S11/L5
PROGETTO

GIULIA FIACCHI

TRACCIA

Con riferimento al codice presente nelle slide successive, rispondere ai seguenti quesiti:

1. Spiegate, motivando, quale **salto condizionale** effettua il Malware.
2. Disegnare un **diagramma di flusso** (prendete come esempio la visualizzazione grafica di IDA) identificando i salti condizionali (sia quelli effettuati che quelli non effettuati). Indicate con una linea verde i salti effettuati, mentre con una linea rossa i salti non effettuati.
3. Quali sono le diverse **funzionalità implementate** all'interno del Malware?
4. Con riferimento alle istruzioni «call» presenti in tabella 2 e 3, dettagliare come sono passati gli argomenti alle successive chiamate di funzione . Aggiungere eventuali dettagli tecnici/teorici.

TABELLA 1

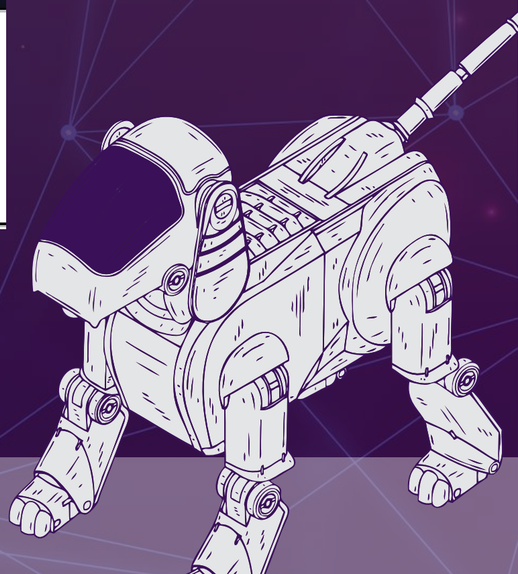
Locazione	Istruzione	Operandi	Note
00401040	mov	EAX, 5	
00401044	mov	EBX, 10	
00401048	cmp	EAX, 5	
0040105B	jnz	loc 0040BBA0	; tabella 2
0040105F	inc	EBX	
00401064	cmp	EBX, 11	
00401068	jz	loc 0040FFA0	; tabella 3

TABELLA 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile ()	; pseudo funzione

TABELLA 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop \Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione



PARTE 1

SALTO CONDIZIONALE

Come possiamo notare nella tabella 1 sono presenti due jump condizionali (che servono solitamente a trasferire il controllo del programma a un'altra parte del codice solo se una certa condizione è vera).

```
CMP EAX, 5
JNZ 0040BBA0  ;TABELLA 2
```

L'istruzione jnz (Jump if Not Zero) verrà eseguita se il confronto precedente (cmp EAX, 5) non risulta zero, ovvero se EAX non è uguale a 5.

Poiché EAX è impostato a 5 nell'istruzione mov precedente, il confronto cmp EAX, 5 restituisce zero, quindi il salto jnz ***non viene eseguito***

```
CMP EBX, 11
JZ 0040FFA0  ;TABELLA 3
```

L'istruzione jz (Jump if Zero) verrà eseguita se il confronto precedente (cmp EBX, 11) restituisce zero, ovvero se EBX è uguale a 11.

Dopo aver incrementato EBX di uno, EBX diventa 11, quindi il confronto cmp EBX, 11 restituisce zero, e il salto jz **viene eseguito**.

PARTE 2

DIAGRAMMA DI FLUSSO

TABELLA 1

Locazione	Istruzione	Operandi	Note
00401040	mov	EAX, 5	
00401044	mov	EBX, 10	
00401048	cmp	EAX, 5	
0040105B	jnz	loc 0040BBA0	; tabella 2
0040105F	inc	EBX	
00401064	cmp	EBX, 11	
00401068	jz	loc 0040FFA0	; tabella 3

TABELLA 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile ()	; pseudo funzione

TABELLA 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop \Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

PARTE 3

FUNZIONALITÀ IMPLEMENTATE

TABELLA 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile ()	; pseudo funzione

1 MOV EAX, EDI

L' istruzione mov copia il contenuto del registro EDI nel registro EAX.
E il valore di EDI è l'URL (***www.malwaredownload.com***), quindi questa istruzione sta trasferendo l'URL nel registro EAX.

2 PUSH EAX

L'istruzione push inserisce il contenuto del registro EAX nello stack.
Questa istruzione sta **spingendo l'URL** (che ora è in EAX) **sullo stack**, preparandolo come parametro per una funzione.

3 CALL DOWNLOADTOFILE()

FUNZIONALITÀ

L'istruzione call viene utilizzata per chiamare una funzione. In questo caso, viene chiamata una pseudo funzione denominata DownloadToFile() che, prende l'URL dallo stack (spinto dall'istruzione precedente) e avvia il download del file dal sito specificato.

Il codice nella Tabella 2 è progettato per scaricare un file da un URL specificato. Il valore di EDI, contenente l'URL, viene trasferito nel registro EAX e poi passato alla funzione DownloadToFile(), che gestisce il download del file.

PARTE 3

FUNZIONALITÀ IMPLEMENTATE

TABELLA 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop \Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

1 MOV EAX, EDI

L' istruzione mov copia il contenuto del registro EDI nel registro EAX.
EDI contiene il percorso di un file eseguibile (**C:\Program and Settings\Local User\Desktop\Ransomware.exe**), quindi questa istruzione trasferisce il percorso del file in EDX.

2 PUSH EAX

L'istruzione push inserisce il contenuto del registro EAX nello stack.
Questa istruzione spinge il percorso del file eseguibile sullo stack, preparandolo come parametro per una funzione.

3 CALL WINEXEC()

FUNZIONALITÀ

L'istruzione call viene utilizzata per chiamare una funzione. In questo caso, viene chiamata una funzione denominata WinExec().
Questa pseudo funzione esegue il file specificato dal percorso che è stato spinto nello stack (ovvero Ransomware.exe).

Il codice nella Tabella 3 è progettato per eseguire un file eseguibile. Il percorso completo del file, che è contenuto in EDI, viene trasferito nel registro EDX e poi passato alla funzione WinExec(), che lo esegue.

PARTE 4

ISTRUZIONI «CALL»

CALL DOWNLOADTOFILE()

In linguaggio assembly, una chiamata di funzione come **DownloadToFile()** non esiste direttamente come istruzione infatti possiamo osservare che viene identificata nel codice come "*pseudo funzione*", perché rappresenta una funzione specifica all'interno di un programma o una libreria. Tuttavia, se vogliamo rappresentare la chiamata a una funzione che potrebbe essere simile a DownloadToFile() in assembly, possiamo immaginarla come un'etichetta o un indirizzo di memoria a cui il processore salta per eseguire il codice della funzione.

Questa funzione, in riferimento al codice, è progettata per scaricare un file da un URL specifico e salvare il file localmente sul sistema.

Le operazioni generali:

1. L'**URL viene spostato dal registro EDI al registro EAX**, e successivamente EAX viene pushato nello stack. Questo prepara l'argomento per la funzione DownloadToFile().
2. L'istruzione call DownloadToFile() trasferisce il controllo alla funzione, **passando l'URL come argomento**.
3. DownloadToFile() esegue il download del file dall'URL specificato e lo salva su disco.

Questo tipo di codice è tipicamente utilizzato in applicazioni che devono scaricare file da internet, e l'uso di registri e stack è una parte fondamentale della gestione degli argomenti e delle chiamate di funzione in assembly.

UTILIZZARE UNA PSEUDO FUNZIONE CONSENTE DI DESCRIVERE L'INTENTO O L'OPERAZIONE A UN LIVELLO PIÙ ALTO, SEMPLIFICANDO L'ANALISI DEL CODICE E RENDENDO CHIARO COSA FA IL PROGRAMMA SENZA DOVER ESAMINARE OGNI SINGOLO DETTAGLIO TECNICO.

PARTE 4

ISTRUZIONI «CALL»

CALL WINEXEC()

In linguaggio assembly, una chiamata di funzione come **WinExec()** non esiste direttamente come istruzione infatti possiamo osservare che anche questa viene identificata nel codice come "*pseudo funzione"* perché , in assembly, non esiste un singolo comando che esegue queste operazioni, ma il termine viene utilizzato per rappresentare un concetto astratto o un blocco di codice che, combinato, esegue l'operazione di download. Tuttavia, se vogliamo rappresentare la chiamata a una funzione che potrebbe essere simile a DownloadToFile() in assembly, possiamo immaginarla come un'etichetta o un indirizzo di memoria a cui il processore salta per eseguire il codice della funzione.

Questa funzione, in riferimento al codice, è progettata per eseguire un comando o avviare un programma in ambiente Windows. Questo è il suo scopo principale e viene utilizzata per eseguire applicazioni o comandi come se fossero eseguiti dalla riga di comando

Le operazioni generali:

1. La prima riga di codice indica che il registro EDI contiene l'indirizzo della stringa che rappresenta il percorso del file eseguibile.
2. La **stringa** che rappresenta il comando o il **percorso dell'eseguibile** viene **passata a WinExec()** come argomento (precedentemente è stata passata nello stack).
3. Quando **WinExec()** viene **chiamato,** Windows usa l'argomento per trovare e avviare l'applicazione specificata. Se l'argomento è un percorso a un file eseguibile, il sistema operativo eseguirà quel file.

Solitamente se la funzione ha esito positivo, il valore restituito è maggiore di 31.
Se la funzione ha esito negativo, il valore restituito è uno dei valori di errore seguenti.

QUESTA FUNZIONE VIENE FORNITA SOLO PER LA COMPATIBILITÀ CON WINDOWS A 16 BIT. LE APPLICAZIONI DEVONO USARE LA FUNZIONE CREATEPROCESS

BONUS

Analizzare il file C: \Users\user\Desktop \Software Malware analysis\SysinternalsSuite \Tcpvcon.exe con IDA Pro

Analizzare SOLO la "funzione corrente" una volta aperto IDA

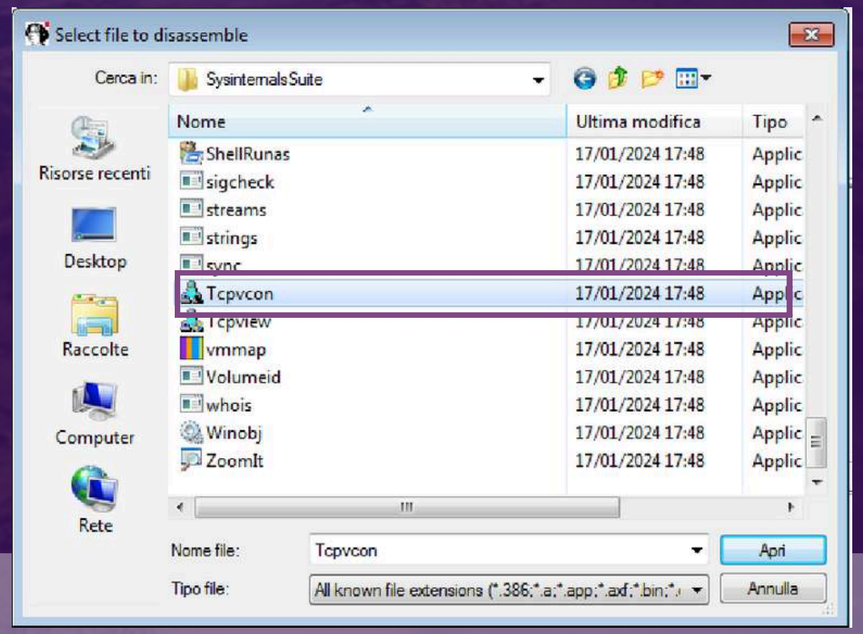
La funzione corrente la visualizzo con il tasto F12 oppure con il tasto blu indicato nella slide successiva.
Se necessario, reperire altre informazioni con OllyDBG oppure effettuando ulteriori analisi con IDA (o altri software).

Mi interessa soltanto il significato/funzionamento/senso di questa parte di codice visualizzato alla pagina successiva.

BONUS

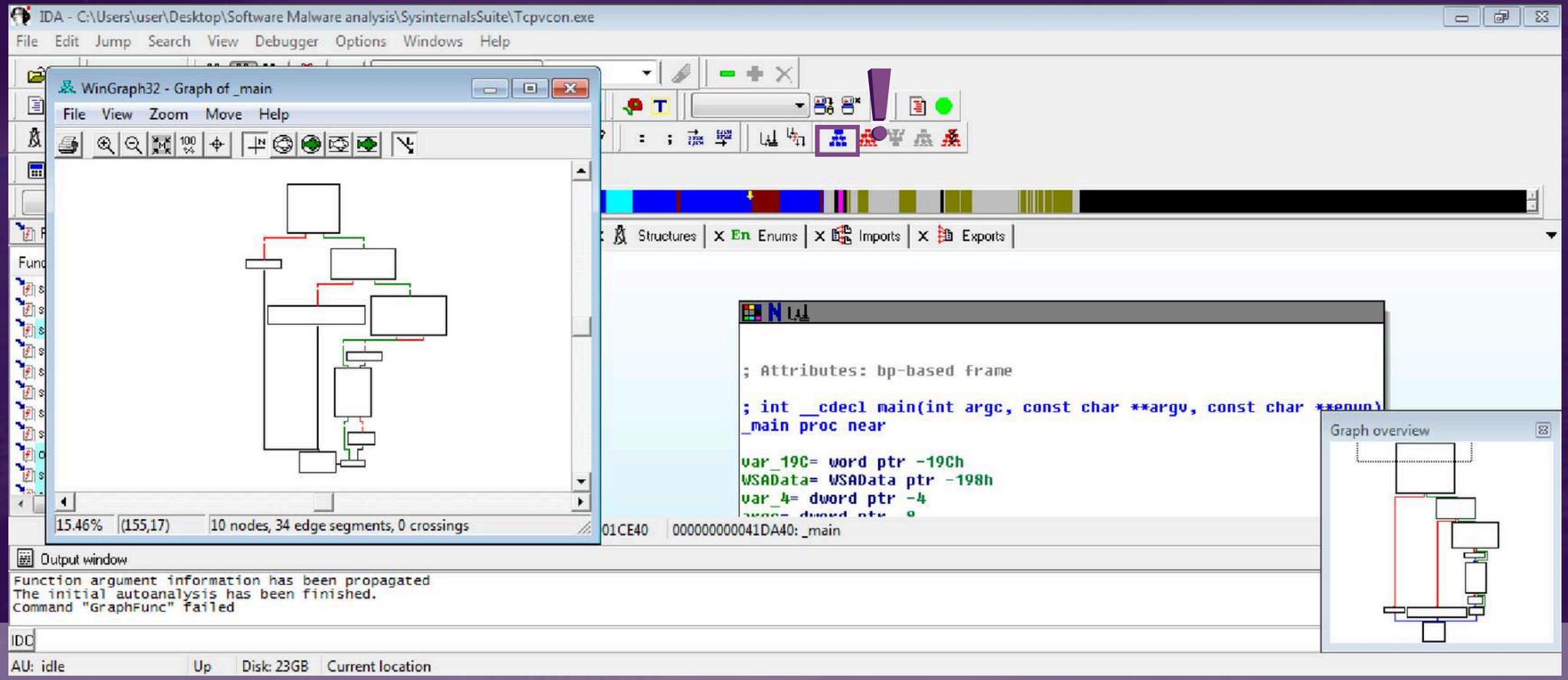
PRIMA DI ESAMINARE IL FILE, ANCHE SE NON SAPPIAMO CHE SI TRATTI EFFETTIVAMENT DI UN FILE MALEVOLO E CHE ANDREMO AD UTILIZZARE IDA PRO E QUINDI AD EFFETTUARE UN'ANALISI STATICA AVANZATA PERCIÒ IL FILE NON VERRÀ ESEGUITO, È CONSIGLIABILE COMUNQUE METTERE IN **SICUREZZA L'AMBIENTE** ANDANDO A CONTROLLARE E AD ASSICURARCI CHE LA RETE SIA IMPOSTATA SU *RETE INTERNA*, ANCORA CHE NON CI SIA *NESSUNA CARTELLA CONDIVISA* (PER NON PERMETTERE LA FUORISCITA DI FILE MALEVOLI) E CHE LE **PORTE USB** SIANO TUTTE **DISABILITATE** E INFINE LA **CREAZIONI DI ISTANTANEE.**

Iniziamo avviando la nostra macchina Windows e seguendo il percorso specificato nella traccia, andando ad aprire il file con IDA pro.

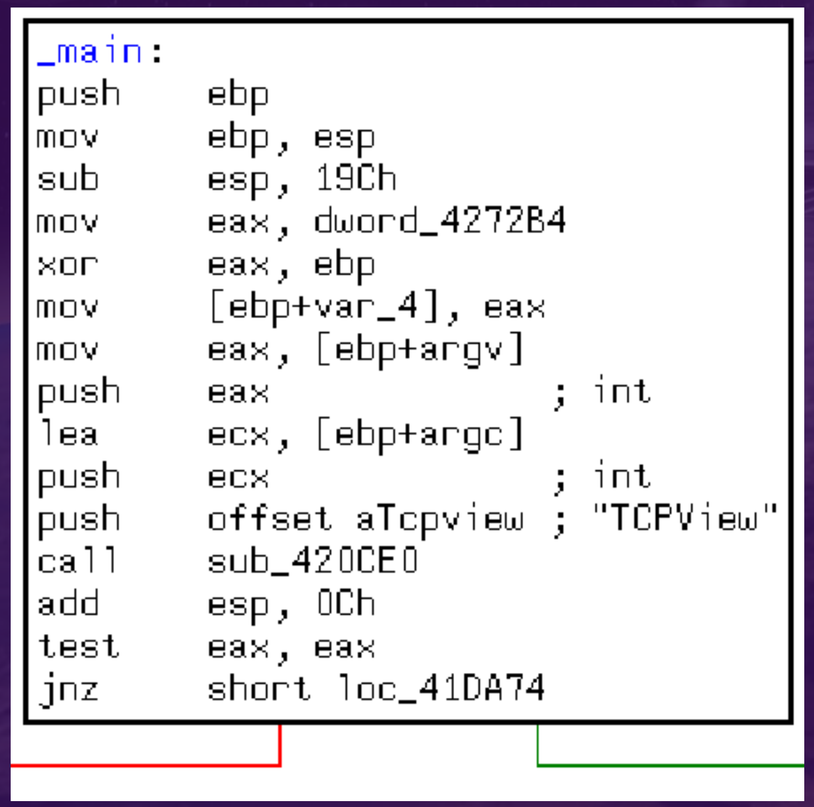


BONUS

Una volta aperto clicchiamo su tasto indicato nell'immagine o premiamo F12 per andare ad esaminare la funzione corrente.



BLOCCO 1



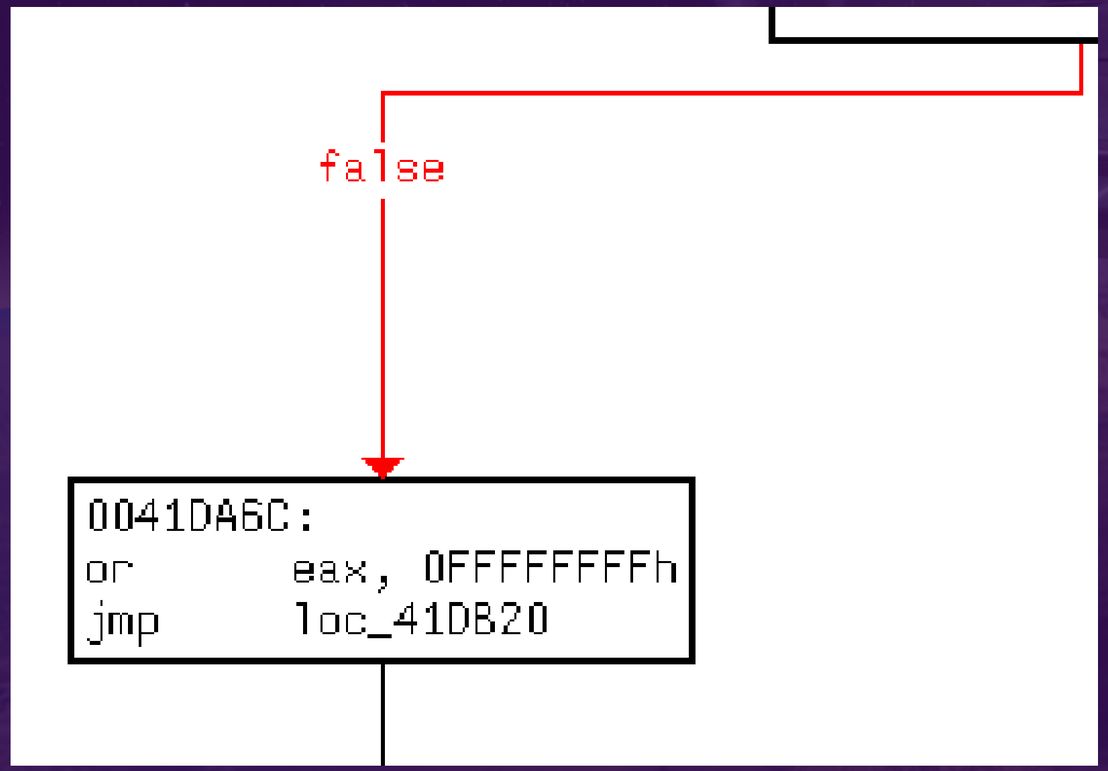
```
_main:
push    ebp
mov     ebp, esp
sub     esp, 19Ch
mov     eax, dword_4272B4
xor     eax, ebp
mov     [ebp+var_4], eax
mov     eax, [ebp+argv]
push    eax             ; int
lea     ecx, [ebp+argc]
push    ecx             ; int
push    offset aTcpview ; "TCPView"
call    sub_420CE0
add     esp, 0Ch
test    eax, eax
jnz     short loc_41DA74
```

Questo codice mostra una tipica sequenza di inizializzazione di un'applicazione in C/C++, in cui:

1. Viene creato il frame stack per le variabili locali.
2. Viene implementata una protezione del stack (con un possibile controllo di sicurezza con XOR).
3. Poi chiama una funzione (sub_420CE0) con tre parametri, che sembra avviare o configurare qualcosa legato all'applicazione.
4. Viene gestito il risultato della funzione chiamata per prendere ulteriori decisioni di controllo del flusso e quindi avviene un *jump condizionale* che se è diverso da zero, salta a una specifica posizione nel codice.

Il codice **push offset aTcpview** mette l'indirizzo della stringa "TCPView" (o meglio, il contenuto della variabile aTcpview) nello stack, che verrà utilizzato come argomento quando si chiama la funzione sub_420CE0.

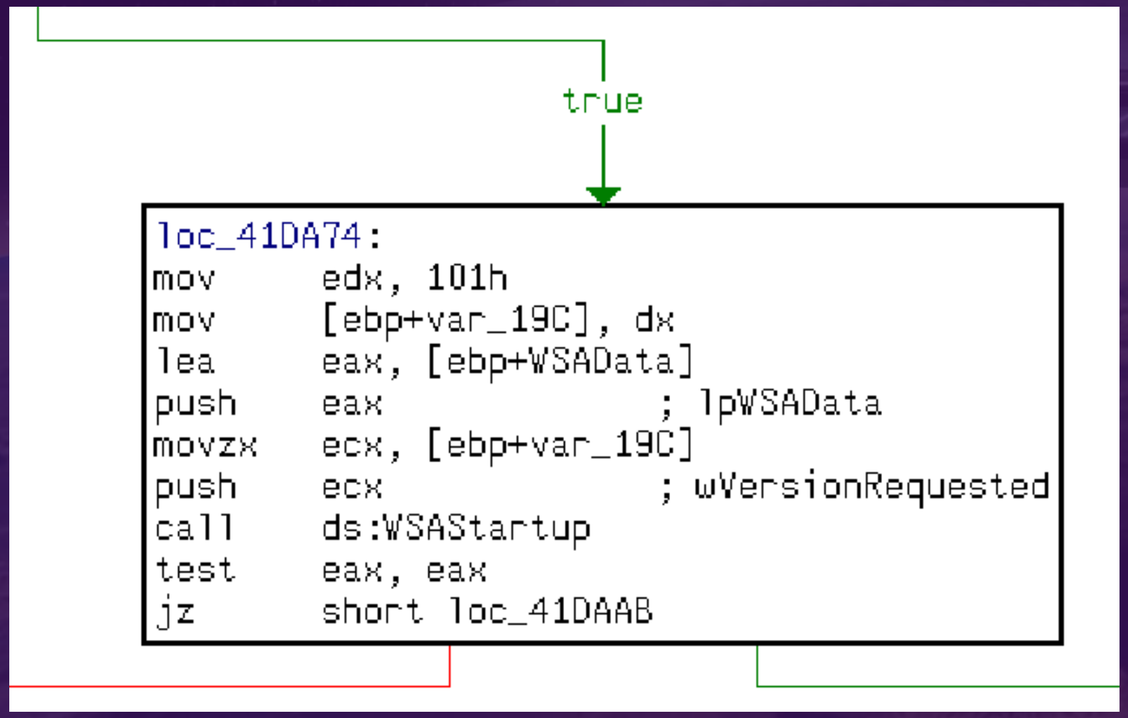
BLOCCO 2



Se il jump condizionale non avviene, avremo come risultato il false (perché non è diverso da 0).

Effettuando un **OR con 0xFFFFFFFF**, il risultato è che tutti i bit di eax vengono impostati a 1. In termini numerici, questo equivale a impostare il valore di eax a 0xFFFFFFFF, che in notazione con segno è -1.
Mentre l'istruzione **jmp** esegue un salto incondizionato all'indirizzo specificato da loc_41DB20 (non importa quale sia il contenuto di eax)

BLOCCO 3

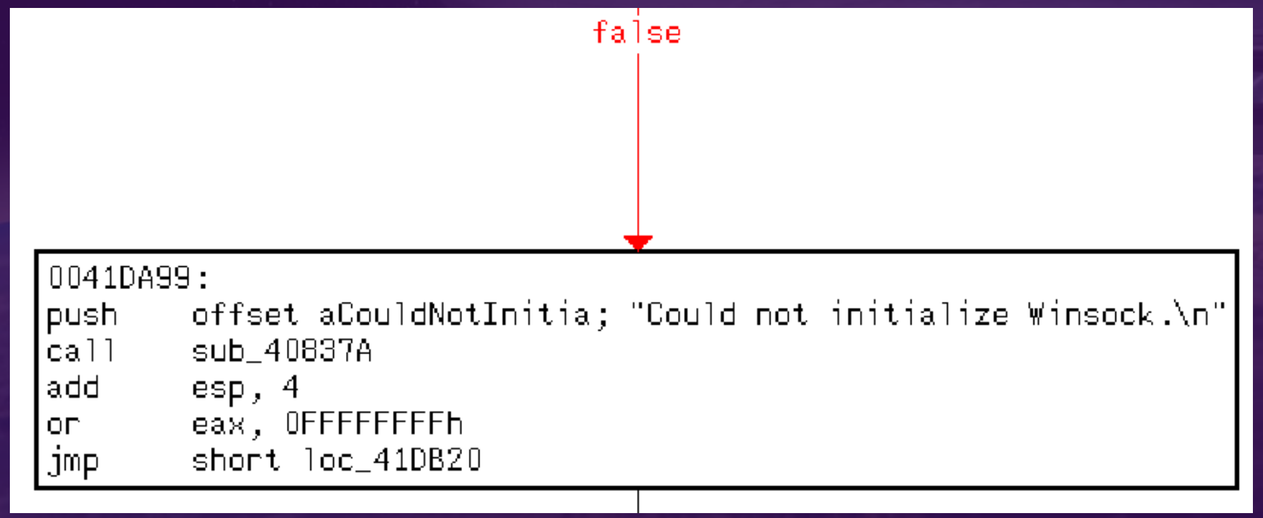


Se invece il jump previsto dal blocco 1 avrà valore diverso da 0, avverrà e come risultato darà true.

Facendo riferimento a loc possiamo dire che è come un segnalibro nel codice a cui il flusso di esecuzione può tornare.

Questo frammento di codice si occupa di inizializzare la libreria di rete Winsock per un'applicazione Windows e lo possiamo capire dall'uso di WSAData e WSAStartup. Il codice richiede la versione 1.1 di Winsock (0x0101) e, se l'inizializzazione è avvenuta correttamente, salta alla prossima parte del programma (etichetta loc_41DAAB). Se l'inizializzazione fallisce, il programma non esegue il salto e continuerà eseguendo altre istruzioni, che probabilmente gestiranno l'errore.

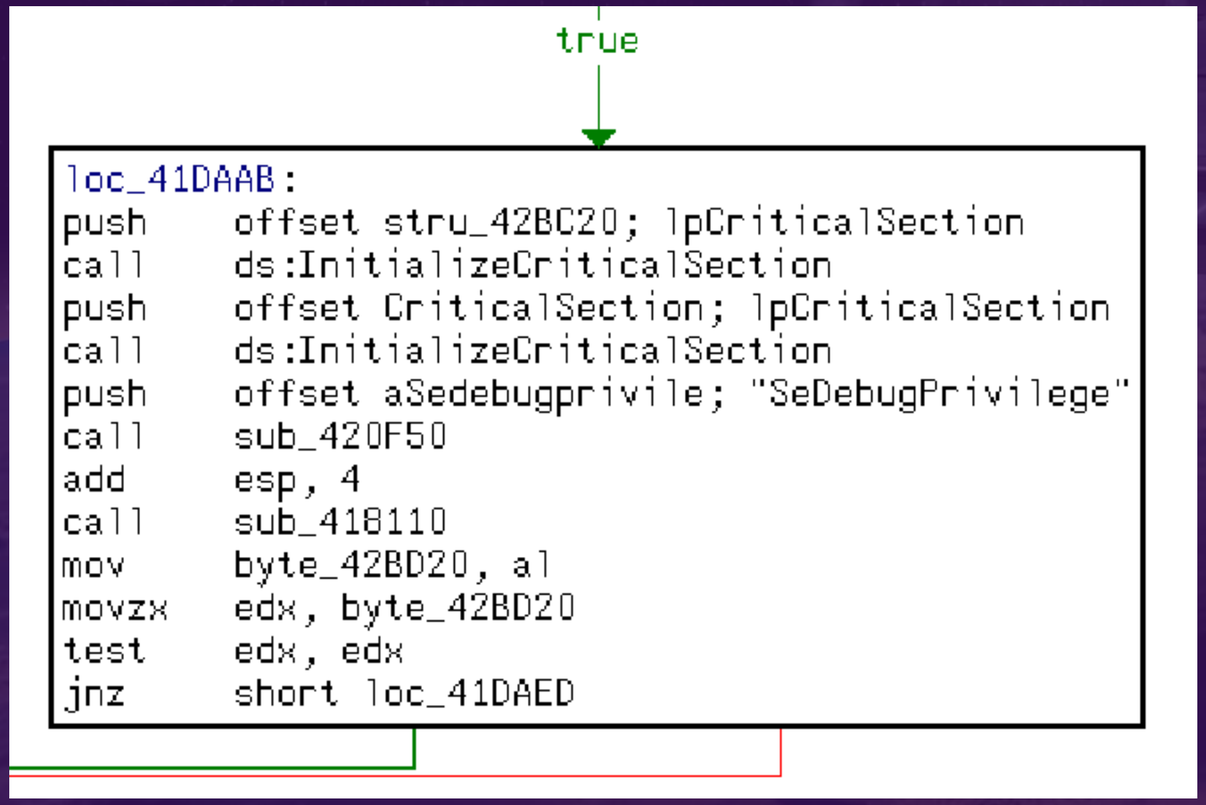
BLOCCO 4



Questo frammento di codice viene eseguito quando l'inizializzazione di Winsock (tramite WSAStartup) fallisce. La sequenza di istruzioni effettua le seguenti azioni:

1. Mostra un messaggio di errore all'utente o lo registra, usando la stringa "Could not initialize Winsock.\n".
2. Imposta il registro eax a -1, probabilmente per indicare al chiamante che si è verificato un errore critico.
3. Salta a una sezione del codice (loc_41DB20) che si occupa di gestire il fallimento dell'inizializzazione, probabilmente per terminare l'applicazione o eseguire altre azioni di pulizia.

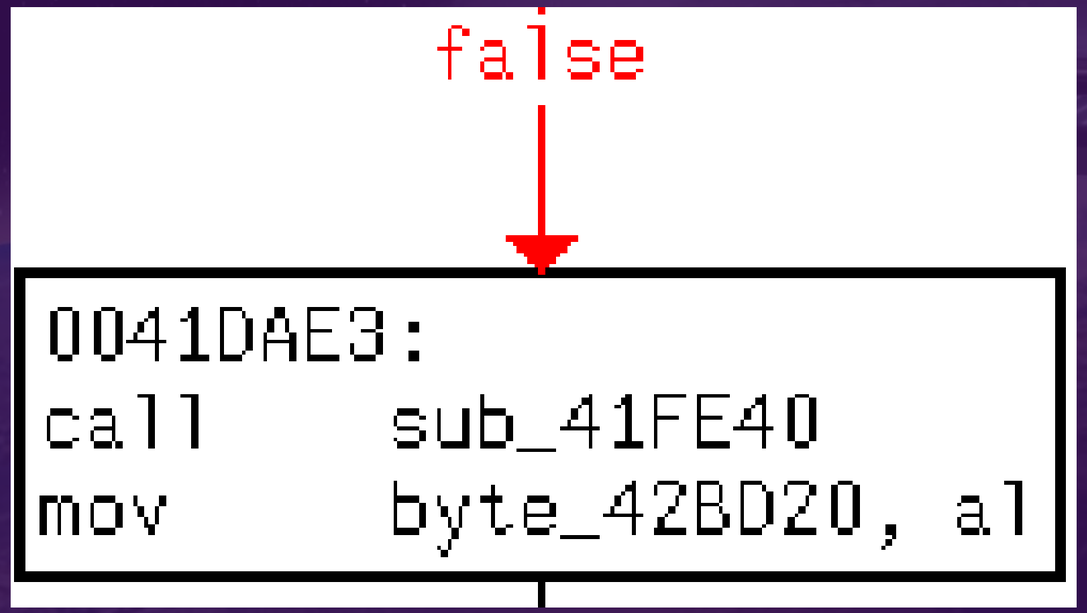
BLOCCO 5



Questo codice indica che il salto è avvenuto e sta inizializzando due sezioni critiche, eseguendo alcune impostazioni di privilegi e poi controllando il risultato di una chiamata a funzione per determinare il flusso di esecuzione successivo.

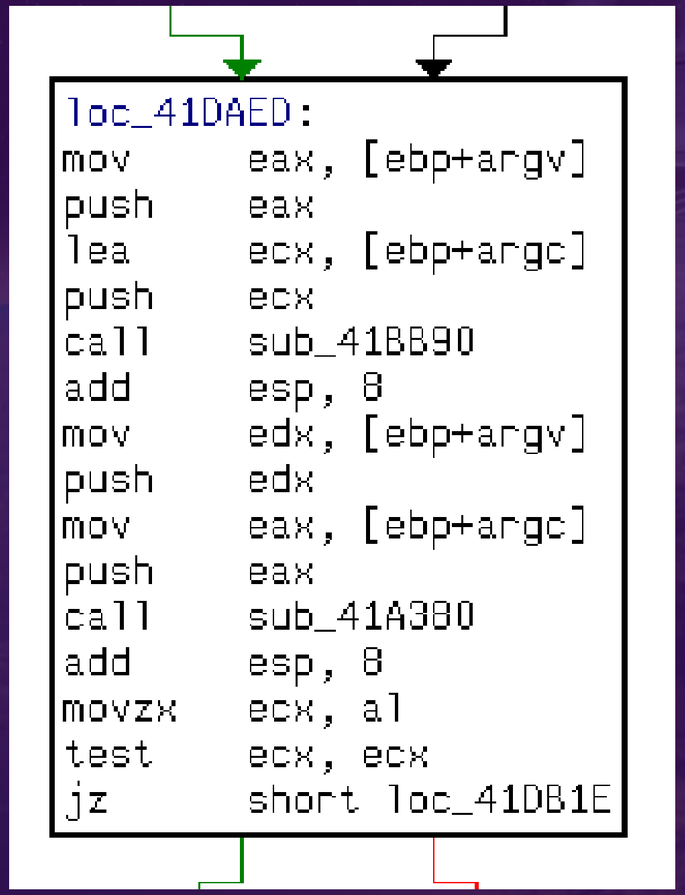
1. Chiama InitializeCriticalSection per inizializzare due oggetti di sezione critica (utilizzati per la sincronizzazione dei thread). Uno è identificato da stru_42BC20 e l'altro da CriticalSection.
2. Esegue una funzione (sub_420F50) utilizzando il nome del privilegio "SeDebugPrivilege". Questa funzione sembra gestire o impostare il privilegio richiesto.
3. Chiama un'altra funzione (sub_418110), memorizza il risultato in una variabile (byte_42BD20), e poi verifica se questo risultato è diverso da zero. Se il risultato è diverso da zero, il flusso di esecuzione salta a una specifica posizione di codice (loc_41DAED).

BLOCCO 6



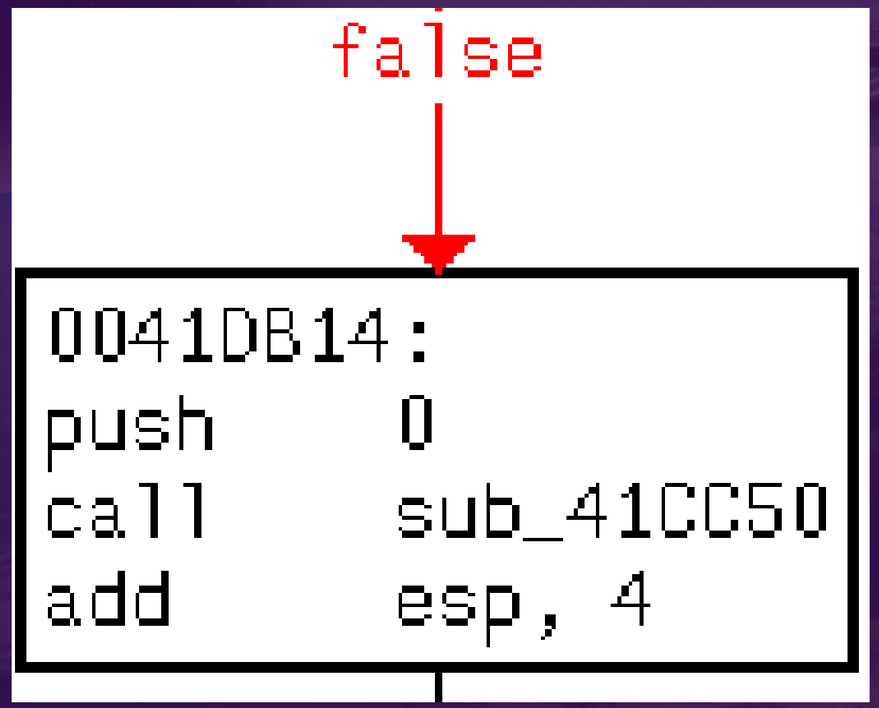
Se il salto non è avvenuto perchè non diverso da 0 come indicato nel blocco 5 (jnz), il codice esegue una chiamata a sub_41FE40 e poi memorizza il valore restituito da questa funzione (che si trova nel registro al) in byte_42BD20. Questo valore è probabilmente un risultato o uno stato che deve essere salvato e utilizzato successivamente.

BLOCCO 7



Il codice chiama due funzioni (sub_41BB90 e sub_41A380), passando a entrambe lo stesso insieme di argomenti (argv e argc). Dopo aver effettuato queste chiamate, verifica il risultato della seconda funzione (sub_41A380). Se il risultato è zero, il flusso di esecuzione salta a un'altra parte del codice (loc_41DB1E).

BLOCCO 8

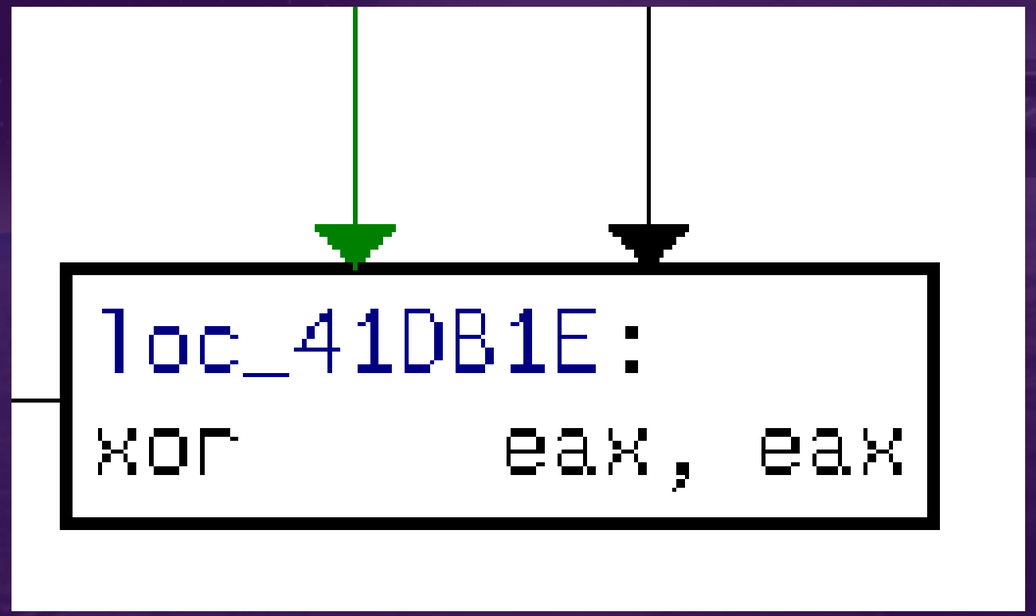


Se il risultato non è uguale a 0, si verifica questa condizione in cui il codice esegue i seguenti passi:

1. Pasa il valore 0 come argomento a sub_41CC50.
2. Chiama la funzione sub_41CC50, che esegue un'operazione utilizzando l'argomento passato.
3. Dopo la chiamata alla funzione, ripristina il puntatore dello stack per rimuovere l'argomento passato e mantenere lo stack pulito.

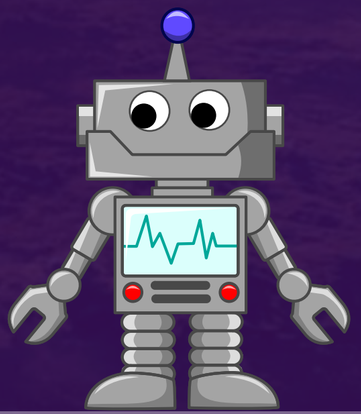
In sintesi, il codice chiama sub_41CC50 con un argomento di 0 e poi ripristina lo stato dello stack.

BLOCCO 9

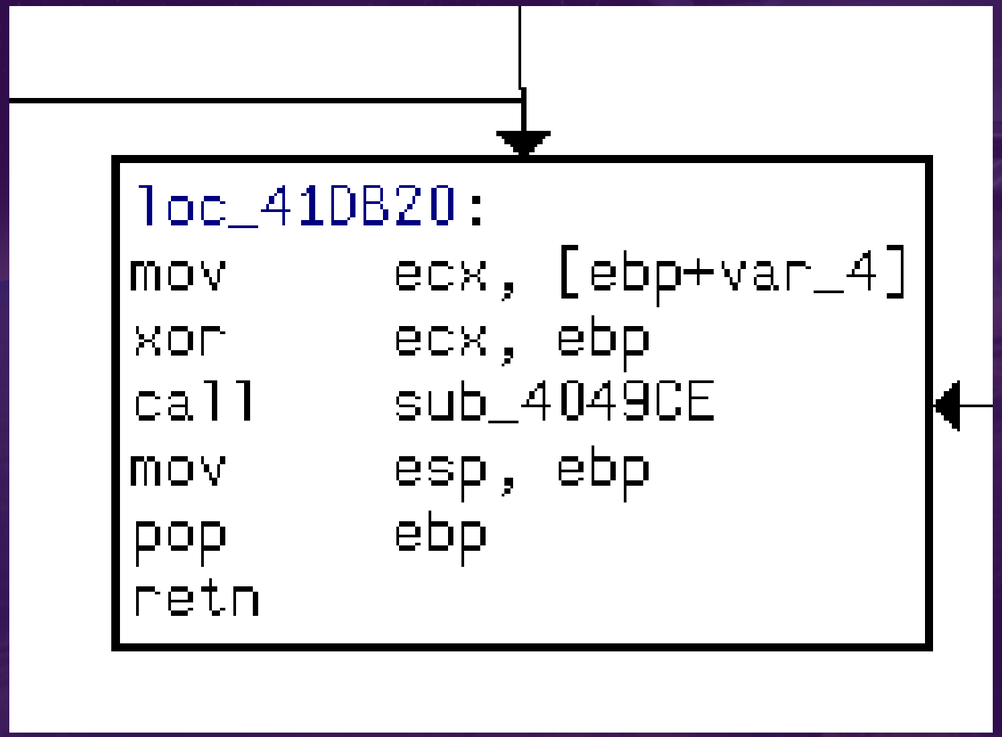


Questo si verifica se il jump è uguale a 0(zero).

Il codice all'etichetta loc_41DB1E azzera il registro eax, impostandolo a 0.
Perché come ben sappiamo xor eax, eax darà come risultato 0.



BLOCCO 10



In quest’ultima parte quindi il codice esegue le seguenti operazioni all'interno della funzione _main:

1. Carica un valore dalla variabile locale var_4 e lo modifica con un'operazione XOR con il valore di ebp.
2. Chiama la funzione sub_4049CE passando il risultato della modifica come argomento.
3. Ripristina il puntatore dello stack e il registro di base per terminare il frame della funzione.
4. Restituisce il controllo al chiamante della funzione main.

In sostanza, il codice modifica una variabile locale, chiama una funzione con il valore modificato, e poi pulisce e restituisce il controllo al chiamante.

CONCLUSIONE

In generale, il programma sembra essere progettato per eseguire operazioni di inizializzazione e configurazione, gestire argomenti di input, e quindi eseguire operazioni di elaborazione e pulizia prima di terminare. La presenza di operazioni di sincronizzazione e gestione di privilegi indica che potrebbe trattarsi di un'applicazione complessa con esigenze di sicurezza e gestione dei thread.