

...

Analisi Malware

Analisi di un malware e interpretazione
Assembly



...

Indice

- 03 • Traccia
- 04 • Analisi malware CFF Explore
- 05 • Analisi Malware Import
- 06 Directory
- 07 • Analisi Malware Headers
- 08 • Analisi Malware Virus Total
- 09
- 10 • Assambly: Intro/Codice
- 11 • Assambly: primo blocco
- 12
- 13 • Assambly: costrutto If
- 14 • Assambly: Ultimo blocco
- 15 • Assambly: Conclusione

Traccia

Con riferimento al file Malware_U3_W2_L5 presente all'interno della cartella «Esercizio_Pratico_U3_W2_L5» sul desktop della macchina virtuale dedicata per l'analisi dei malware, rispondere ai seguenti quesiti:

- Quali librerie vengono importate dal file eseguibile?
- Quali sono le sezioni di cui si compone il file eseguibile del malware?

Con riferimento alla figura in pagina 10, risponde ai seguenti quesiti:

- Identificare i costrutti noti (creazione dello stack, eventuali cicli, costrutti)
- Ipotizzare il comportamento della funzionalità implementata

Analisi Malware: Cff Explore

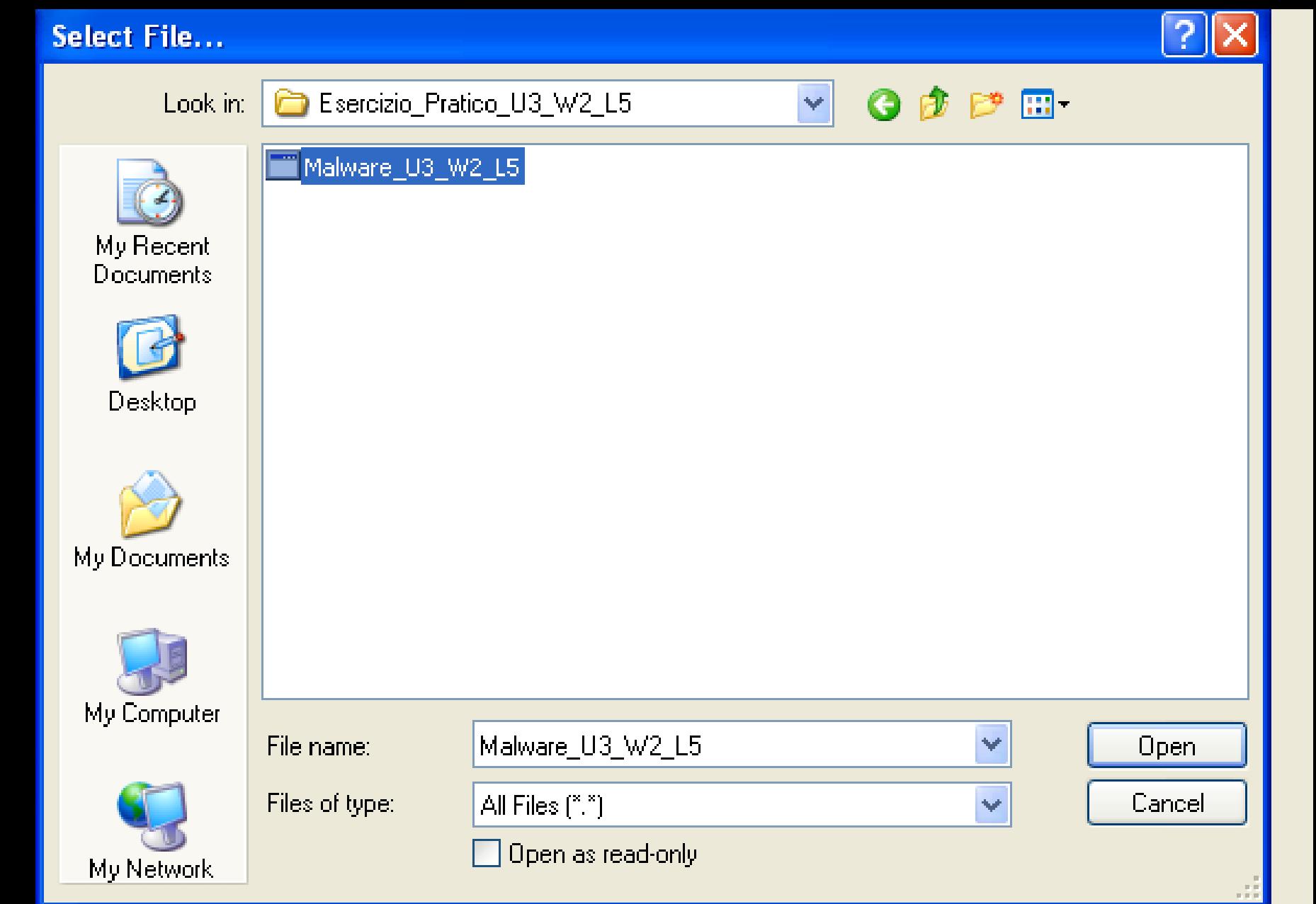
Come prima cosa ci viene chiesto di fare l'analisi di un eseguibile .exe.

Una volta aperta la nostra macchina virtuale windows XP, apriamo il programma CFF Explore.

CFF Explore è utilizzato per visualizzare i file importati ed esportati da un eseguibile.

A noi torna molto utile quando dobbiamo capire cosa fa un malware senza doverlo eseguire.

Apriamo con un doppio click CFF Explore, clickiamo sull'immagine del folder in alto a sinistra e quindi carichiamo l'eseguibile (in questo caso Malware_U3_W2_L5, presente nella cartella sul Desktop Esercizio_Pratico_U3_W2_L5).



Analisi Malware: Import Directory

Ora che abbiamo aperto i file del malware possiamo analizzarlo; come possiamo notare dall'immagine vengono importate 2 Directory:

- KERNELL32.dll = contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria
- WININET.dll = contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP.

dll = libreria dinamica

Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064E0	000064E4	000064E8	000064EC
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

La prima contiene 44 elementi e la seconda sono 5 , nelle prissime slide andremo a vederre il contenuto nel dettaglio

Analisi Malware: Import Directory

- KERNELL32.dll = contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria

In questa tabella possiamo individuare i processi che vengono importati con la directorni KERNELL32.dll e le rispettive allocazioni di memoria.

Non staremo a spiegare processo per processo ma possiamo vedere tra i primi che ci sono delle richieste get, come GetVersion, ExitProcess, TerminateProcess.

Questo ci potrebbe fare intuire che il malware richiede informazioni al sistema operativo, avvia chiude e riprende diversi processi, utilizza stringhe specifiche ecc..

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep
00006940	00006940	027C	SetStdHandle
0000692E	0000692E	0156	GetStringTypeW
0000691C	0000691C	0153	GetStringTypeA
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA
000068E6	000068E6	01E4	MultiByteToWideChar
00006670	00006670	00CA	GetCommandLineA
00006682	00006682	0174	GetVersion
00006690	00006690	007D	ExitProcess
0000669E	0000669E	029E	TerminateProcess
000066B2	000066B2	00F7	GetCurrentProcess
000066C6	000066C6	02AD	UnhandledExceptionFilter
000066E2	000066E2	0124	GetModuleFileNameA
000066F8	000066F8	00B2	FreeEnvironmentStringsA

Analisi Malware: Headers

Seconda parte fondamentale per analizzare il malware è attraverso la sezione Header, le diverse sezioni danno ulteriori informazioni sul comportamento del software.

- **.text** contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato
- **.rdata** include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile, quelle che abbiamo appena recuperato dalle slide precedenti
- **.data** include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile

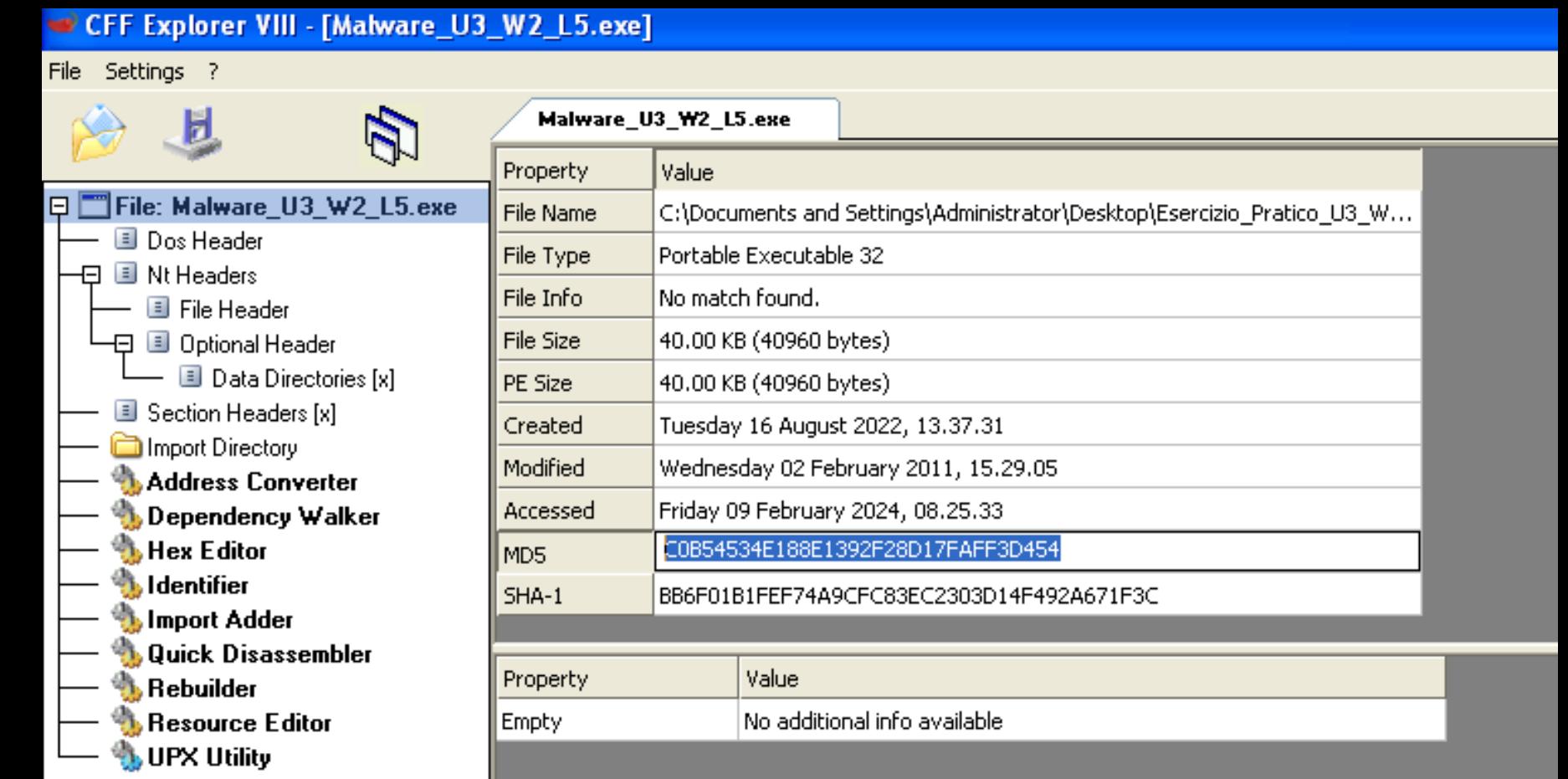
The screenshot shows the CFF Explorer VIII interface. The main window displays the headers of the executable file "Malware_U3_W2_L5.exe". The left pane shows a tree view of the file structure, with the "File: Malware_U3_W2_L5.exe" node expanded to show sub-sections like Dos Header, Nt Headers, File Header, Optional Header, Data Directories, and Section Headers. The "Section Headers" node is currently selected. The right pane is a table with the following data:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linumber... Characteristics
000001E0	000001E8	000001EC	000001F0	000001F4	000001F8	000001FC	00000200	00000202 00000204
Byte[8]	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

Analisi Malware: Virus Total

Infine dalla schermata principale di CFF Explore, una volta caricato il malware, possiamo notare una serie di info generali tra cui gli hash univoci identificativi del malware sia in MD5 che SHA-1.

Utilizzando tool online come **Virus Total**, possiamo controllare se il virus è già stato catalogato e ottenere in questo modo informazioni aggiuntive, già elaborate da altri professionisti.



Quindi copiamo l'hash del malware e lo andiamo a copiare nella barra di ricerca di Virus Total. Ricordiamo che Virus Total è un tool online quindi ci si accede da un browser.

Analyse suspicious files, domains, IPs and URLs to detect malware and other breaches, automatically share them with the security community.

FILE URL SEARCH

COB54534E188E1392F28D17FAFF3D454

Search for a hash, domain, IP address, URL or gain additional context and threat landscape visibility with [VT ENTERPRISE](#).

Analisi Malware: Virus Total

I malware viene identificato da molti utenti come un Troja, cioè un malware che si presenta come un programma legittimo ma che in realtà svolge funzioni dannose, come il furto di dati, la creazione di backdoor per l'accesso remoto al sistema, il danneggiamento dei file o l'installazione di altri malware sul sistema.

The screenshot shows the VirusTotal analysis interface for a file named Lab06-02.exe. The main header displays a red circular progress bar with the number 40/72. Below it, a green checkmark icon indicates a positive result. The file hash is b7177edbf21167c96d20ff803cbcb25d24b94b3652db2f286dc6efd3d8416a. The file type is EXE. It was last analyzed 6 days ago and has a size of 40.00 KB. A warning message states: "40 security vendors and no sandboxes flagged this file as malicious". The detection tab is selected, showing the following vendor analysis:

Security vendor	Detection	Family
Alibaba	Trojan:Win32/Generic.2cc376c1	Antiy-AVL
Avast	Win32:PUP-gen [PUP]	AVG
Bkav Pro	W32.Common.362CBAB4	CrowdStrike Falcon
Cybereason	Malicious.1fef74	Cylance
Cynet	Malicious (score: 100)	DeepInstinct
DrWeb	Trojan.MulDrop7.63090	Elastic

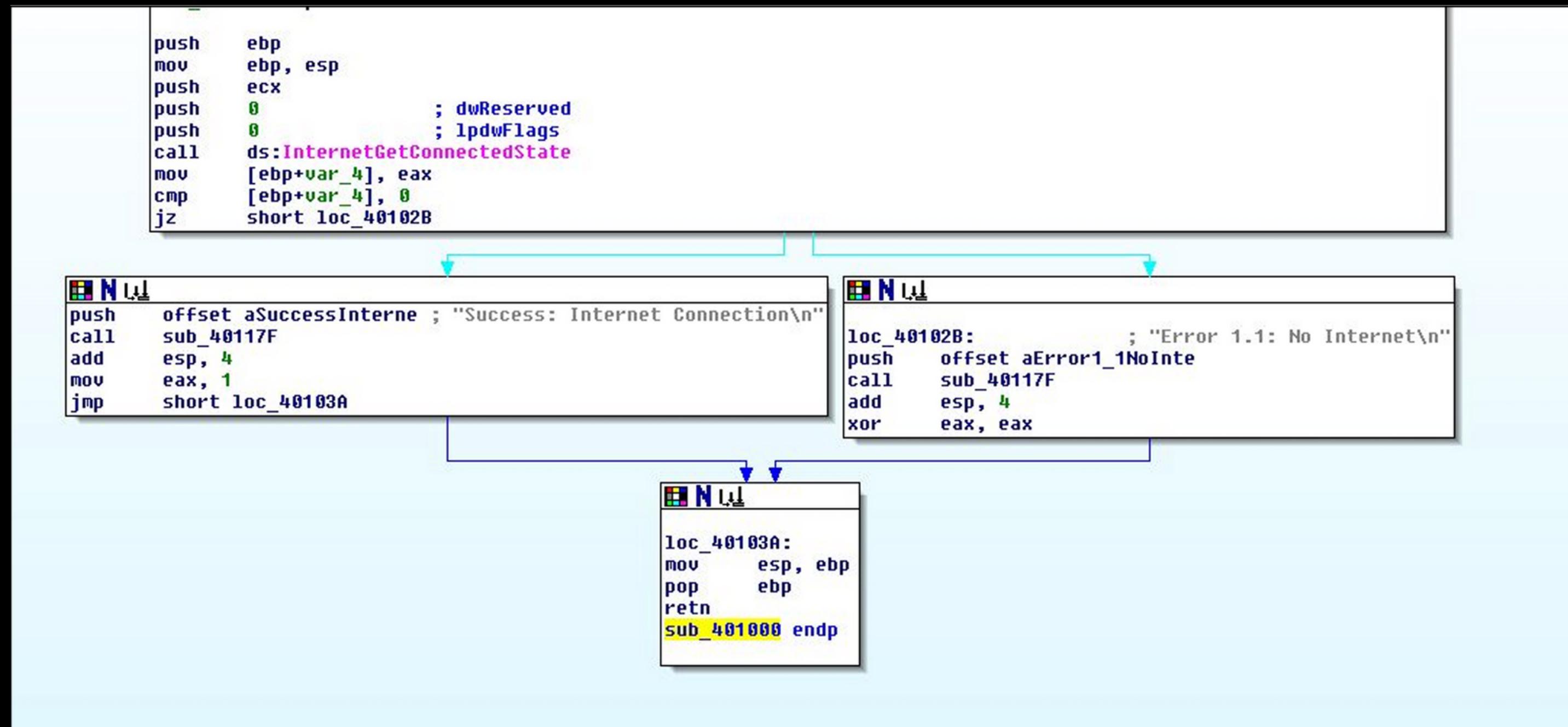
A call-to-action banner encourages users to "Join the VT Community" and "automate checks". Other sections visible include DETAILS, RELATIONS, BEHAVIOR, and COMMUNITY (7).

Analisi Assambly

Premessa :

Assambly è un linguaggio di programmazione a basso livello , cioè è composto da una serie di istruzioni che corrispondono direttamente a istruzioni macchina specifiche per l'architettura del processore su cui il programma viene eseguito . Quello che vediamo noi fa riferimento ad un architettura x86.

Inoltre possiamo notare che ci viene presentato con un diagramma di flusso che è più intuitivo da leggere.



Analisi Assamby: Primo Blocco

Inziamo ad analizzare i diversi costrutti così come si presentano, per avere anche il flusso di esecuzione

```
push ebp |  
mov ebp, esp
```

Queste due funzioni servono per creare un nuovo stack, definendo l'inizio e la fine dello stack.

```
push 0 ; dwReserved  
push 0 ; lpdwFlags
```

Vengono inserite due variabili a valore 0 rispettivamente: un valore intero DWord riservato alla funzione che segue, si può presupporre che lp indica un puntatore alla DWord Flags (quindi punta a dei flags che influenzano la funzione che segue).

```
call ds:InternetGetConnectedState
```

Viene chiamata la funzione che sembra essere una verifica della connessione internet.

Quindi in base al valore delle due variabile precedenti si verifica o meno se c'è connessione ad internet, nello specifico se il valore delle variabili è diverso da quello impostato (0).

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

Analisi Assambly:

Primo Blocco

```
mov [ebp+var_4], eax  
cmp [ebp+var_4], 0  
jz short loc_40102B
```

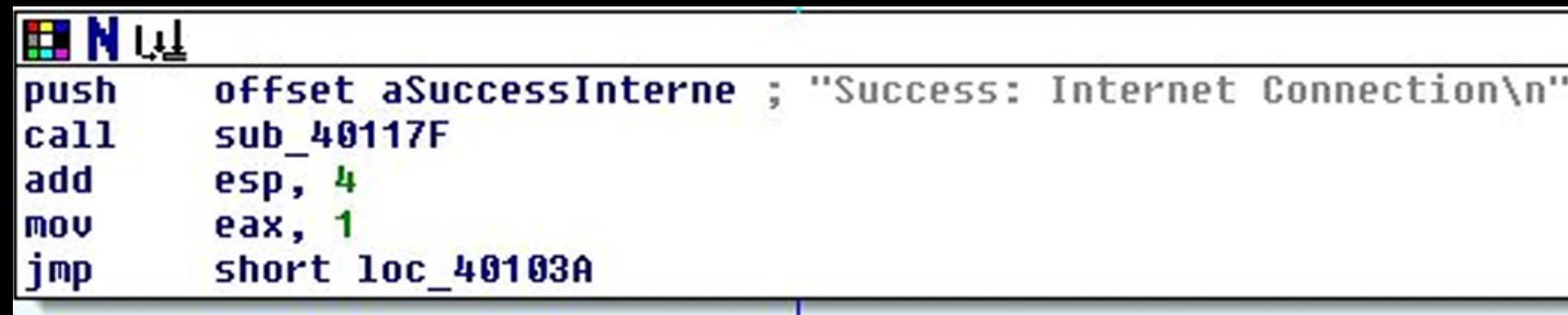
Questo blocco salva come prima cosa il risultato nella funzione `InternetGetConnectedState` (si può interpretare eax come la variabile che rappresenta il risultato della funzione), in uno specifico spazio dello stack; poi compara il risultato della funzione con il valore 0, se risulta uguale 0 salta alla porzione di memoria loc_40102B. Supponiamo che questa locazione di memoria dice che essendo 0 NON c'è connessione.

push	ebp
mov	ebp, esp
push	ecx
push	\$
push	\$
call	ds:InternetGetConnectedState
mov	[ebp+var_4], eax
cmp	[ebp+var_4], \$
jz	short loc_40102B

; dwReserved
; lpdwFlags

Analisi Assambly: Costrutto IF

A questo punto in base al risultato della funzione chiamata si creano due possibilità



```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add    esp, 4
mov    eax, 1
jmp    short loc_40103A
```

Se l'esito del **cmp**, compare, è positivo allora si eseguirà questa parte di codice; viene mostrato il messaggio di successo della connessione e si va a chiamare una serie di operazione presenti della locazione di memoria **sub_40117F**.

```
add esp, 4
mov eax, 1
jmp short loc_40103A
```

Questa parte ripulisce lo stack dopo aver eseguito le funzioni indicate nella subroutine e salta alla parte di memoria indicata.



```
loc_40102B:                         ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
add    esp, 4
xor    eax, eax
```

Se l'esito del compare è negativo si eseguirà questa parte del codice:

push offset aError1_1NoInternet

indica un messaggio di errore alla connessione interent e chiama una determinata subroutine, che come si può notare è la stessa dell'esito positivo **sub_40117F**.

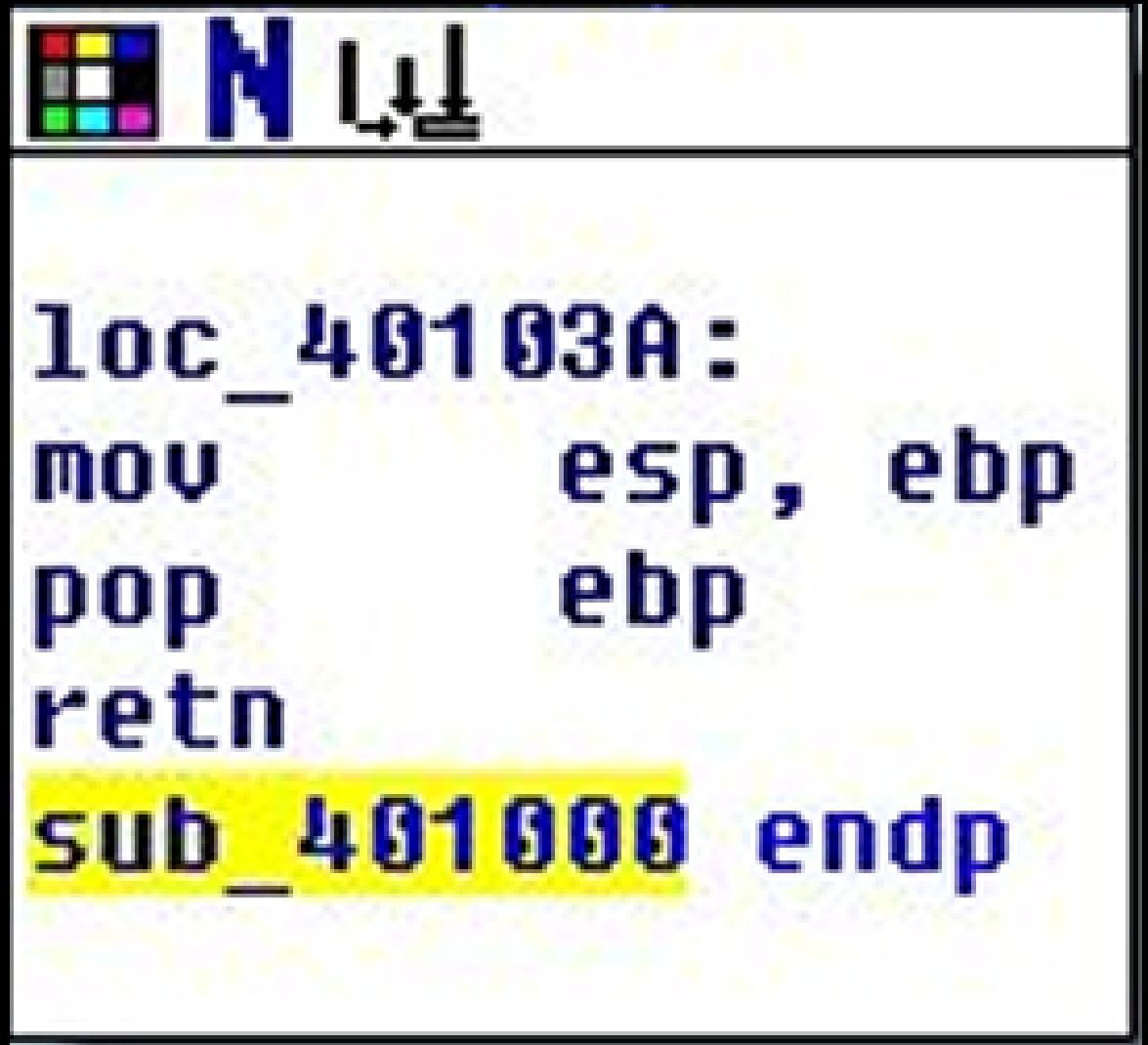
Lo stack viene ripulito e con il **xor eax, eax** azzerà il registro, in quanto fa un costrutto or con lo stesso registro.

Analisi Assambly: Ultimo Blocco

Infine l'ultimo blocco in `loc_40103A` ripulisce lo stack, copiando il valore di `ebp` in `esp` e con il comando `pop` viene estratto dallo stack.

In parole più semplici lo stack viene ripulito togliendo i registri che si sono creati per l'esecuzione del programma.

Infine `retn` equivale al return in linguaggio C, e l'ultima `sub_401000 endp`, termina definitivamente la subroutine indicata nella porzione di memoria indicata => termina il programma.



```
loc_40103A:
    mov    esp, ebp
    pop    ebp
    retn
sub_401000 endp
```

Analisi Assamby: Conclusione

Recap:

Il programma che abbiamo analizzato in Assamby controlla molto semplicemente se c'è o meno connessione ad internet ed in base all'esito procederà in due maniere diverse, per poi terminare sempre l'esecuzione del programma.

Ora è evidente che abbiamo analizzato un costrutto **if** in linguaggio Assamby:

si introduce una funzione **InternetGetConnectedState**, si verifica sè lo **ZeroFlags** è uguale a 0 o diverso (questo è il costrutto **if**) ed in base all'esito si avvieranno due processi diversi.

...



Fine!

