

Progetto S10 L5

Davide Andreozzi

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

1) Librerie

Aperto il malware con CFF Explorer possiamo andare nella sezione relativa alle librerie importate «Import Directory» e verificare le librerie presenti:

- KERNEL32.dll

Questa libreria controlla le funzioni principali del sistema operativo, ovvero, gestione della memoria, processi, file, gestione degli eventi, ecc.

- WININET.dll

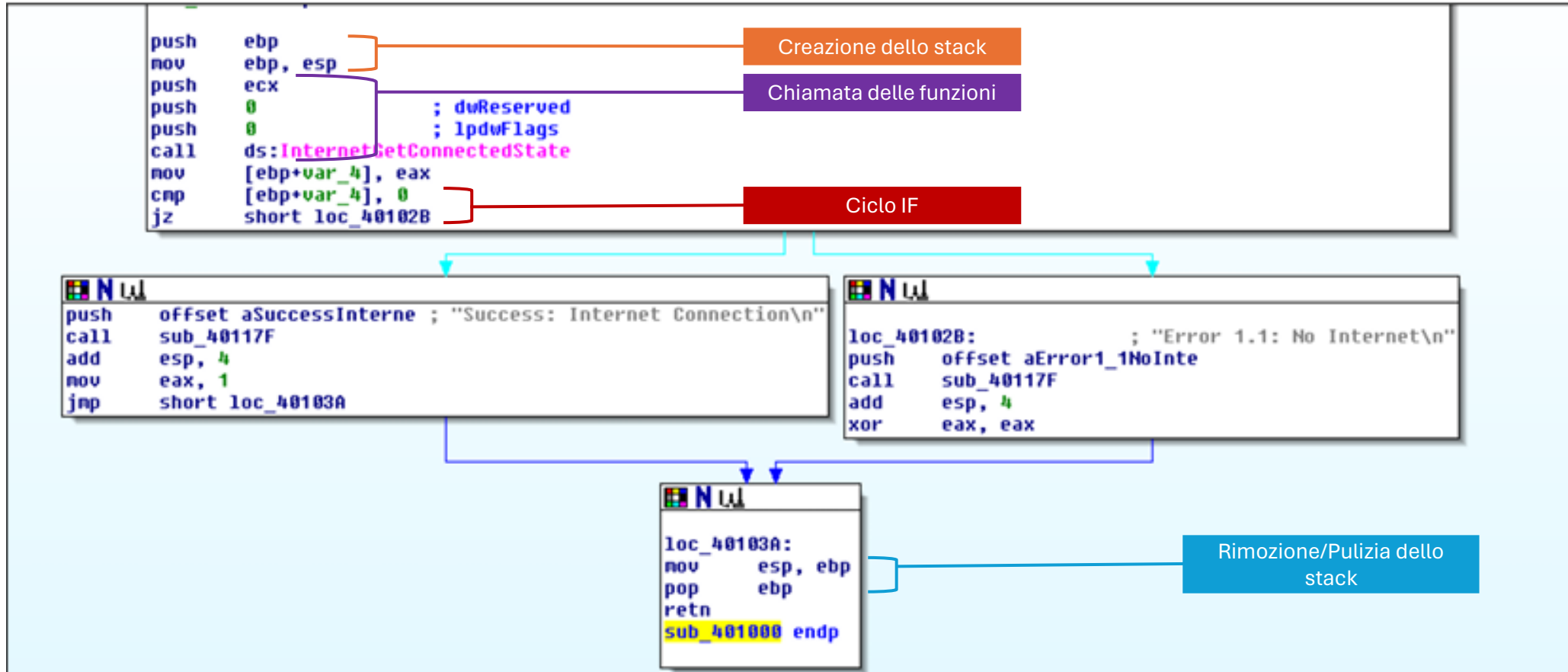
Questa libreria gestisce le funzioni per stabilire e gestire le connessioni con i protocolli FTP e HTTP, un programma che usa questa libreria può stabilire connessioni di rete

2) Sezioni

Sempre all'interno di CFF Explorer andiamo nella sezione «Section Headers» per vedere le sezioni presenti nell'eseguibile:

- `.text`
In questa sezione sono contenute le istruzioni che la CPU eseguirà quando il programma viene avviato. È come se fosse il cuore del programma.
- `.rdata`
In questa sezione sono incluse le informazioni delle librerie e le funzioni importate ed esportate dal programma. Ad esempio, stringhe di testo per messaggi di errore o avviso, tabelle e altri dati che rimangono invariati durante l'esecuzione del programma.
- `.data`
Questa sezione contiene i dati e le variabili globali del programma, cioè quelle variabili che sono state dichiarate globalmente per essere accessibili da qualsiasi funzione.

3) Costrutti noti



4) Funzionalità

Il programma della slide precedente chiama la funzione ***InternetGetConnectedState*** che va a verificare lo stato della connessione.

Dopo la chiamata della funzione il programma memorizzerà all'interno della variabile **[ebp+var_4]** il valore restituito dalla chiamata alla funzione.

Tramite (cmp) viene eseguito un confronto, se il risultato del valore della variabile è uguale a 0 il programma farà un salto condizionale (jmp) alla locazione **loc_40102B** e mostrerà il messaggio ***Error 1.1: No internet.***

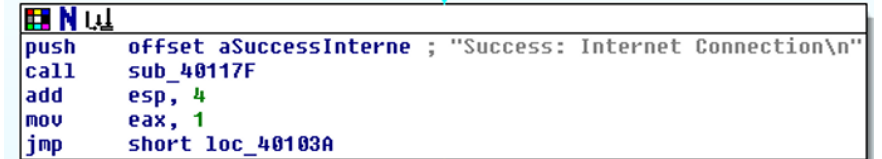
Se il risultato è diverso da 0 il programma continuerà la sua esecuzione e mostrerà il messaggio ***Success: Internet Connection.***

In sostanza il programma va a verificare lo stato della connessione internet e comunica all'utente se la connessione è attiva o meno.

5) Bonus – Spiegazione riga per riga

Codice	Spiegazione
push ebp	Salva il valore corrente di EBP sullo stack
mov ebp, esp	Copia il valore del registro ESP in EBP
push ecx	Salva il valore corrente di ECX sullo stack
push 0 ; dwReserved	Imposta a 0 il valore sullo stack per la funzione in oggetto
push 0 ; lpdwFlags	Imposta a 0 il valore sullo stack per la funzione in oggetto
call ds:InternetGetConnectedState	Chiamata alla funzione InternetGetConnectedState
mov [ebp+var_4], eax	Copia il valore del registro EAX dato dalla chiamata precedente nella variabile locale [ebp+var_4]
cmp [ebp+var_4], 0	Esegue un confronto (sub) tra il valore il valore contenuto nella variabile locale e il valore sorgente (0). Di conseguenza ne modifica il ZF
jz short loc_401028	Se il ZF=1 avverrà un salto alla locazione descritta

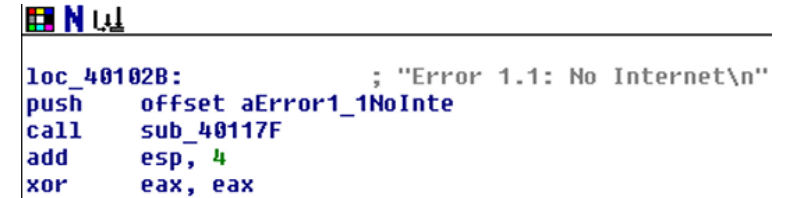
5) Bonus – Spiegazione riga per riga

A screenshot of a debugger window showing assembly code. The code is as follows:

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

Codice	Spiegazione
push offset aSuccessInterne ; "Success: Internet Connection\n"	Chiama l'istruzione che verrà eseguita successivamente dalla chiamata
call sub_40117F	Chiama la subroutine in oggetto, probabilmente la stampa del messaggio di successo della connessione
add esp, 4	Fa una somma aggiungendo il valore 4 al registro ESP e ne salva il risultato
mov eax, 1	Copia il valore 1 nel registro EAX
jmp short_loc_40103A	Fa un salto all'indirizzo loc_40103A

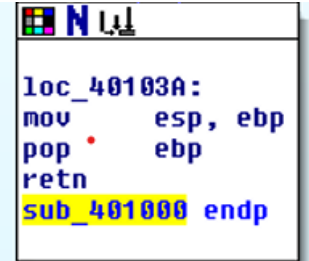
5) Bonus – Spiegazione riga per riga



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

Codice	Spiegazione
loc_40182B:	Rappresenta l'indirizzo di memoria
push offset aError1NoInte ; "Error 1.1: No Internet\n"	Chiama l'istruzione che verrà eseguita successivamente dalla chiamata
call sub_40117F	Chiama la subroutine in oggetto, probabilmente la stampa del messaggio di assenza di connessione
add esp, 4	Somma il valore 4 al valore di ESP e ne salva il contenuto
xor eax,eax	Questa è un operazione dell'operatore XOR, e va a inizializzare a 0 il registro. XOR restituisce 1 se i due bit sono diversi, quindi possiamo quindi dire che quando opera tra se stesso il risultato sarà sempre 0

5) Bonus – Spiegazione riga per riga

A screenshot of a debugger window titled "NUL" showing assembly code. The code is as follows:
loc_40103A:
mov esp, ebp
pop ebp
retn
sub_401000 endp
The line "sub_401000 endp" is highlighted in yellow.

Codice	Spiegazione
loc_40103A:	Rappresenta l'indirizzo di memoria
Mov esp , ebp	Copia il valore di EBP in ESP
pop ebp	Ripristina il valore EBP a quello dell'inizio della funzione
retn	Ritorna all'istruzione successiva
sub_40103A endp	Indica la fine della funzione