




ANALISI MALWARE E ASSEMBLY

1. Consegna



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

Ho iniziato verificando tramite **CFF Explorer** quali fossero le librerie importanti del malware:

- **KERNEL32.DLL** = una libreria che contiene le funzioni principali per interagire col sistema operativo o per gestire la memoria.
- **WININET.DLL** = una libreria che permette l'implementazione di protocolli di rete come HTTP, FTP, NTP. La presenza di questa libreria mi fa ipotizzare che il malware si connetta ad internet per scaricare altro oppure condividere nostre informazioni sensibili ad un dispositivo o server remoto.

2. Consegna

Malware_U3_W2_L5.exe									
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	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

Successivamente sono andato a verificare le sezioni da cui è composto il file(Malware), esse infatti ci possono dare altri suggerimenti su come funziona:

.text = dove vengono contenuti tutte le istruzioni che verranno inviate alla CPU una volta che il malware sarà attivo.

.rdata = include al suo interno informazioni sulle librerie importate ed esportate.

.data = include le variabili globali dell'eseguibile. Vengono definite globali perché sono disponibili in qualsiasi funzione dell'eseguibile.

3. Consegna

Come seconda parte ci viene dato un codice assembly in cui devo prima di tutto identificare i costrutti:

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

- Creazione dello stack.

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

- Chiamata della funzione "InternetGetConnectedState".

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

- IF, ovvero un comparazione tra due valori, che poi tramite *Jz (jump zero)* viene effettuato un jump nella cella di memoria indicata.

```
loc_40103A:
mov     esp, ebp
pop     ebp
ret     0
```

- Chiusura dello stack.

4. Consegna

Ho poi ipotizzato il suo comportamento:

Il codice in questione sembra essere una funzione scritta in linguaggio assembly x86 che gestisce lo stato della connessione Internet. Inizia con il settaggio del frame del registro **base** (*ebp*) e dello stack **pointer** (*esp*). Successivamente, prepara gli argomenti per la chiamata alla funzione *InternetGetConnectedState* mettendo dei valori sulla pila. Dopo aver ottenuto il risultato della chiamata, controlla se la connessione è attiva confrontando il risultato con zero. Se la connessione è attiva, il programma stampa un messaggio di successo tramite la funzione *sub_40105F*, altrimenti passa alla gestione degli errori, chiamando la funzione *sub_40117F*. Infine, l'epilogo ripristina lo stack pointer e il registro base, terminando la funzione.

```
push    ebp
mov     ebp, esp
push    ecx
push    0           ; duReserved
push    0           ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

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

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

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

5. Consegna

Infine ecco la spiegazione riga per riga del codice assembly:

ISTRUZIONE	Significato
push ebp	Salva il valore corrente del registro base sulla pila
mov ebp, esp	Imposta il frame del registro base con il valore dello stack pointer
push ecx	Salva il valore corrente di ecx sulla pila
push 0	Mette il valore 0 sulla pila
call ds:InternetGetConnectedState	Chiama la funzione InternetGetConnectedState
mov [ebp+var_4], eax	Memorizza il risultato della chiamata alla funzione nella variabile var_4
cmp [ebp+var_4], 0	Compara il valore memorizzato in var_4 con 0
jz short loc_40102B	Salta a loc_40102B se il risultato è zero
push offset asuccessInterne	Mette l'indirizzo del messaggio di successo sulla pila
call sub_40105F	Chiama la funzione sub_40105F per gestire il successo
add esp, 4	Pulisce gli argomenti dalla pila
mov eax, 1	Carica il valore 1 in eax
jmp short loc_40103A	Salta a loc_40103A
push offset aError1_1NoInte	Mette l'indirizzo del messaggio di errore sulla pila
call sub_40117F	Chiama la funzione sub_40117F per gestire l'errore
add esp, 4	Pulisce gli argomenti dalla pila
xor eax, eax	Esegue un'operazione XOR per azzerare eax
mov esp, ebp	Ripristina il valore dello stack pointer
pop ebp	Ripristina il valore del registro base
ret	Restituisce il controllo al chiamante