

Week 10 - TEST Settimanale

Analisi del Malware

Consegna

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?

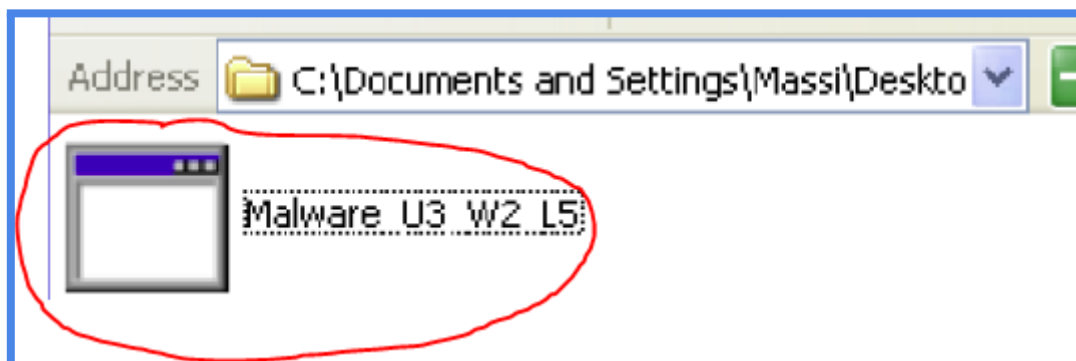
Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:

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

Procedimento

Dopo aver aperto la nostra macchina virtuale in Windows 10, andiamo ad analizzare il Malware richiesto dalla consegna.

Questo è il Malware che prendiamo in analisi:



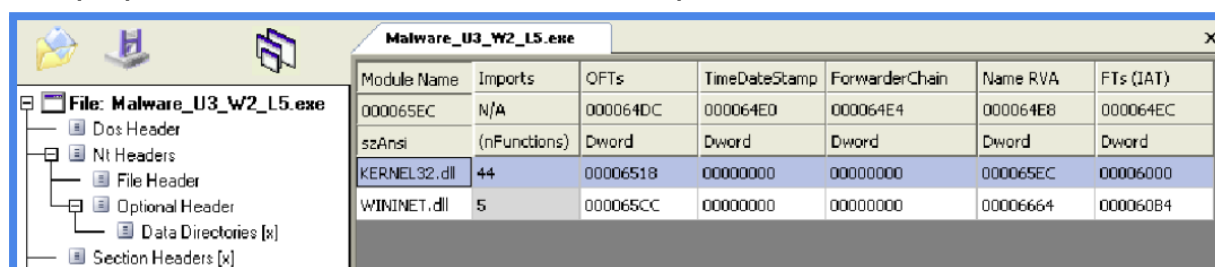
Per analizzarlo vado ad utilizzare **CFF Explorer**.

Per ottenere informazioni generali sul Malware la tecnica di analisi che utilizzeremo sarà l'**Analisi Statica Basica**.

1. Librerie importate dal file eseguibile

Lanciando il programma e analizzando l'Header del PE (Portable Executable) andiamo nella sezione **Import Directory**.

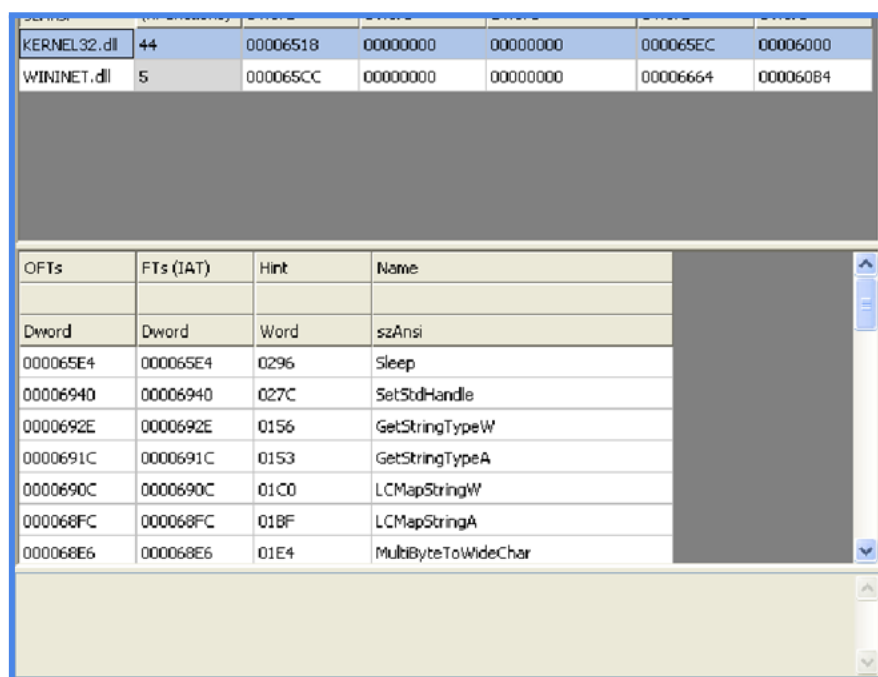
Da qui possiamo analizzare le librerie importate dal file:



Module Name	Imports	OFIs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064ED	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

Come si può notare le librerie importate sono **KERNEL32.dll** e **WININET.dll**

La prima libreria contiene funzioni per interagire con l'OS in questione, mentre la seconda permette al Malware di implementare i protocolli di rete.



Module Name	Imports	OFIs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064ED	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

OFIs	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

WININET.dll	5	000065CC	00000000	00000000	00006664	00006084
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			

2. Sezioni che compongono il malware

Nella slide seguente abbiamo riportato le sezioni delle quali si compone il software:

.text

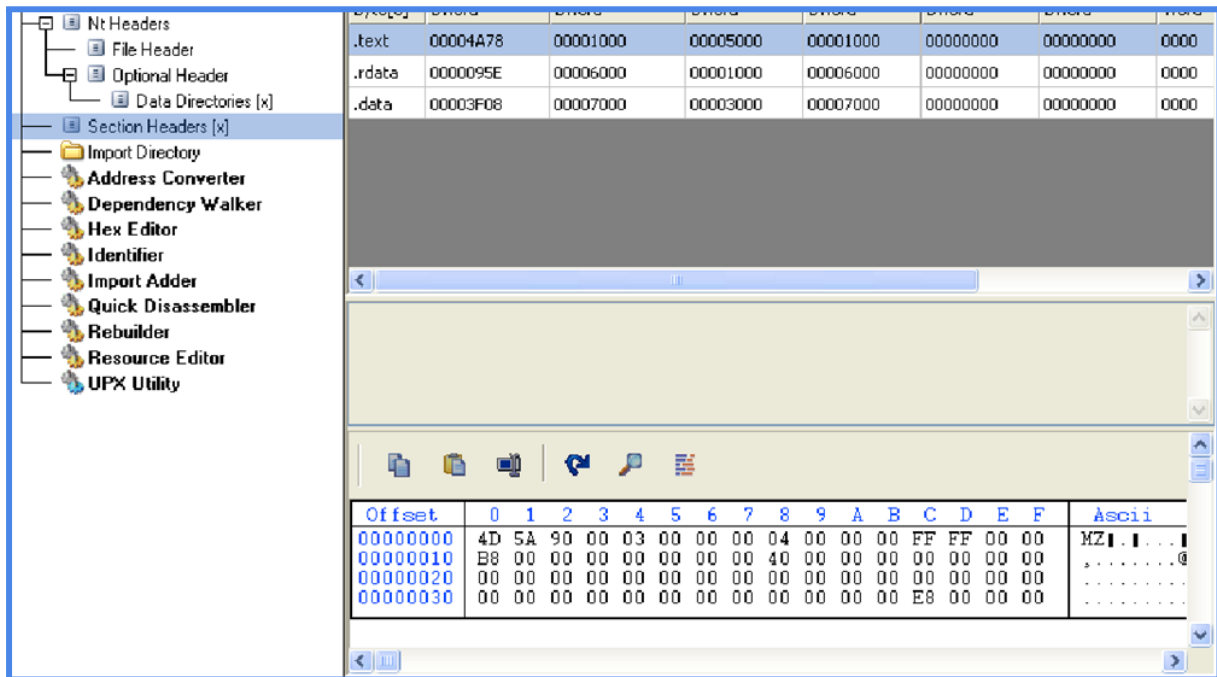
Questa sezione contiene le righe di codice che la CPU eseguirà una volta avviato il software.

.rdata

Sezione che include le informazioni relative alle librerie e le loro funzioni, importate prima ed esportate poi dall'eseguibile

.data

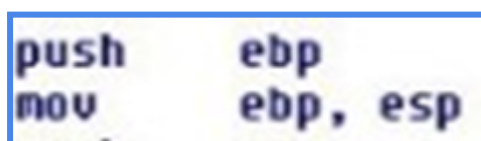
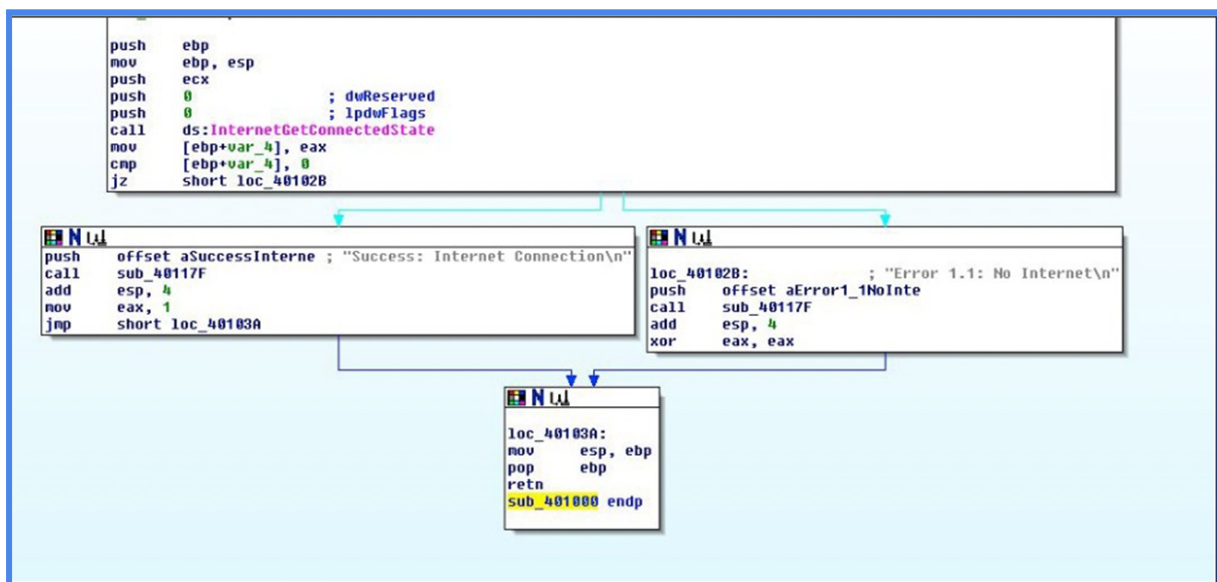
Quest'ultima contiene i dati e le variabili globali del programma eseguibile. E' importante sapere che essendo variabili globali, esse sono accessibili da qualsiasi funzione dell'eseguibile.



3. Costrutti Noti

In questa fase andremo a fare un'**Analisi Statica Avanzata**

Dopo aver tradotto il programma in Assembly X86 sono andato ad analizzare i costrutti noti, come richiesto dalla consegna:



Creazione dello stack

```

push    ecx
push    0                ; duReserved
push    0                ; lpdwFlags
call    ds:InternetGetConnectedState

```

Passaggio parametri sullo Stack attraverso il comando Push e la chiamata seguente

```

cmp     [ebp+var_4], 0
jz      short loc_40102B

```

Ciclo IF per controllare se la connessione è attiva oppure no. Con il comando “jz” (JUMP) vengono controllati i due differenti stati.

```

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

```

```

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

```

Nelle due immagini precedenti avviene la risposta al controllo dell'IF:

Nel primo caso la ZF è a 0, mentre nel secondo lo ZF è a 1

```

loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp

```

In questa fase lo STACK viene ripulito

4. Comportamento della funzionalità implementata

In base alle analisi effettuate, si può supporre che il codice inizia dichiarando una variabile denominata `var_4`, seguita dalla chiamata della funzione `InternetGetConnectedState`.

Successivamente, si presenta una decisione basata su una condizione IF, in cui se il risultato della funzione è 0, si salta a una determinata locazione, altrimenti si continua con l'esecuzione del programma, segnalando una connessione attiva.

Infine, in entrambi i casi si procede con la pulizia dello stack.

Includendo librerie che implementano protocolli di rete, nel caso specifico ho esaminato il protocollo NTP, comunemente utilizzato per attacchi DDoS.

L'aggressore invia richieste ai server utilizzando l'IP del server bersaglio anziché il proprio, causando un loop di pacchetti mancanti sul server.