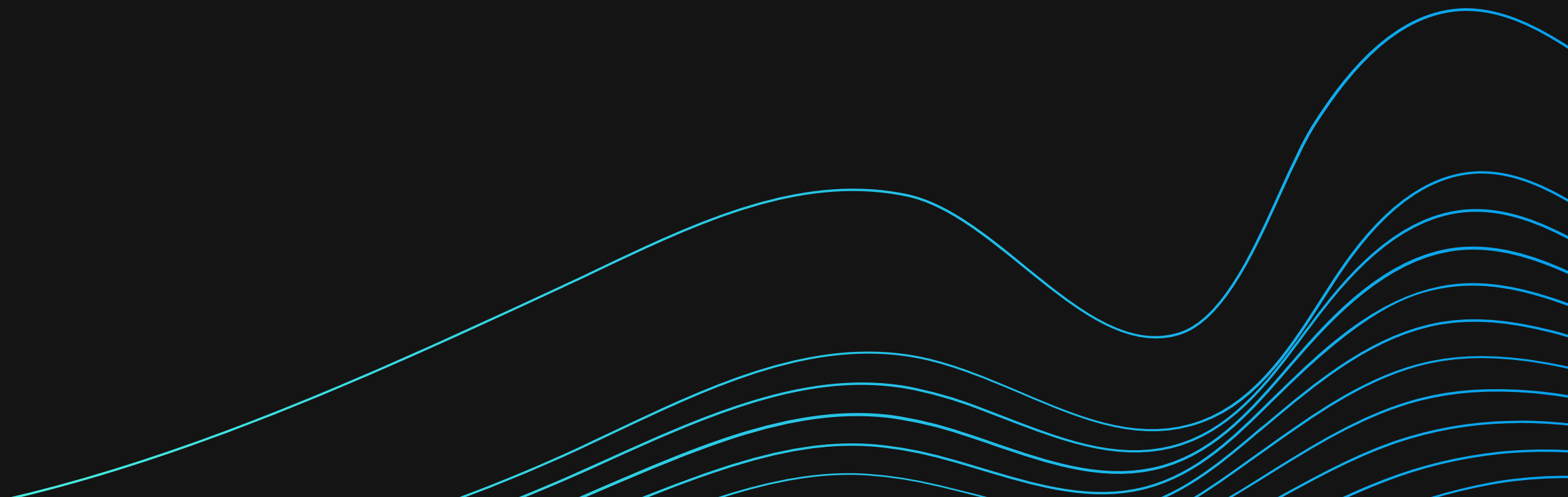
ANALISI LIBRERIE

Possiamo osservare che sono presenti:

- InternetOpenURLA:** apre un determinato link (il quale potrebbe scaricare file dannosi da internet)
- InternetCloseHandle:** va a chiudere determinati processi; spesso questa funzione può essere utilizzata in maniera malevola per terminare processi importanti per il sistema operativo, far crashare il sistema, ottenere privilegi da amministratore e molto altro

ANALISI LIBRERIE

-**InternetOpenA:** avvia una connessione ad internet, con la potenzialità di scaricare contenuto malevolo o ricevere comandi da una shell remota evitando facilmente i controlli di sicurezza, come il firewall.



FUNZIONI LIBRERIE

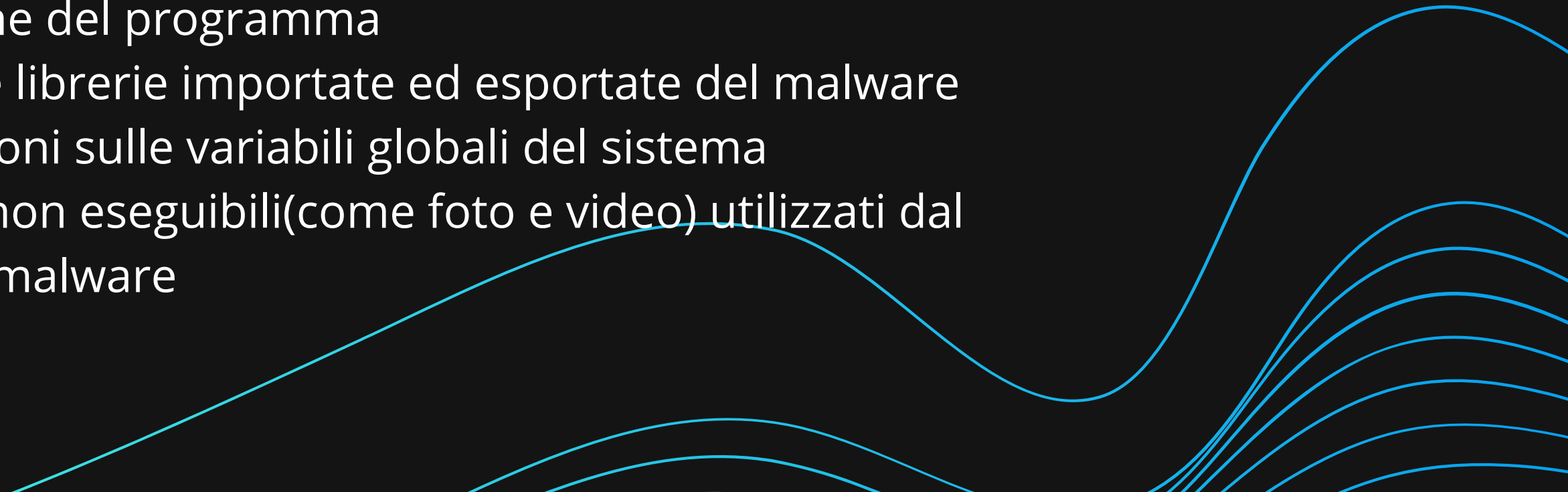
Le librerie importate dal programma svolgono le seguenti funzioni:

- Kernel32.dll**: interagisce con il sistema operativo e gestisce (anche impropriamente) la memoria e le risorse del sistema operativo.
- Wininet.dll**: permette alle applicazioni di interagire con protocolli come HTTP, HTTPS ed FTP.

SEZIONI DEL MALWARE

Di seguito vediamo quali sono le sezioni che compongono il malware. Si tratta di un passaggio molto importante, poiché funzionale a trovare ulteriori istruzioni che il malware esegue.

Generalmente, si passa ad analizzare le sezioni del malware per rilevare file di comportamento del programma; in particolare è bene confutare la presenza dei seguenti file:

- “.text”**: contiene informazioni sulla parte di memoria che verrà usata durante l'esecuzione del programma
 - “.rdata”**: contiene informazioni sulle librerie importate ed esportate del malware
 - “.data”**: contiene informazioni sulle variabili globali del sistema
 - “.rsrc”**: include informazioni sui file non eseguibili(come foto e video) utilizzati dal malware
- 

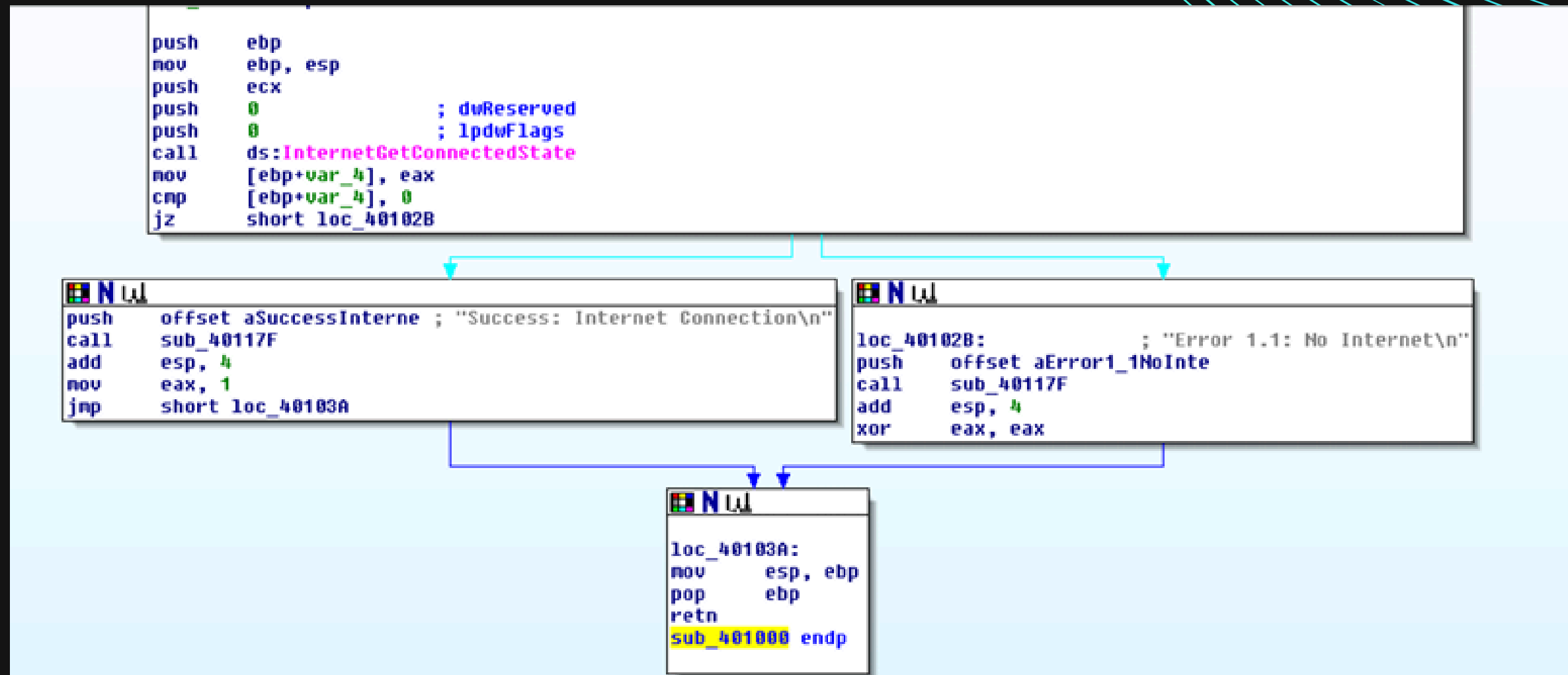
SEZIONI DEL MALWARE

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

Come possiamo osservare, all'interno del malware sono presenti le sezioni ".text", ".rdata" e ".data". Sebbene questi siano file comunemente associati ai file eseguibili (e non necessariamente malevoli) è sempre bene analizzarli per comprendere meglio il comportamento del file eseguibile.

ANALISI COSTRUTTI





Questo codice verifica se c'è una connessione Internet utilizzando la funzione **"InternetGetConnectedState"**. Se è presente, stampa **"Success: Internet Connection"** e restituisce 1. Se non è presente, stampa **"Error 1.1: No Internet"** e restituisce 0. In entrambi i casi, pulisce lo stack e ritorna correttamente.

Le istruzioni evidenziate in **blu** costituiscono il primo costrutto, ovvero la creazione dello stack, una struttura dati LIFO (Last In, First Out) utilizzata per memorizzare dati temporanei, come variabili locali e indirizzi di ritorno.

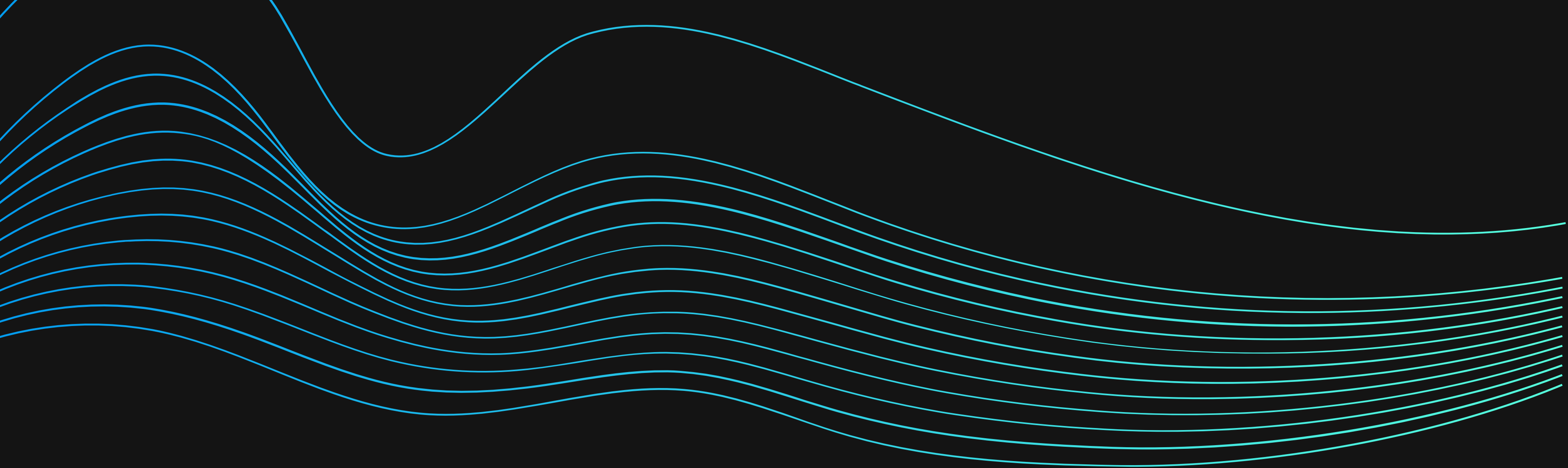
La sequenza di istruzioni prepara un nuovo frame di stack utilizzato per gestire le variabili locali della funzione.

Le istruzioni evidenziate in **rosso** identificano invece un costrutto condizionale, simile ad un'istruzione 'if' in linguaggi di alto livello come C o C++. In sintesi, implementa una condizione if che verifica se [ebp + var_4] è uguale a 0 e, se lo è, esegue un salto condizionale al blocco di codice designato per gestire il caso in cui non c'è connessione a Internet.

Le istruzioni evidenziate in **verde** indicano un costrutto usato alla fine di una funzione per ripristinare lo stato dello stack e del base pointer, preparandosi a ritornare alla funzione chiamante.

```
push    ebp
mov     ebp, esp
push    ecx
push    0           ; dwReserved
push    0           ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

```
NW
loc 40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

Blocco 1: Inizio della Funzione

- **push ebp**: Salva il valore corrente di ebp sullo stack.
Dopo l'esecuzione di questa istruzione, il puntatore dello stack (esp) viene decrementato, puntando alla nuova cima dello stack, che ora contiene il vecchio valore di ebp.
- **mov ebp, esp**: copia il valore corrente del registro esp nel registro ebp.
Ora ebp punta all'indirizzo di memoria che rappresenta l'inizio del frame di stack corrente.
- **push ecx**: Salva ecx sullo stack.
- **push 0**: Passa il valore 0 come argomento alla funzione InternetGetConnectedState.
- **call ds**: Chiama la funzione di Windows InternetGetConnectedState per verificare lo stato della connessione Internet.
- **mov [ebp+var_4], eax**: Memorizza il valore di ritorno della funzione in una variabile locale.
- **cmp [ebp+var_4], 0**: Confronta il valore di ritorno con 0.
- **jz short loc_40102B**: Se il valore è 0 (nessuna connessione Internet), salta all'etichetta loc_40102B. Se non è 0, il flusso del programma continua con il blocco successivo (che gestisce il successo della connessione a Internet).

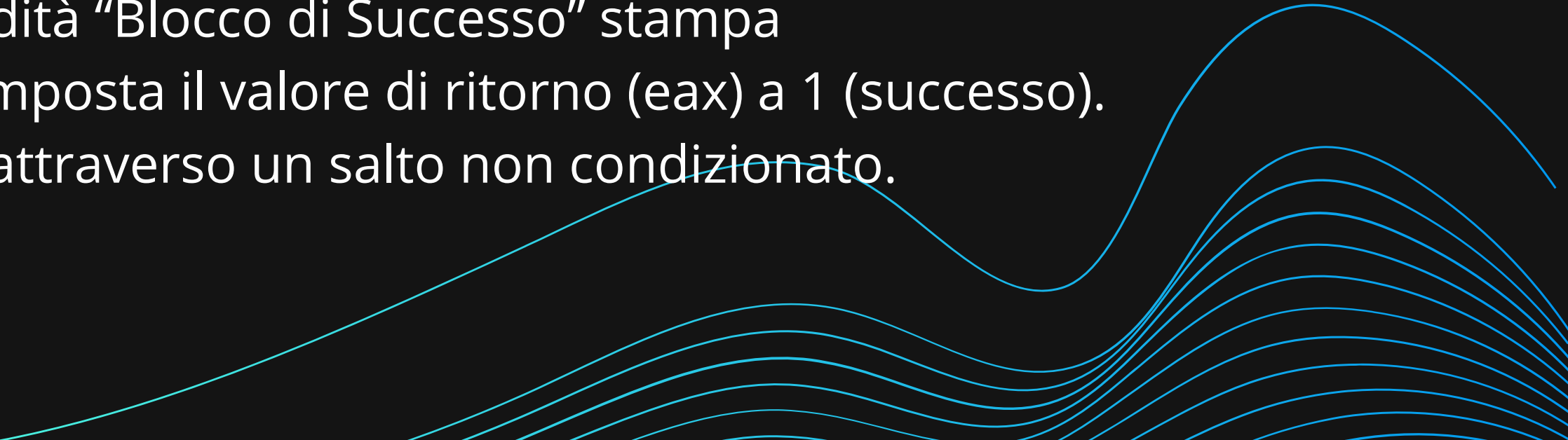
Riassumendo, Il blocco Iniziale Imposta il frame di stack, salva i registri e chiama InternetGetConnectedState per verificare la connessione a Internet. Se la connessione è presente salta al blocco 2, altrimenti passa al blocco 3.

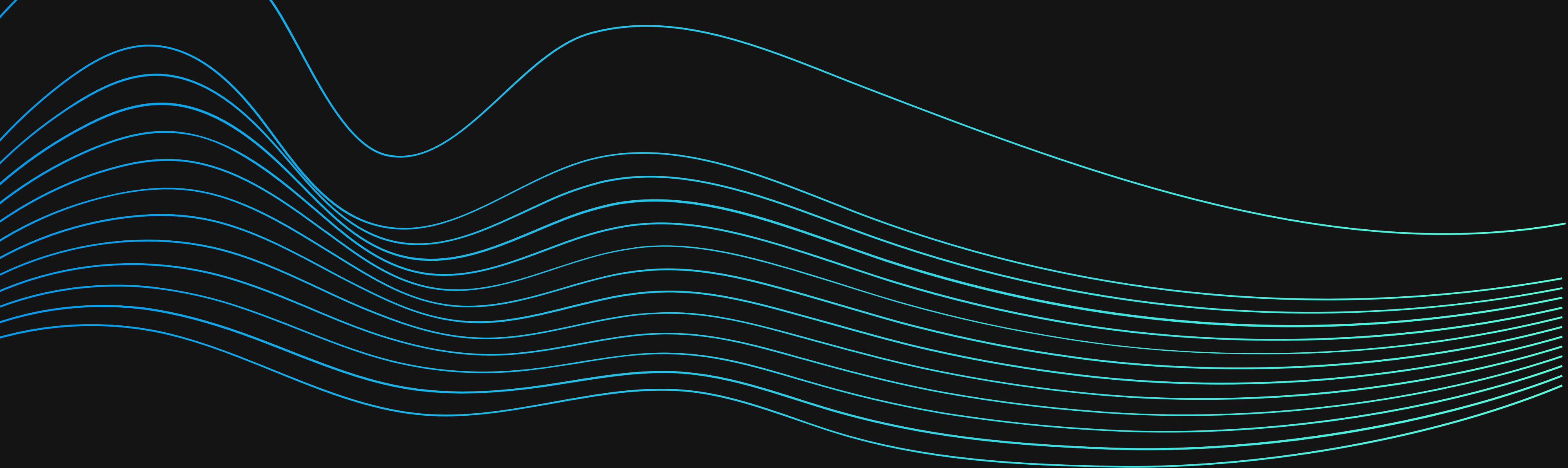
An abstract graphic consisting of numerous thin, wavy lines in a light blue or cyan color. These lines flow from the left side of the frame towards the right, creating a sense of movement and depth. The lines vary in frequency and amplitude, with some having more pronounced peaks and valleys than others. They are set against a solid black background.

Blocco 2: Successo Connessione Internet

- **push offset aSuccessInterne**: Passa l'indirizzo della stringa "Success: Internet Connection\n" come argomento.
- **call sub_40117F**: Chiama una funzione di supporto (probabilmente per stampare la stringa).
- **add esp, 4**: Ripristina lo stack rimuovendo l'argomento.
- **mov eax, 1**: Imposta eax a 1 (indicando successo).
- **jmp short loc_40103A**: Salta alla fine della funzione.

Questo blocco, denominato per comodità "Blocco di Successo" stampa "Success: Internet Connection\n" ed imposta il valore di ritorno (eax) a 1 (successo). Infine, salterà alla fine della funzione attraverso un salto non condizionato.

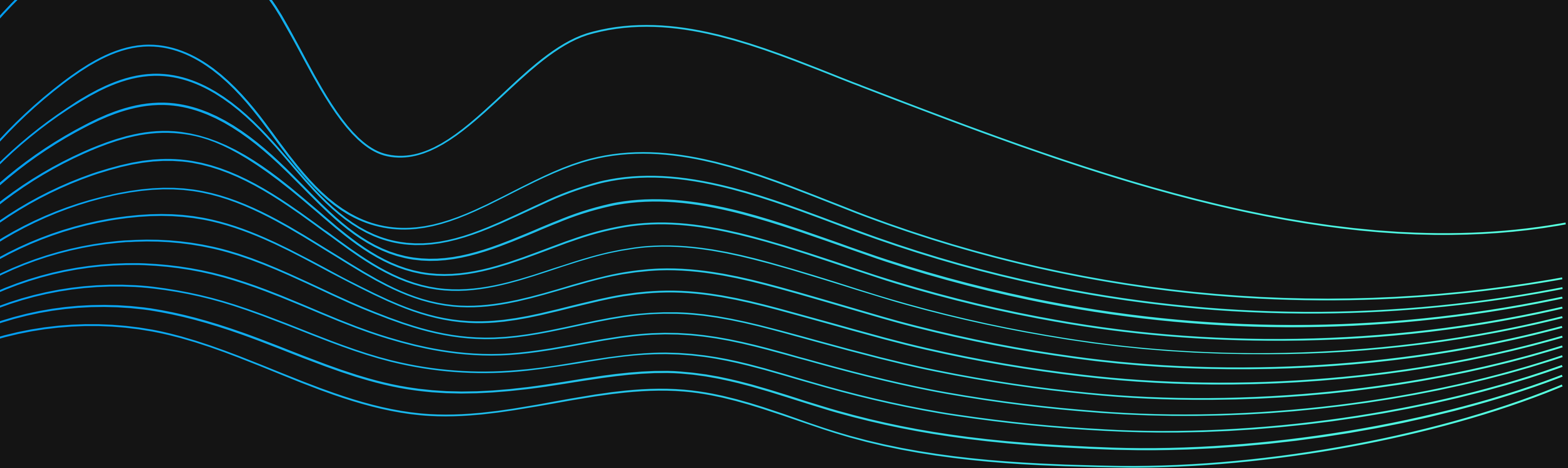




Blocco 3: Errore di Connessione

- **loc_40102B**: Etichetta che indica l'inizio della sezione di errore.
- **push offset aError1_1NoInte**: Passa l'indirizzo della stringa "Error 1.1: No Internet\n" come argomento.
- **call sub_40117F**: Chiama una funzione di supporto (probabilmente per stampare la stringa).
- **add esp, 4**: Ripristina lo stack rimuovendo l'argomento.
- **xor eax, eax**: Questa istruzione è utilizzata per azzerare il registro eax in modo efficiente eseguendo un'operazione XOR bit a bit tra eax e sé stesso. Questo imposta tutti i bit del registro a 0, risultando in $eax = 0$. È preferita per la sua velocità e la sua dimensione ridotta del codice rispetto ad altre istruzioni equivalenti.

Il "Blocco di Errore", se non c'è connessione a Internet, stamperà "Error 1.1: No Internet\n" impostando il valore di ritorno (eax) a 0 (fallimento).



Blocco 4: Fine della Funzione

- **loc_40103A:** Etichetta che indica l'inizio della fine della funzione.
- **mov esp, ebp:** copia il valore del registro ebp nel registro esp.
Il registro esp (stack pointer) ora punta all'inizio del frame di stack corrente, che è il valore che ebp sta tenendo.
Questo ripristina il puntatore dello stack al valore che aveva quando il frame di stack è stato creato, essenzialmente "pulendo" qualsiasi spazio allocato per variabili locali.
- **pop ebp:** recupera il valore in cima allo stack e lo mette nel registro ebp, ripristinando il base pointer al valore che aveva prima che la funzione fosse chiamata.
- **retn:** Ritorna dalla funzione.

Il “Blocco Finale” ripristina lo stack e ritorna alla funzione chiamante.





**GRAZIE PER
L'ATTENZIONE**