Report Malware Analysis

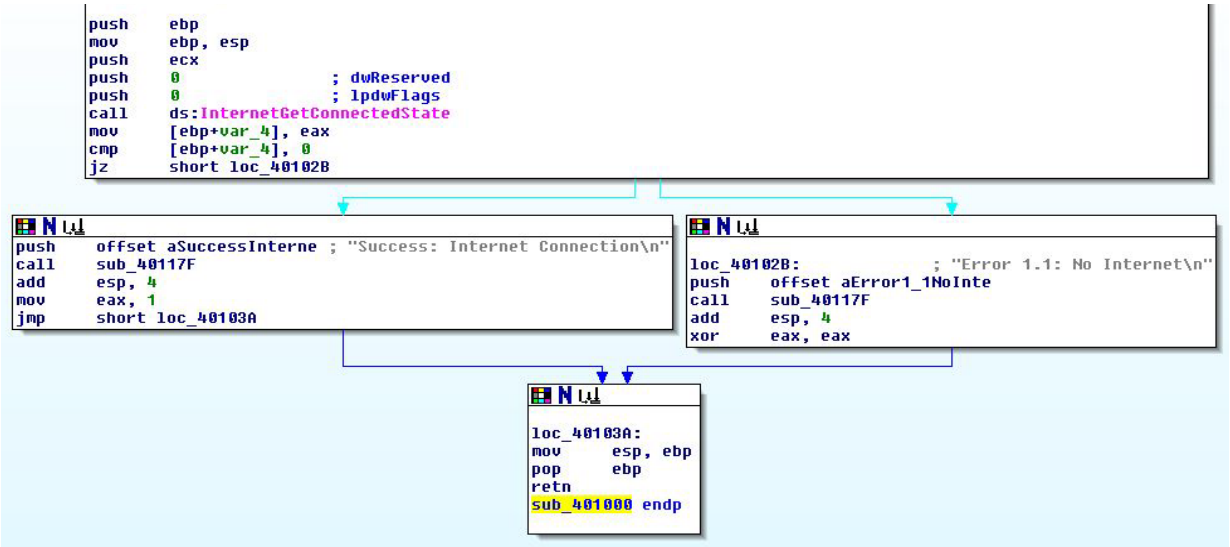
Questo progetto prevede l'esecuzione di attività di **malware analysis**. In particolare, vengono effettuate due attività distinte.

Nella prima parte viene effettuata, all'interno del **laboratorio virtuale** appositamente configurato, l'analisi di base del malware al seguente path:

C:\Documents and Settings\Administrator\Desktop\Esercizio_Pratico_U3_W2_L5\Malware_U3_W2_L5.exe

Tale attività è finalizzata a reperire le seguenti informazioni sul malware: quali **librerie** vengono importate dall'eseguibile e da quali **sezioni** è composto.

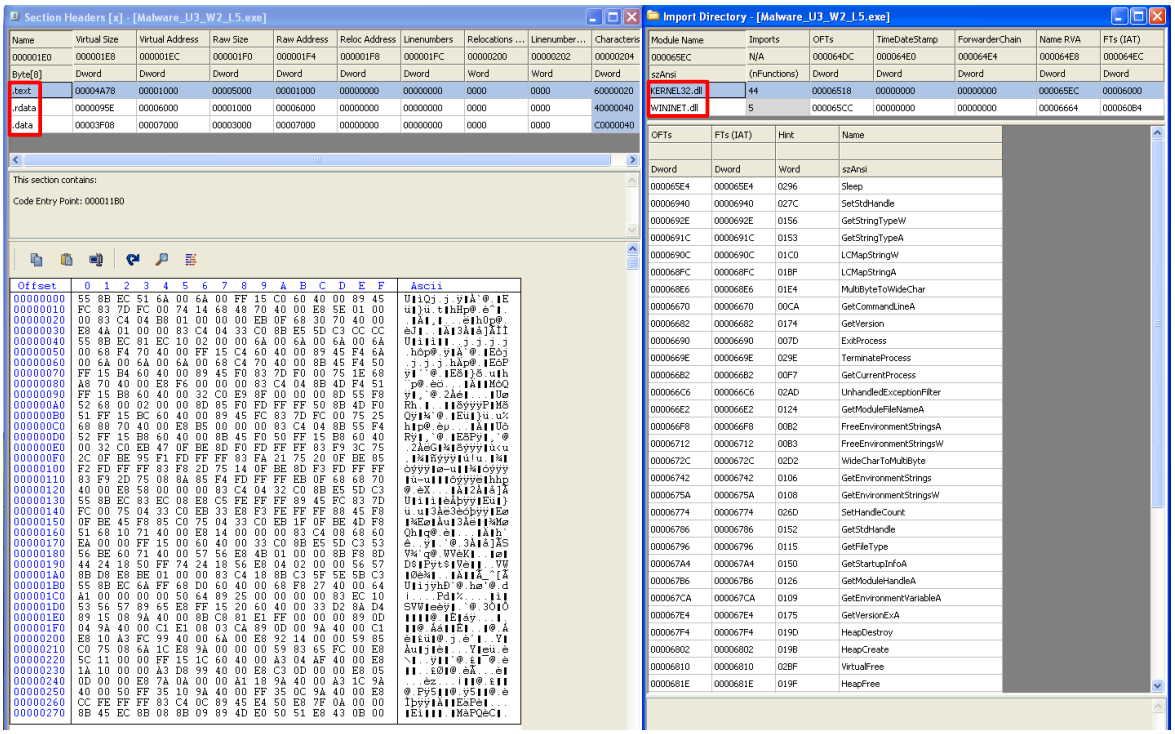
Nella seconda parte, invece, viene effettuata un'analisi più approfondita sul seguente estratto di codice **Assembly x86**:



Gli obiettivi dell'analisi sono: identificare i **costrutti** noti ed ipotizzare il **comportamento** della funzionalità implementata.

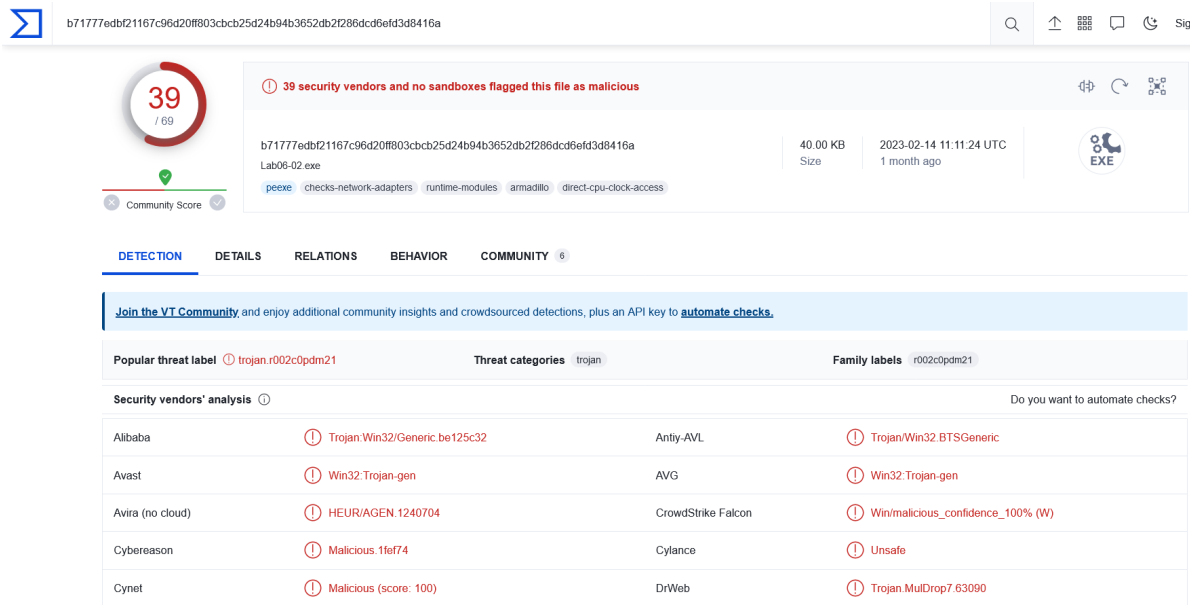
Prima parte

Per ottenere le informazioni richieste è sufficiente effettuare un'analisi **statica** di base del malware. A tal proposito, viene utilizzato il tool CFF Explorer, che restituisce i seguenti risultati:

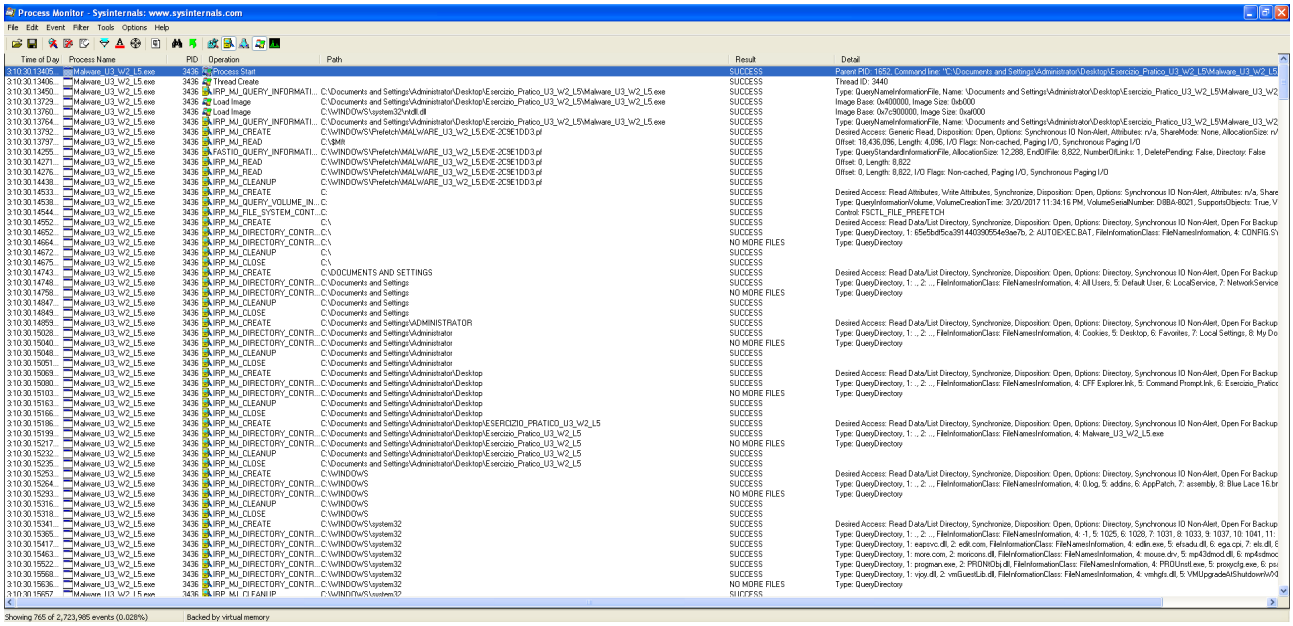


Dallo screenshot in alto si evince che il malware è un file Portable Executable composto dalle sezioni **.text**, **.rdata** e **.data**. Inoltre importa le librerie **KERNEL32.dll** e **WININET.dll**.

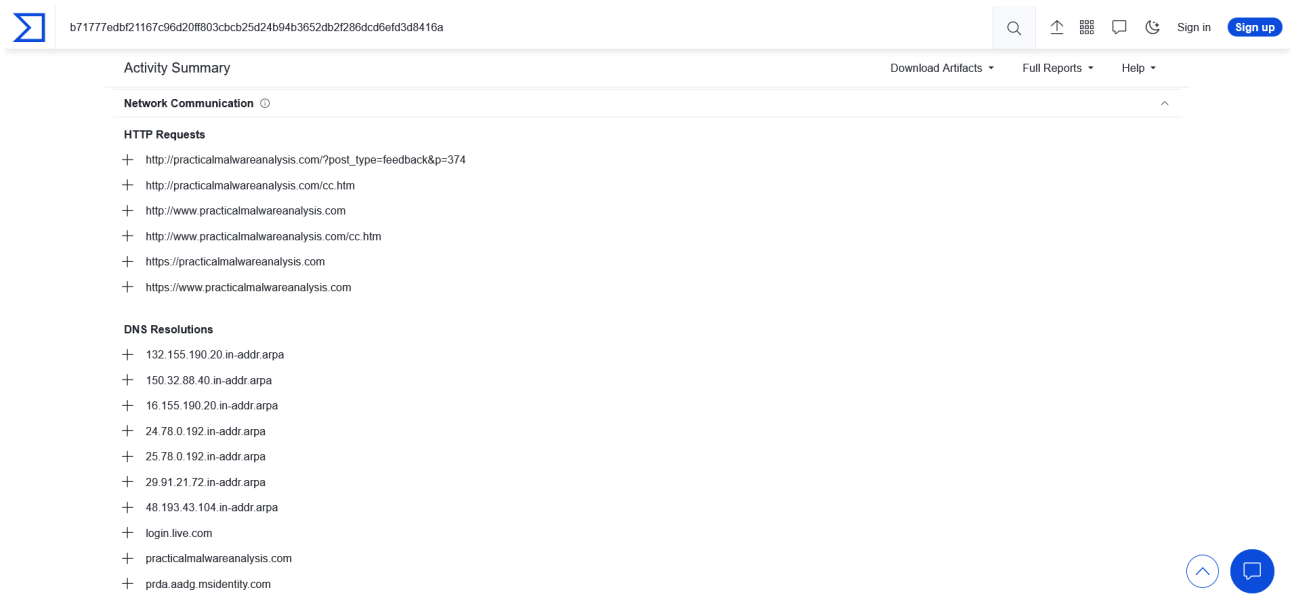
Una scansione con Virustotal conferma la pericolosità del file PE:

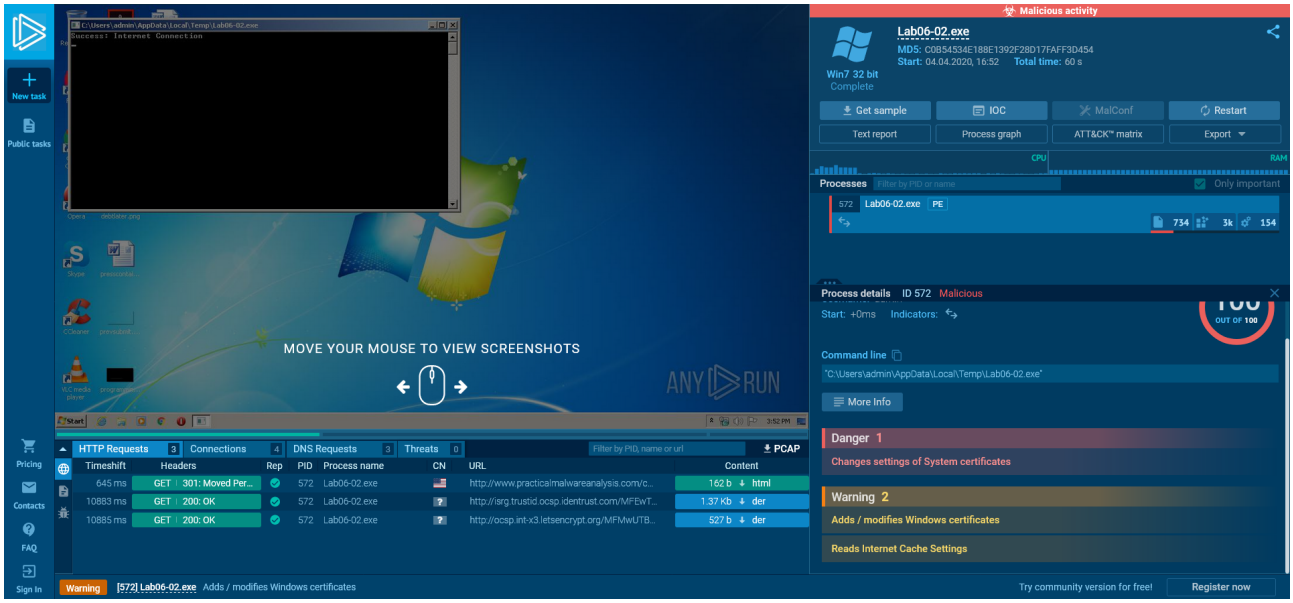


Anche da una sommaria analisi **dinamica** emerge che si tratta di un malware. In particolare, dai seguenti screenshot si evince la creazione da parte del malware di alcuni thread e attività sul filesystem, oltre che numerose chiavi di registro:



Tra le attività più pericolose poste in essere dal malware in esame ritroviamo: la creazione con privilegi elevati di processi e file, recupero di informazioni sul dispositivo, modifica di certificati, controllo sulla presenza di una connessione ad Internet. Inoltre, attraverso l'immissione dell'hash del file in servizi online di malware analysis (VirusTotal e any.run) vengono confermati i sospetti, a cui vengono aggiunti anche altri dettagli, come le richieste HTTP, che non erano state catturate nell'attività precedente:





Seconda parte

Viene, innanzi tutto, effettuata un'analisi del codice riga per riga:

`push ebp` -salva il valore del registro ebp nello stack

`mov ebp, esp` -imposta il registro ebp con il valore corrente dello stack

`push ecx` -salva il valore del registro ecx nello stack

`push 0 ; dwReserved` -salva il valore 0 nello stack come dwReserved

`push 0 ; lpdwFlags` -salva il valore 0 nello stack come lpdwFlags

`call ds: InternetGetConnectedState` -chiama la funzione InternetGetConnectedState dal Data segment

`mov [ebp+var_4], eax` -salva il valore di ritorno della funzione in [ebp+var_4]

cmp [ebp+var_4], 0 -confronta [ebp+var_4] con 0

jz short loc_40102B -salta a loc_h6102B se [ebp+var_4] è zero

offset aSuccessInterne ; "Success: Internet Connection\n" -stringa di output

sub_40117F -chiama la funzione sub_40117F

add esp, 4 -aggiunge 4 byte allo stack

mov eax, 1 -imposta il registro eax a 1

short loc_40103A -salta a loc_40103A

loc_40102B: ; "Error 1.1: No Internet\n" -*label* di destinazione

push offset aError1_1NoInte -pone l'indirizzo della stringa "*Error 1.1: No Internet\n*" nello stack

call sub_40117F -chiama la funzione sub_40117F

add esp, 4 -aggiunge 4 byte allo stack

xor eax, eax -esegue l'operazione xor con il registro eax

loc_40103A: -*label* di destinazione

mov esp, ebp -ripristina lo stack pointer originale

pop ebp -ripristina il valore di ebp dallo stack

retn -ritorna al chiamante

sub_401000 endp -fine della funzione

Volendo effettuare un'analisi più approfondita, vengono esaminati i possibili **costrutti**:

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

Inizializzazione dello stack per la funzione.

```
push    ecx
push    0                   ; dwReserved
push    0                   ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
```

Chiamata di funzione e salvataggio del valore di ritorno nella variabile *var_4*.

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

Confronto e salto condizionato, da cui si può dedurre l'utilizzo di if-else.

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

IF: se la funzione ritorna il valore 0, viene richiamata una funzione che, presumibilmente, restituisce in output

la stringa contenente il messaggio d'errore.

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

ELSE: se la funzione ritorna un valore diverso da 0, viene richiamata una funzione che, presumibilmente, restituisce in output la stringa contenente il messaggio sullo stato della connessione. Esegue infine un salto incondizionato.

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

Epilogo della funzione e cancellazione dello stack.