

# **Unit 3 Week 10**

## **Lesson 5:**

### **Malware e Assembly**

---

7 LUGLIO

---

**Epicode**

**Autore: Pierluigi Amorese**

---

# Analisi statica basica

## Malware\_U3\_W2-L5

La prima parte dell'esercizio di questa settimana ci propone di analizzare un malware presente sulla macchina virtuale Malware Analysis\_Final.

Viene richiesto dalla traccia di specificarne le librerie importate e le sezioni di cui si compone il file eseguibile. Per fare ciò ci avvaleremo di una analisi statica basica:

***"L'analisi statica basica di un malware è una tecnica che consiste nell'esaminare il codice sorgente o l'eseguibile del malware senza eseguirlo, al fine di identificarne le caratteristiche, i comportamenti e le potenziali vulnerabilità."***

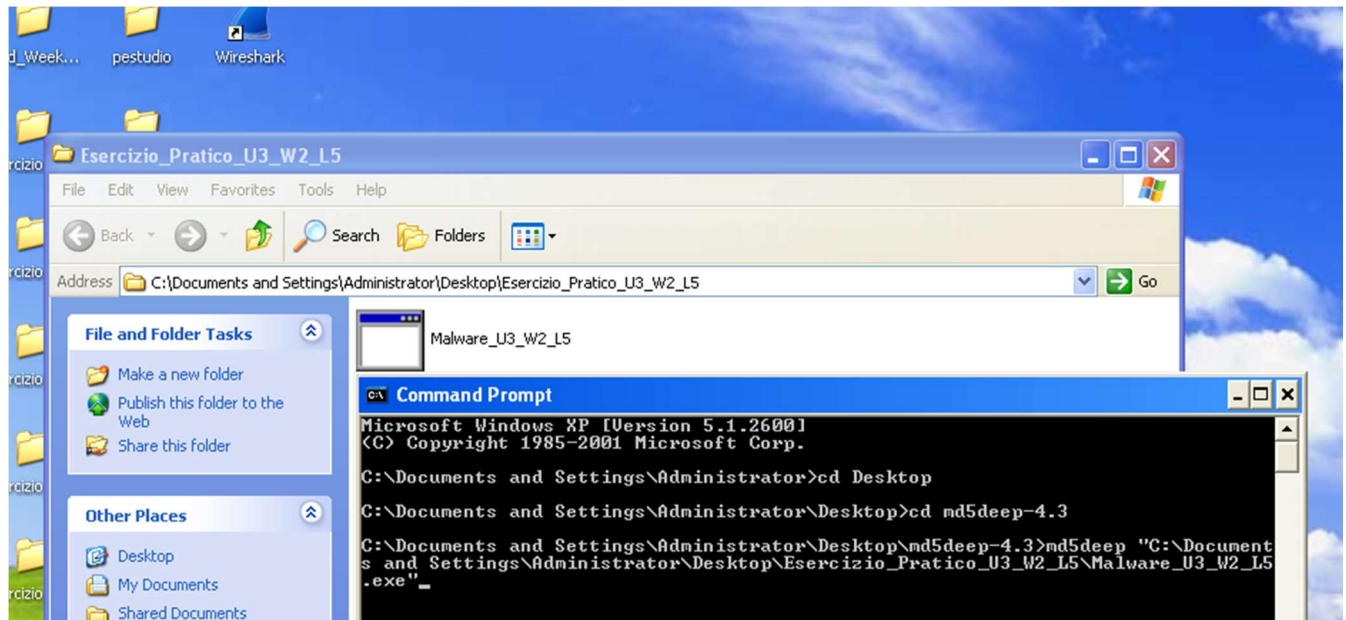
Questa analisi può coinvolgere l'ispezione del file in assembly, l'identificazione di stringhe di testo o pattern sospetti, l'individuazione di funzioni dannose e la valutazione delle possibili azioni che il malware potrebbe intraprendere sui sistemi infettati.

Quando si analizza un potenziale malware il primo passo è assicurarsi che sia effettivamente tale. Sappiamo che ogni file ha una propria firma, quindi possiamo controllare nei database se la firma del nostro presunto malware è nota.

Il primo passo quindi è trovare la firma del file "Malware\_U3\_W2-L5". Per fare ciò utilizziamo il tool da riga di comando **md5deep** già presente sulla VM.

Tale tool ci permette di calcolare l'hash del file, ovvero una stringa alfanumerica unica per identificare un file, che poi si andrà a confrontare con i database su internet.

Sulla VM apriamo Command Prompt e ci spostiamo nella directory del tool e tramite il comando md5deep seguito dal nome del file compreso il suo path.



Tale comando ci restituirà l'hash del file.

```
C:\Documents and Settings\Administrator\Desktop\md5deep-4.3>md5deep "C:\Documents and Settings\Administrator\Desktop\Esercizio_Pratico_U3_W2_L5\Malware_U3_W2_L5.exe"
c0b54534e188e1392f28d17faff3d454 C:\Documents and Settings\Administrator\Desktop\Esercizio_Pratico_U3_W2_L5\Malware_U3_W2_L5.exe
C:\Documents and Settings\Administrator\Desktop\md5deep-4.3>_
```

Dopo una ricerca su Internet troviamo su Any.Run il report del nostro file:

<https://app.any.run/tasks/73dd103b-eb7b-4b2c-a2bf-865f819103cf/>

## General Info

File name:	Lab06-02.exe
Full analysis:	<a href="https://app.any.run/tasks/73dd103b-eb7b-4b2c-a2bf-865f819103cf">https://app.any.run/tasks/73dd103b-eb7b-4b2c-a2bf-865f819103cf</a>
Verdict:	<b>No threats detected</b>
Analysis date:	November 01, 2019 at 13:32:04
OS:	Windows 7 Professional Service Pack 1 (build: 7601, 32 bit)
MIME:	application/x-dosexec
File info:	PE32 executable (console) Intel 80386, for MS Windows
MD5:	<b>C0B54534E188E1392F28D17FAFF3D454</b>
SHA1:	BB6F01B1FEF74A9CFC83EC2303D14F492A671F3C
SHA256:	B71777EDBF21167C96D20FF803CBCB25D24B94B3652DB2F286DCD6EFD3D8416A
SSDEEP:	384:5PvvWL94iMg9lVrpf6lXT2pCeea0dNDJXdhcYyfdyNugreAWoWv:ubvONpf6FT2QbvhDDuGeVoW

Come possiamo vedere dalle General Info nella voce MD5 l'hash corrisponde a quello presente sulla nostra VM. Dalla voce File info capiamo che si tratta di un formato di un file eseguibile per Windows compilato per l'architettura Intel 80386 (processore x86) a 32 bit.

Continuando con il report, troviamo le voci **Sections** e **Imports** che ci serviranno per dedurre il comportamento del file una volta avviato.

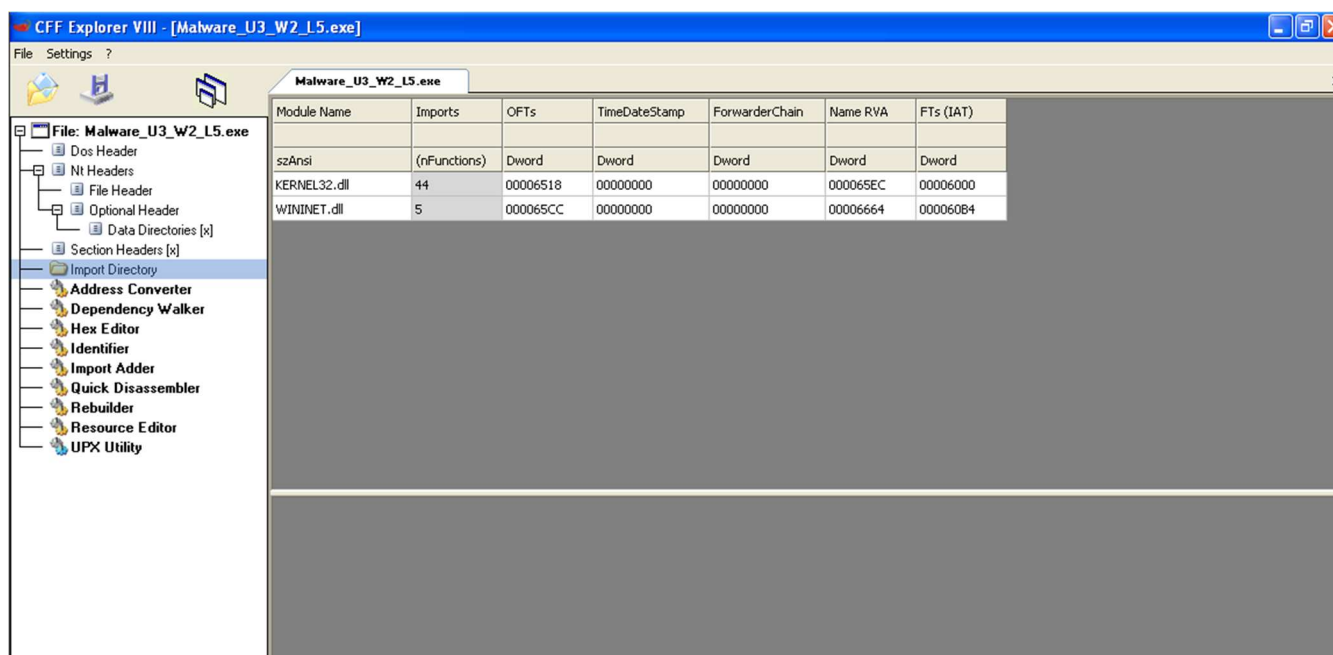
### Sections

Name	Virtual Address	Virtual Size	Raw Size	Characteristics	Entropy
.text	0x00001000	0x00004A78	0x00005000	IMAGE_SCN_CNT_CODE, IMAGE_SCN_MEM_EXECUTE, IMAGE_SCN_MEM_READ	6.37474
.rdata	0x00006000	0x0000095E	0x00001000	IMAGE_SCN_CNT_INITIALIZED_DATA, IMAGE_SCN_MEM_READ	3.66267
.data	0x00007000	0x00003F08	0x00003000	IMAGE_SCN_CNT_INITIALIZED_DATA, IMAGE_SCN_MEM_READ, IMAGE_SCN_MEM_WRITE	0.70192

### Imports

KERNEL32.dll
WININET.dll

A supporto di Any.Run usiamo un tool che ci permette di controllare le librerie e le funzioni importate. Il tool in questione è **CFF Explorer**.



La libreria Kernel32.dll e la libreria Wininet.dll sono componenti del sistema operativo Microsoft Windows che forniscono funzionalità importanti per l'esecuzione di applicazioni.

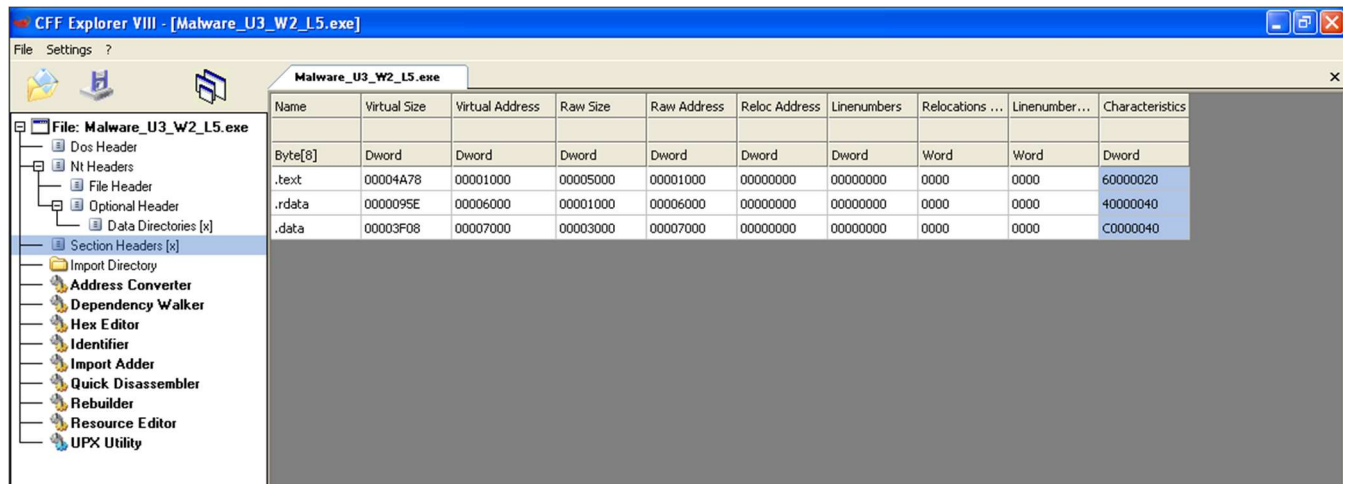
La libreria **Kernel32.dll** contiene numerose funzioni di basso livello utilizzate da Windows e dalle applicazioni per l'accesso al sistema operativo. Alcune delle funzionalità offerte da Kernel32.dll includono la gestione della memoria, la gestione dei file e delle cartelle, la gestione dei processi e dei thread, la gestione degli errori, l'accesso alle risorse del sistema, l'interazione con i dispositivi di input e output, e molte altre funzioni di sistema essenziali.

La libreria **Wininet.dll**, invece, è principalmente utilizzata per fornire funzionalità di rete alle applicazioni Windows. Essa offre un'interfaccia per l'accesso a protocolli di rete come HTTP, HTTPS, FTP e Gopher. Wininet.dll consente alle applicazioni di comunicare con i server remoti, inviare richieste di download o upload di file, effettuare richieste HTTP per ottenere dati da Internet, gestire i cookie e le credenziali di autenticazione, e molto altro.

Entrambe le librerie, Kernel32.dll e Wininet.dll, sono fondamentali per l'esecuzione di molte applicazioni Windows. Forniscono funzionalità di sistema e di rete essenziali che



permettono alle applicazioni di interagire con il sistema operativo e di accedere a risorse esterne come file e servizi di rete.



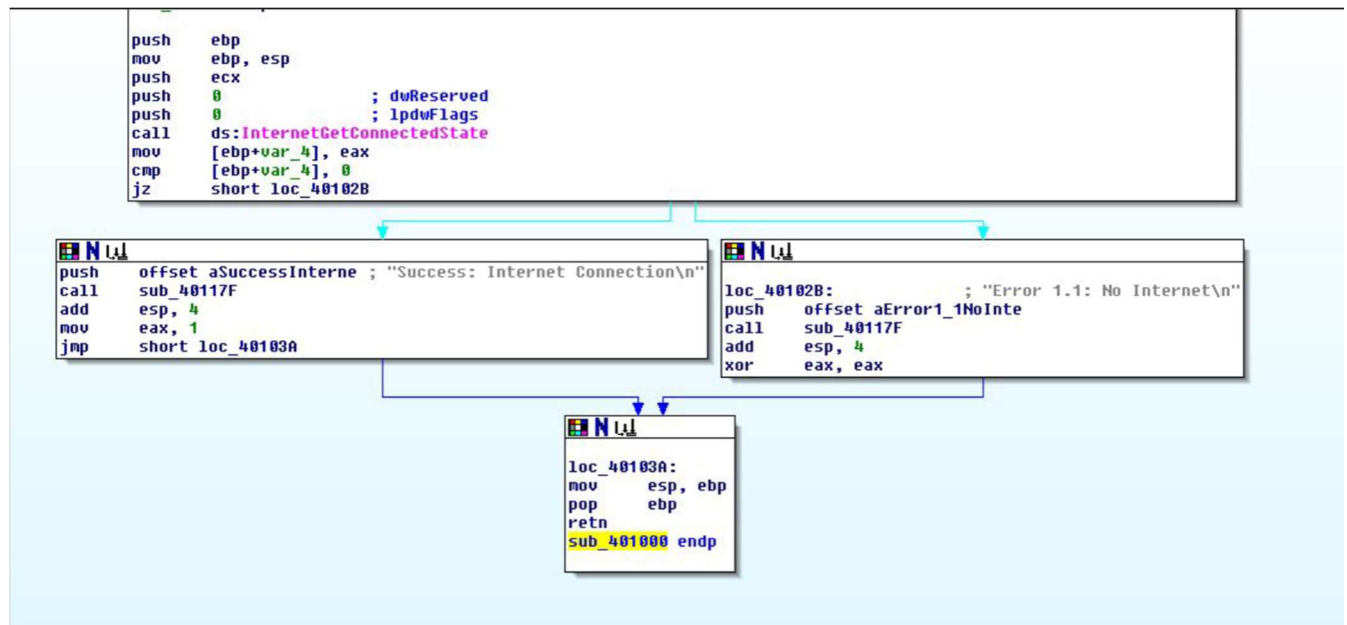
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

Le sezioni di cui si compone il presunto malware sono: .text, .rdata e .data:

1. La sezione .text che contiene il codice eseguibile del programma. Questa sezione contiene le istruzioni in linguaggio macchina che vengono eseguite quando il programma viene avviato. È in questa sezione che risiedono le funzioni e le istruzioni del programma.
2. La sezione .rdata contiene dati di sola lettura, come stringhe di testo o costanti, utilizzate dal programma. Questi dati possono essere letti dal programma, ma non possono essere modificati durante l'esecuzione.
3. La sezione .data contiene dati inizializzati che possono essere letti e modificati durante l'esecuzione del programma. Questa sezione include variabili globali e dati inizializzati che vengono utilizzati dal programma.

# Analisi Assembly

In questa seconda parte passiamo all'analisi del codice assembly che ci viene consegnato.



Si inizia con le istruzioni:

- `push ebp` ; Salva il valore corrente di ebp nello stack
- `mov ebp, esp` ; Crea una nuova base per il puntatore dello stack
- `push ecx` ; Salva il valore corrente di ecx nello stack
- `push 0` ; dwReserved viene messo nello stack come argomento per la funzione successiva
- `push 0` ; lpdwFlags (puntatore a dword per i flag) viene messo nello stack come argomento per la funzione successiva
- `call ds:InternetGetConnectedState` ; Chiama la funzione InternetGetConnectedState
- `mov [ebp+var_4], eax` ; Salva il risultato della chiamata alla funzione nella variabile locale var\_4
- `cmp [ebp+var_4], 0` ; Confronta il valore di var\_4 con 0

- `jz short loc_40102B` ; Salta a `loc_40102B` se `var_4` è uguale a 0 (quindi nessuna connessione Internet); Se la connessione Internet è presente, stampa "Success: Internet Connection" e imposta `eax` a 1
- `push offset aSuccessInterne` ; Mette l'indirizzo del messaggio "Success: Internet Connection" nello stack come argomento per la funzione successiva
- `call sub_40117F` ; Chiama una funzione `sub_40117F` per stampare il messaggio
- `add esp, 4` ; Libera lo spazio per l'argomento dalla stack
- `mov eax, 1` ; Imposta `eax` a 1
- `jmp short loc_40103A` ; Salto incondizionato a `loc_40103A`
  
- `loc_40102B`: ; Se non c'è connessione Internet, stampa "Error 1.1: No Internet"
- `push offset aError1_1NoInter` ; Mette l'indirizzo del messaggio "Error 1.1: No Internet" nello stack come argomento per la funzione successiva
- `call sub_40117F` ; Chiama una funzione `sub_40117F` per stampare il messaggio
- `add esp, 4` ; Libera lo spazio per l'argomento dello stack
- `xor eax, eax` ; esegue un'operazione con l'operatore logico XOR tra il registro EAX e se stesso, che serve ad azzerare il valore contenuto fino a quel momento in EAX ed inizializzarlo a 0, in quanto un operatore logico tra due bit identici restituisce sempre 0
  
- `loc_40103A`:
- `mov esp, ebp` ; Ripristina il puntatore dello stack all'indirizzo originale
- `pop ebp` ; Ripristina `ebp` dallo stack
- `retn` ; Restituisce il controllo alla chiamata della funzione

Questa porzione di codice assembly x86 verifica se il sistema ha una connessione Internet attiva e stampa un messaggio di successo o di errore in base al risultato. Quindi si può affermare che il codice presenta un costrutto if alla funzione **"InternetGetConnectedState"** che restituisce TRUE se è presente una connessione Internet attiva o FALSE se non c'è una connessione Internet.



