

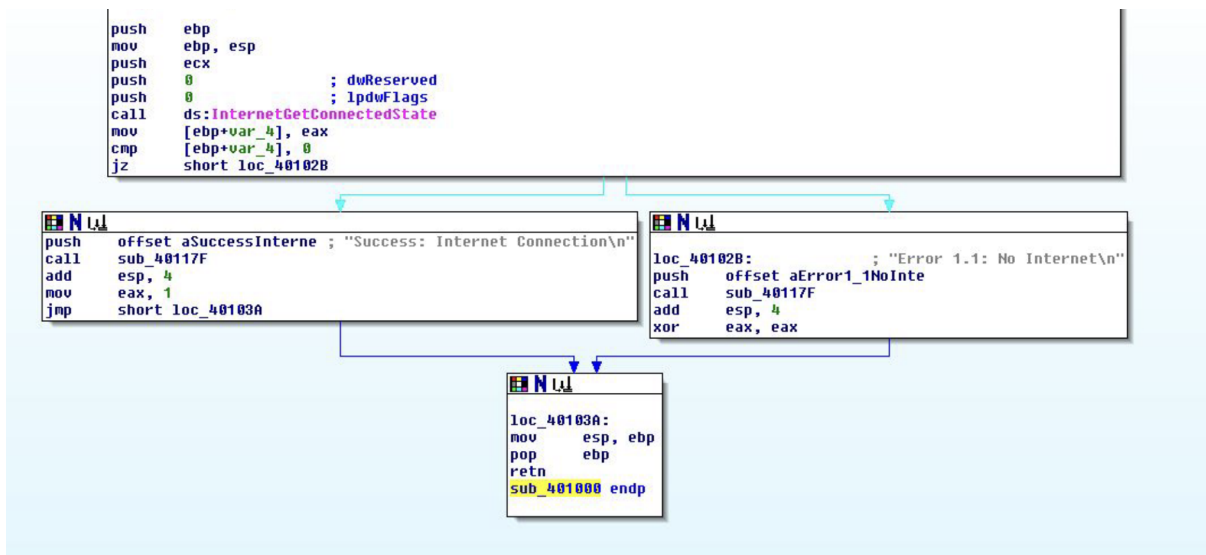
**Traccia:**

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 del malware, rispondere ai seguenti quesiti:

1. Quali **librerie** vengono importate dal file eseguibile? Fare anche una descrizione
2. Quali sono le **sezioni** di cui si compone il file eseguibile del malware? Fare anche una descrizione

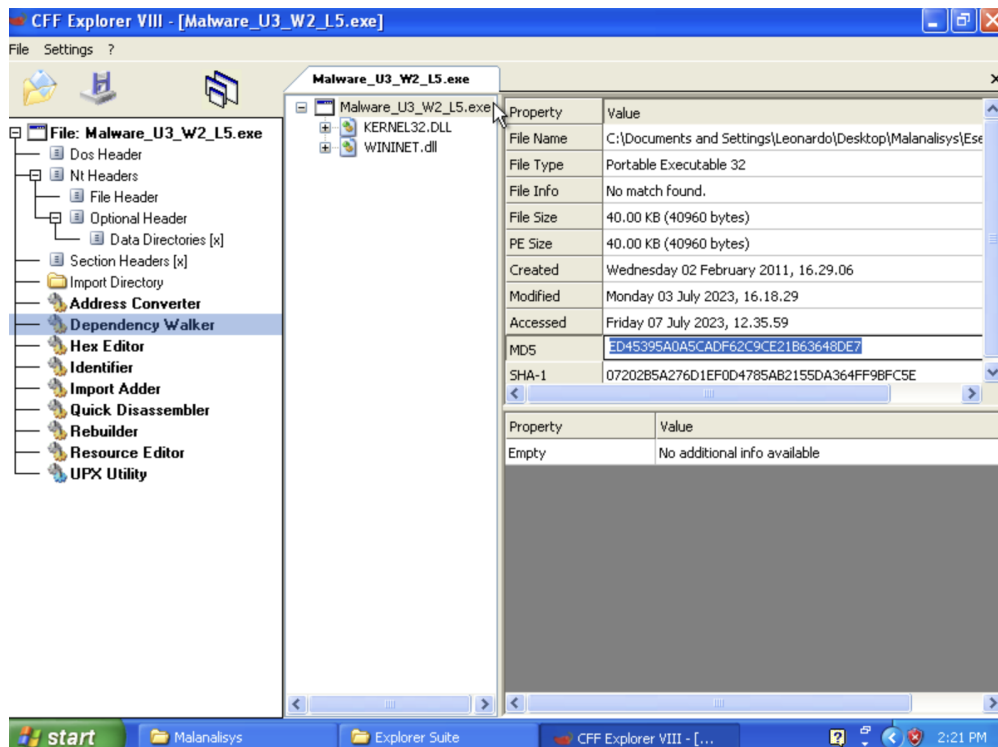
Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:

3. Identificare i **costrutti** noti (creazione dello stack, eventuali cicli, altri costrutti)
4. **Ipotizzare il comportamento della funzionalità implementata**
5. Come ultimo punto, dopo il bonus, spiegare quale istruzione assembly complessa



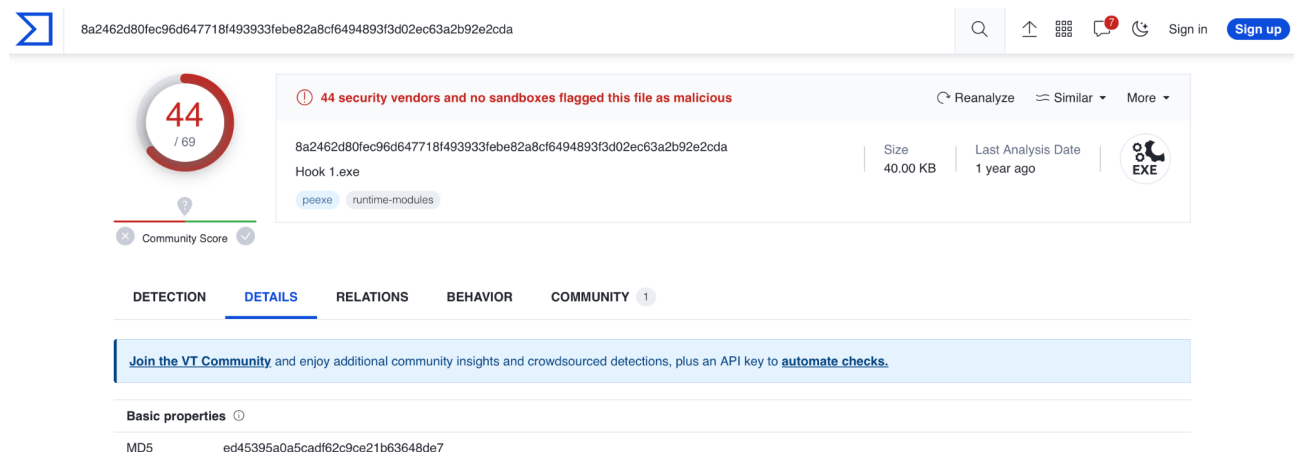
Inizio facendo un analisi statica con **CFF Explorer** del **Malware\_U3\_W2\_L5.exe**

L' hash **MD5** del **Malware\_U3\_W2\_L5.exe** che è presente nel **DEPENDENCY WALKER**



**MD5** → **ED45395A0A5CADF62C9CE21B63648DE7**

Eseguo una ricerca su **VirusTotal**



Viene classificato come **Trojan** → è un tipo di Malware che si presenta come un file apparentemente sicuro ma che una volta eseguito svolge azioni malevole senza il consenso dell' utente

Popular threat label ⓘ **trojan.**

Threat categories **trojan**

**peexe** → Fa riferimento al tipo di file che può essere eseguito (PE)

**runtime - modules** → Indica funzioni o librerie durante l' esecuzione.

## DIRECTORY

KERNEL32.DLL

WININET.dll

Noto che il programma va ad importare due librerie

**Kernel32.dll** → è una libreria di sistema di Windows che serve per interagire con il sistema operativo. Gestisce processi, thread (flussi di esecuzione), memoria, file, tempo e risorse. Essenziale per molti programmi consentendo di eseguire operazioni tipo creare processi, leggere/scrivere file, allocare memoria e gestire il tempo di sistema.

**Wininet.dll** → è una libreria Windows Per effettuare richieste HTTP, inviare e ricevere dati tramite protocolli HTTP, FTP, gestire cookie, cache, proxy e altre operazioni di rete. Fornisce funzionalità per la comunicazione via internet

### SECTION HEADERS (unpack) per vedere in chiaro

|        |          |          |          |          |          |          |    |
|--------|----------|----------|----------|----------|----------|----------|----|
| .text  | 00004A78 | 00001000 | 00005000 | 00001000 | 00000000 | 00000000 | 00 |
| .rdata | 0000095E | 00006000 | 00001000 | 00006000 | 00000000 | 00000000 | 00 |
| .data  | 00003F08 | 00007000 | 00003000 | 00007000 | 00000000 | 00000000 | 00 |

**.text** Questa sezione contiene le istruzioni, ovvero le righe di codice che la CPU eseguirà quando il software viene avviato, è la sezione principale di un file eseguibile (contenente il codice)

**.rdata** Questa sezione contiene informazioni sulle librerie e le funzioni importate ed esportate dall' eseguibile

**.data** Questa sezione contiene dati e variabili globali del programma eseguibile, le variabili seguenti sono accessibili da qualsiasi parte del programma essendo globali

////////////////////////////////////

////////////////////////////////////

```
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
```

## CREAZIONE STACK

**push ebp** → Inserisce il valore EBP(Extended base pointer) nello stack(Memoria temporanea) Crea un frame pointer(puntatore) per accedere ai parametri e alle variabili locali

**mov ebp, esp** → Con il comando **mov ebp, esp** , si stabilisce una connessione tra il frame pointer (**EBP**) e lo stack copiando il valore corrente di **ESP** (Stack Pointer) in **EBP**.

**CHIAMATA DI FUNZIONE** (I parametri vengono passati sulla stack tramite il push)

```
push    ecx
push    0                ; dwReserved
push    0                ; lpdwFlags
call    ds:InternetGetConnectedState
```

## CICLO IF

**cmp [ebp+var\_4], 0** → Confronta il valore della variabile locale **[ebp+var\_4]** con **0**  
**jz short loc\_40102B** → Salta l'indirizzo **loc\_40102B** se il risultato del confronto precedente è uguale a 0

////////////////////////////////////

////////////////////////////////////

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

**I COSTRUTTORI** (creazione stack)

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
```

**CHIAMATA DI FUNZIONE** → (subroutine)

```
call    sub_40117F
```

**ESEGUE UNA PULIZIA DELLO STACK** → (add)

```
add     esp, 4
```

**FA UN SALTO CONDIZIONALE**

```
jmp     short loc_40103A
```

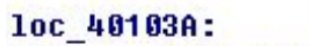
////////////////////////////////////



A screenshot of assembly code from a debugger. The code is as follows:  

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

INDIRIZZO DI MEMORIA “etichetta” (punto di riferimento nell’ esecuzione del programma)

A screenshot of assembly code showing the label `loc_40103A:`.

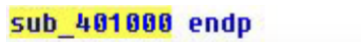
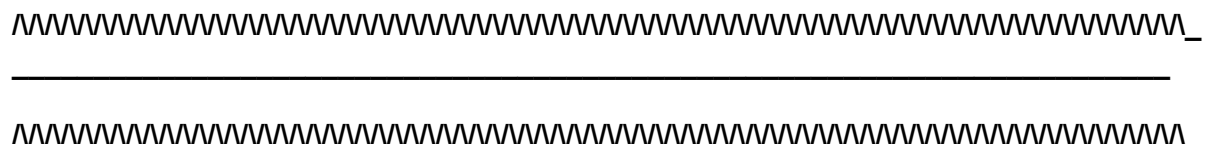
RIMOZIONE DELLO STACK

A screenshot of assembly code showing the instruction `pop ebp`.

RITORNO DELLA FUNZIONE

A screenshot of assembly code showing the instruction `retn`.

FINE DELLA FUNZIONE SUB\_401000

A screenshot of assembly code showing the instruction `sub_401000 endp`.

“Il codice fa una verifica dello stato di connessione, se è attiva mostra un messaggio, altrimenti mostra un errore”