

# Analisi statica e dinamica: un approccio pratico.

S10L5

# Indice

- [Traccia](#)
- [Librerie](#)
- [Sezioni](#)
- [Costrutti](#)
- [Ipotesi](#)
- [Bonus](#)

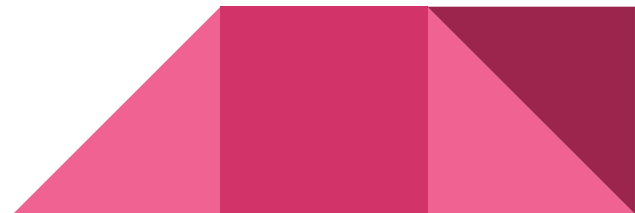
# Traccia

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. **Ipotesizzare il comportamento della funzionalità implementata**
5. **BONUS** fare tabella con significato delle singole righe di codice assembly



# Traccia

```
push    ebp
mov     ebp, esp
push    ecx
push    0           ; dwReserved
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:
push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
call    sub_40117F
add     esp, 4
xor     eax, eax
```

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

# Librerie

Malware_U3_W2_L5.exe						
Module Name	Imports	OFTs	TimeStamp	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

Una libreria in assembly è un insieme di funzioni predefinite o codice riutilizzabile scritto in linguaggio assembly e compilato in linguaggio macchina, che può essere richiamato da altri programmi per eseguire operazioni comuni come input/output, gestione della memoria, operazioni matematiche, manipolazione di stringhe, e altro ancora.

Nel nostro caso vengono importate due librerie: **KERNEL32.dll**, **WININET.dll**.

Queste librerie vengono utilizzate dal malware per interagire con il sistema operativo e con i protocolli di rete.

# Sezioni

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

Le sezioni di un malware si riferiscono alle parti o ai segmenti del codice del malware che svolgono funzioni specifiche all'interno del suo comportamento complessivo. Queste sezioni possono essere organizzate in modo da gestire diverse attività del malware, come la propagazione, l'esecuzione, la persistenza, la raccolta di informazioni, la comunicazione con un server di comando e controllo (C2), la manipolazione dei file di sistema, il danneggiamento dei dati dell'utente, e così via.

Nel nostro caso abbiamo tre sezioni:

**.text:** contiene informazioni sulle righe di codice che verranno utilizzate

**.rdata:** contiene solitamente informazioni sulle librerie e funzioni importate

**.data:** contiene informazioni sui dati e variabili presenti

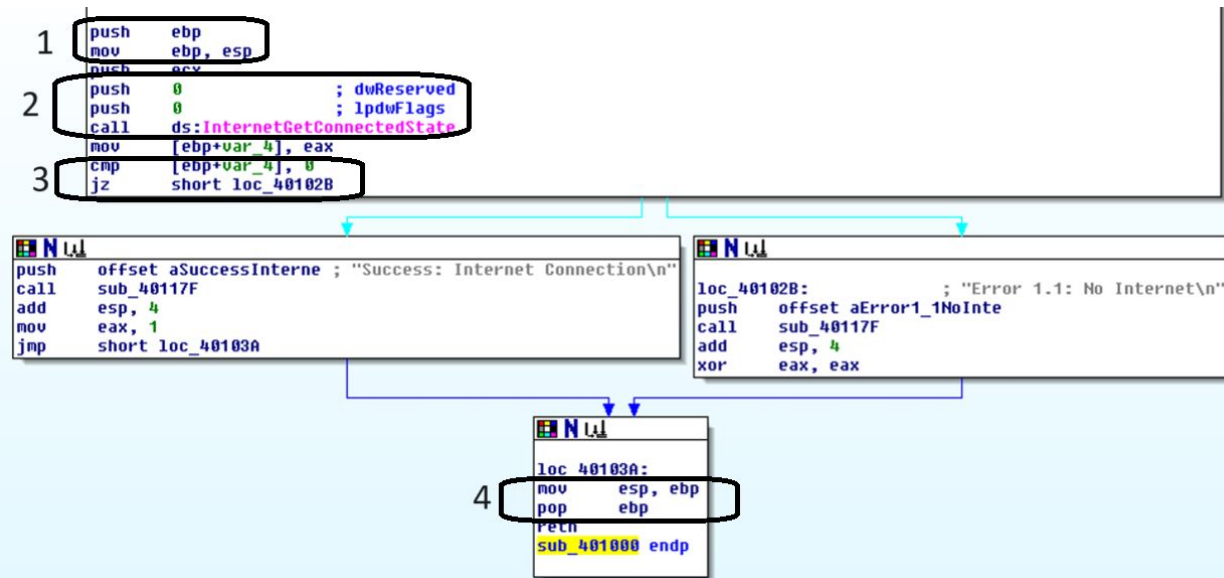


```
CA: C:\Windows\system32\cmd.exe
ExitProcess
TerminateProcess
GetCurrentProcess
UnhandledExceptionFilter
GetModuleFileNameA
FreeEnvironmentStringsA
FreeEnvironmentStringsW
WideCharToMultiByte
GetEnvironmentStrings
GetEnvironmentStringsW
SetHandleCount
GetStdHandle
GetFileType
GetStartupInfoA
GetModuleHandleA
GetEnvironmentVariableA
GetVersionExA
HeapDestroy
HeapCreate
VirtualFree
HeapFree
RtlUnwind
WriteFile
HeapAlloc
GetCPInfo
GetACP
GetOEMCP
VirtualAlloc
HeapReAlloc
GetProcAddress
LoadLibraryA
GetLastError
FlushFileBuffers
SetFilePointer
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
SetStdHandle
CloseHandle
d5e
```

## Sezioni

Come piccola ricerca in più sono andato a vedere cosa era presente nelle stringhe del malware grazie allo strumento da riga di comando **“strings”**. Questo ci consente di vedere delle stringhe importanti. Nel nostro caso possiamo vedere che carica una libreria **“LoadLibraryA”** e molte altre funzioni.

# Costrutti



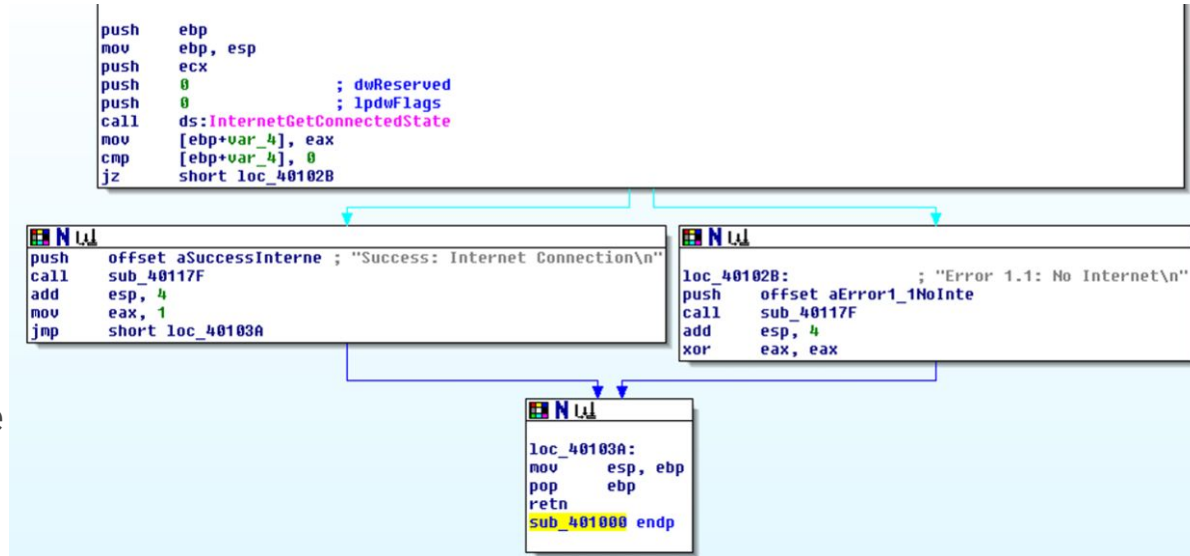
Come evidenziato in figura, vediamo 4 costrutti:

- 1) Va a creare uno stack;
- 2) Chiamata della funzione "InternetGetConnectedState";
- 3) Costrutto "If";
- 4) Rimuove lo stack



# Ipotesi

Possiamo ipotizzare da ciò che abbiamo visto che inizia con il settaggio del registro base (**ebp**) e dello stack pointer (**esp**). Poi fa la chiamata alla funzione **"InternetGetConnectedState"**. Successivamente controlla se la connessione è attiva utilizzando il costrutto **"If"**. Se la connessione è attiva allora il programma ci dice che ci si è connessi con successo altrimenti ci dice che c'è un errore di connessione. In finale abbiamo la pulizia e il ripristino dello stack pointer.



# Bonus

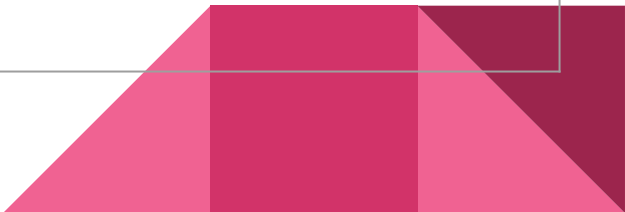
Istruzione	Descrizione
push ebp	mette il valore contenuto nel registro “ebp” sulla pila dello stack.
mov ebp, esp	“muove” il contenuto di esp in ebp. (assimilabile al copia, incolla)
push ecx	mette il valore contenuto nel registro “ecx” sulla pila dello stack.
push 0	mette il valore zero sulla pila dello stack.
call ds:InternetGetConnectedState	Chiama la funzione InternetGetConnectedState
mov [ebp+var_4], eax	memorizza il risultato della call nella variabile var_4
cmp [ebp+var_4], 0	compara il valore 0 con quello nella variabile var_4

# Bonus

Istruzione	Descrizione
jz short loc_40102B	Salta alla “location” 40102B se il risultato è zero
push offset aSuccessInternet	mette il messaggio sulla pila
call sub_40117F	chiama la subroutine 40117F
add esp, 4	aggiunge 4 byte alla pila, “pulendo” la pila
mov eax, 1	“muove” il valore 1 in eax
jmp short loc_40103A	salta (senza condizioni) alla location 40103A
push offset aError1_1NoInte	mette il messaggio di errore sullo stack

# Bonus

Istruzione	Descrizione
call sub40117F	chiama la subroutine 40117F
add esp, 4	aggiunge 4 byte alla pila, “pulendo” la pila
xor eax, eax	esegue un'operazione xor per azzerare eax
mov esp, ebp	ripristino del valore dello stack pointer
pop ebp	al contrario di push, elimina ebp
retn	ritorna il valore al chiamante
end	end





Grazie