

LUCA GASPARI

# Analisi Malware\_U 3\_W2\_L5

2024

# Traccia esercizio S10L5

## 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:

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

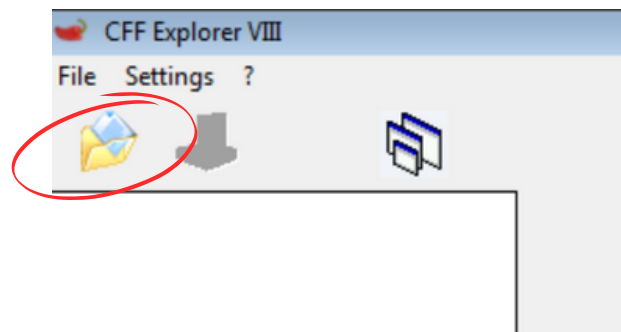
Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:

3. Identificare i **costrutti** noti (creazione dello stack, eventuali cicli, altri costrutti )
4. **Ipotizzare il comportamento della funzionalità implementata**
5. **BONUS** fare tabella con significato delle singole righe di codice assembly

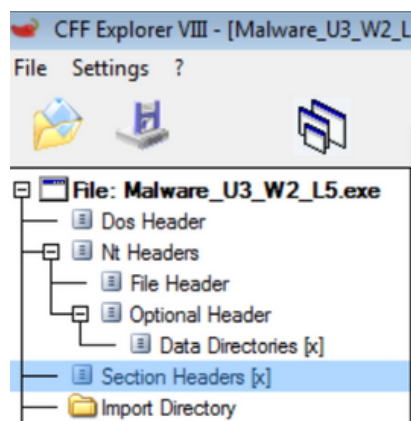
Questo report analizza il malware chiamato "Malware\_U3\_W2\_L5.exe" per identificare le librerie importate, le sezioni del file eseguibile, i costrutti noti, il codice assembly e per ipotizzare il comportamento della funzionalità implementata.

## Librerie Importate

Per capire le librerie che importa questo malware sono andato ad utilizzare il programma CFF Explorer, questo è un potente strumento di analisi e modifica per file eseguibili (PE - Portable Executable) su piattaforma Windows, ci permette di caricare file eseguibili ed andare ad analizzare le componenti, passaggio molto importante per fare delle ipotesi del funzionamento di un malware.



Per prima cosa sono andato a caricare il file eseguibile cliccando sull'icona della cartella, successivamente una volta caricato mi sono mosso nelle directory nel file per andare a cercare le librerie che importa e le sezioni che compongono questo malware.



Nella sezione **section headers** si trovano le sezioni che formato il malware:

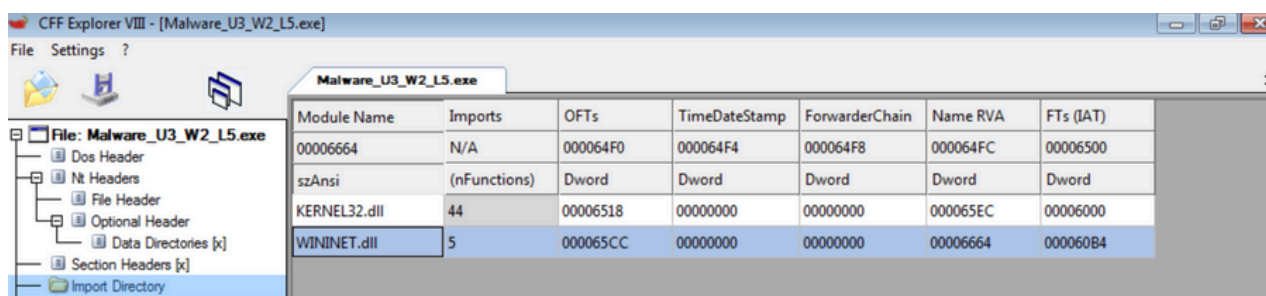
Malware_U3_W2_L5.exe								
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumb
000001E0	000001E8	000001EC	000001F0	000001F4	000001F8	000001FC	00000200	00000202
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000

Quindi il file eseguibile di questo malware è composto dalle seguenti sezioni:

- **.text**: Contiene il codice eseguibile, cioè le istruzioni che andrà ad eseguire la CPU una volta che il software sarà avviato.
- **.rdata**: Contiene le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile.
- **.data**: Contiene dati di lettura/scrittura, come variabili globali iniziate.

Queste sezioni sono standard per la maggior parte degli eseguibili PE (Portable Executable).

Nella sezione **import directory** si trovano le librerie che importa l'eseguibile.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
00006664	N/A	000064F0	000064F4	000064F8	000064FC	00006500
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

Quindi le librerie che importa sono le seguenti:

- **KERNEL32.dll**
- **WININET.dll**

La prima libreria cioè **KERNEL32.dll** è una libreria fondamentale del sistema operativo Windows, contiene le funzioni per interagire con il sistema operativo, permette ad esempio di manipolare i file e gestire la memoria.

La seconda libreria **WININET.dll** è utilizzata per far interagire le applicazioni con Internet e implementare alcuni protocolli di rete come **http, ftp**.

Le librerie contengono al loro interno delle funzioni con scopi specifici:

Funzioni libreria **KERNEL32.dll**:

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

## Sleep

Sospende l'esecuzione del thread chiamante per un intervallo di tempo specificato, ed è utilizzata per introdurre ritardi nell'esecuzione del codice.

## SetStdHandle

Imposta il valore di un handle standard (standard input, standard output, o standard error) per il processo chiamante, ed è utilizzata per reindirizzare l'input o l'output standard.

## GetStringTypeW

Recupera informazioni sul tipo di carattere **Unicode** specificato.  
Utilizzata per l'elaborazione dei caratteri e la gestione delle stringhe Unicode

## GetStringTypeA

Recupera informazioni sul tipo di carattere **ANSI** specificato, ed è utilizzata per l'elaborazione dei caratteri e la gestione delle stringhe ANSI.

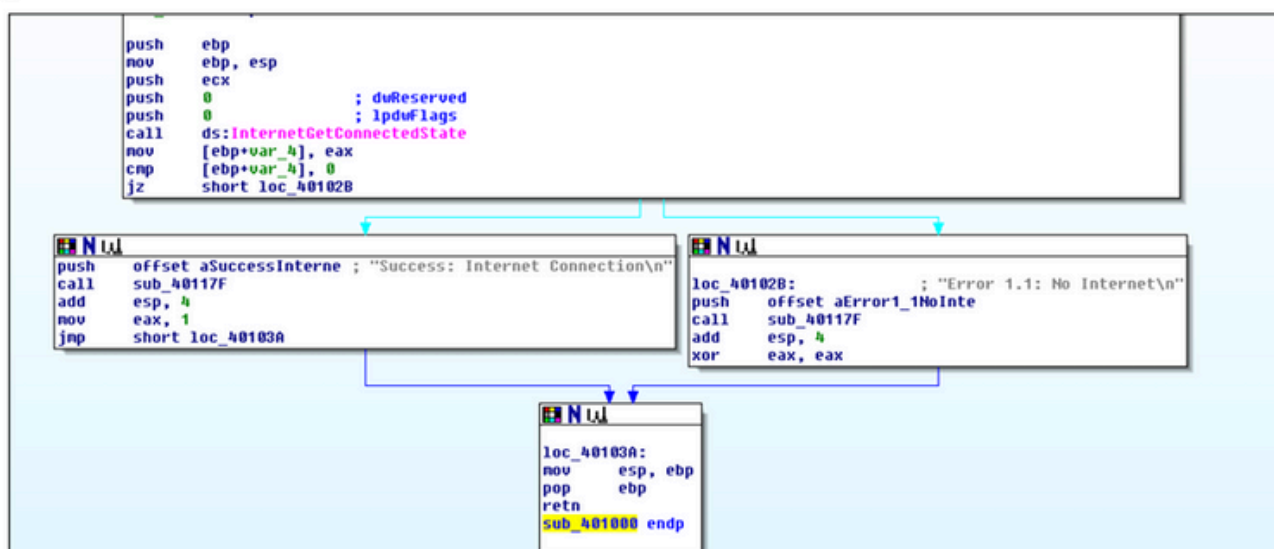
Funzioni libreria **WININET.dll**:

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

Queste funzioni consentono al malware di aprire connessioni URL, leggere dati da Internet, e verificare lo stato della connessione.

## Costrutti Noti nel Codice Assembly

Figura 1





Analizzando il codice assembly, ho identificato i seguenti costrutti noti:

### 1.Creazione dello stack:

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

Questi comandi salvano il puntatore base corrente (**EBP**) nello **stack** e poi impostano **EBP** all'attuale puntatore dello stack (**ESP**), preparando lo stack frame per la funzione.

### 2.Controllo Condizionale if:

```
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

#### Confronto (cmp):

L'istruzione **cmp [ebp+var\_4], 0** confronta il valore memorizzato nella variabile locale **[ebp+var\_4]** con 0.

**[ebp+var\_4]** è una variabile che contiene il risultato della chiamata a **InternetGetConnectedState**, che verifica lo stato della connessione Internet.

#### Salto condizionale:

L'istruzione **jz short loc\_40102B** (jump if zero) salta all'etichetta **loc\_40102B** se il confronto precedente ha trovato che il valore in **[ebp+var\_4]** è zero. Questo è equivalente a dire "se non c'è connessione Internet, salta all'etichetta loc\_40102B".

### 3.Chiamata di funzione:

```
call    ds:InternetGetConnectedState
call    sub_40117F
```

La prima chiamata di funzione verifica lo stato della connessione a Internet, mentre la seconda è una chiamata a una subroutine interna per la gestione dei messaggi.

## **Ipotesi del comportamento della funzionalità implementata**

Il malware verifica lo stato della connessione a Internet all'avvio, se una connessione è disponibile, il malware continua con la sua routine stampando un messaggio di successo. Se non è disponibile, stampa un messaggio di errore e interrompe l'esecuzione o potrebbe attivare ulteriori routine di gestione dell'errore.

### **Spiegazione del codice Assembly:**

**push ebp:** salva il base pointer attuale nello stack

**mov ebp, esp:** imposta il base pointer all'attuale stack pointer

**push ecx:** salva il registro ecx nello stack

**push 0:** inserisce 0 nello stack

**push 0:** inserisce 0 nello stack

**call ds:** chiama la funzione per verificare lo stato della connessione

**mov [ebp+var\_4], eax:** salva il risultato della chiamata a funzione

**cmp [ebp+var\_4], 0:** confronta il risultato con 0

**jz short loc\_40102B:** se il risultato è 0, salta a loc\_40102B

**push offset aSuccessInterne:** inserisce l'indirizzo del messaggio di successo nello stack

**call sub\_40117F:** chiama la subroutine per stampare il messaggio

**add esp, 4:** aggiunge 4 al stack pointer

**mov eax, 1:** imposta eax a 1

**jmp short loc\_40103A:** salta a loc\_40103A

**push offset aError1\_1NoInte:** inserisce l'indirizzo del messaggio di errore nello stack

**call sub\_40117F:** chiama la subroutine per stampare il messaggio

**add esp, 4:** aggiunge 4 al stack pointer

**xor eax, eax:** imposta eax a 0

**mov esp, ebp:** ripristina lo stack pointer

**pop ebp:** ripristina il base pointer

**ret:** ritorna dalla funzione