

Consegna U3 S10 L5

Analisi Malware

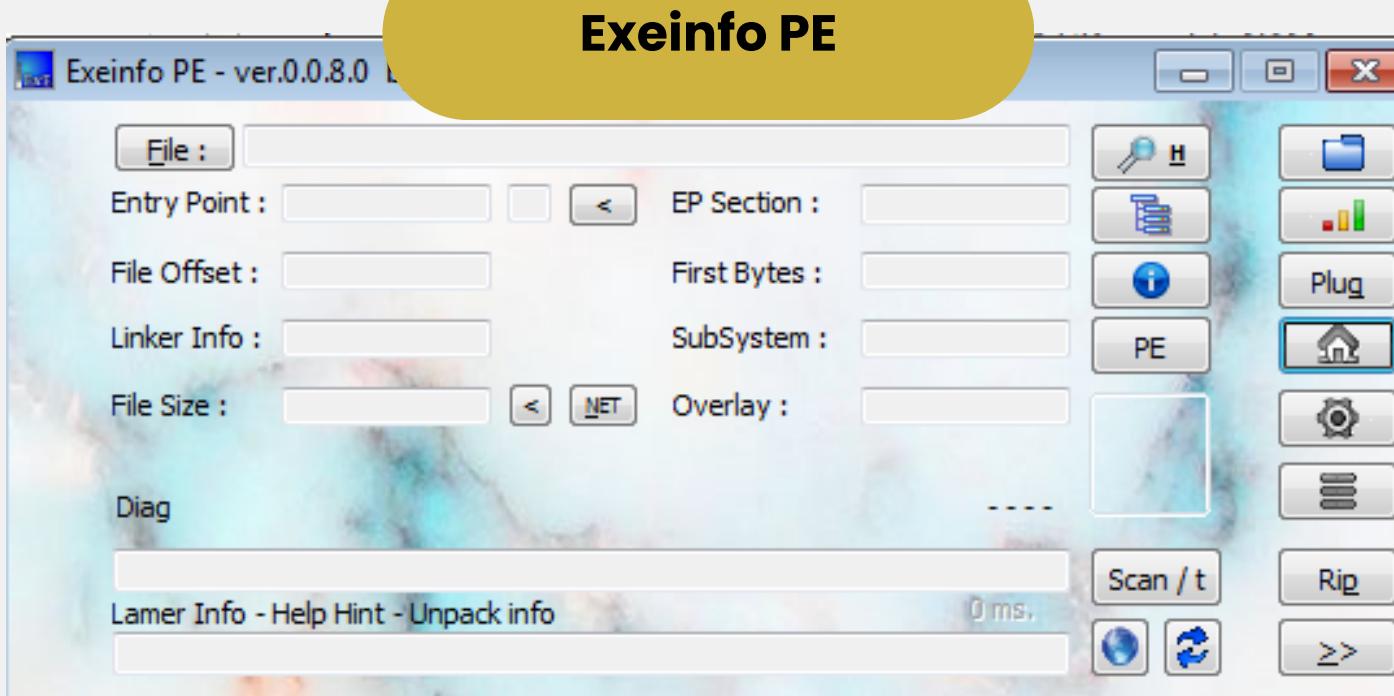
Preparazione ambiente di analisi



E' fondamentale procedere per step onde evitare errori nell'analisi e danni all'ambiente di lavoro. Dopo aver isolato la macchina **guest** rispetto alla machina **host**, disattivando l'accesso ad **internet**, alle **cartelle condivise** ed alle porte **USB** nonchè aver settato la connessione su **interna** da VirtualBox, si può iniziare avviando i tools necessari all'analisi. In questo caso è richiesta la verifica delle librerie necessarie al funzionamento del malware.

I tools necessari da avviare sono:

- **CFF Explorer**
- **Exeinfo PE**



Traccia n.1

Librerie importate

Librerie importate



Ecco l'elenco delle librerie importate dal malware per funzionare:

- **KERNEL32.dll**: libreria usate dalle applicazioni per comunicare col sistema operativo. Essa si occupa della gestione memoria (RAM), gestione Input/Output, gestione dei processi e degli errori. Fa da tramite, posizionandosi a metà fra l'hardware e l'esecuzione software dei vari programmi.
- **WININET.dll**: si occupa delle funzioni di rete, come le connessioni **FTP**, **HTTP** e **HTTPS**. Gestisce anche la funzionalità della cache e dei cookies.

Ci sono poi numerose chiamate a delle funzioni all'interno delle sudette librerie.

Alcuni esempi contenuti in **KERNEL32.dll**:

- **GetProcAddress**: recupera l'indirizzo di una funzione esportata (nota anche come routine) o variabile dalla libreria di collegamento dinamico specificata (DLL).
- **SetStdHandle**: imposta l'handle per il dispositivo standard specificato (input standard, output standard o errore standard).
- **GetModuleFileNameA**: Recupera il percorso completo per il file contenente il modulo specificato. Il modulo deve essere stato caricato dal processo corrente.
- **GetStartupInfoA**: si occupa di gestire correttamente la creazione di una finestra per un processo appena avviato.
- **GetEnvironmentVariableA**: recupera il contenuto della variabile specificata dal blocco di ambiente del processo chiamante.
- **GetCPIInfo**: recupera informazioni su qualsiasi tabella codici installata o disponibile valida.

Librerie importate



Alcuni esempi contenuti in **WININET32.dll**:

- **InternetOpenUrlA**: apre una risorsa specificata da un URL FTP o HTTP completo.
- **InternetCloseHandle**: chiude un singolo handle Internet.
- **InternetGetConnectedState**: recupera lo stato di connessione del sistema locale.
- **InternetOpenA**: inizializza l'uso di un'applicazione delle funzioni WinINet.

Librerie importate



CFF Explorer VIII - [Malware_U3_W2_L5.exe]

File Settings ?

Malware_U3_W2_L5.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name F
000065EC	N/A	000064DC	000064E0	000064E4	000064E
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065E
WININET.dll	5	000065CC	00000000	00000000	0000666

File: Malware_U3_W2_L5.exe

- Dos Header
- Nt Headers
 - File Header
 - Optional Header
 - Data Directories [x]
- Section Headers [x]
- Import Directory
- Address Converter
- Dependency Walker
- Hex Editor
- Identifier
- Import Adder
- Quick Disassembler
- Rebuilder
- Resource Editor
- UPX Utility

OFTs FTs (IAT) Hint Name

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
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA
000068E6	000068E6	01E4	MultiByteToWideChar
00006670	00006670	00CA	GetCommandLineA
00006682	00006682	0174	GetVersion
00006690	00006690	007D	ExitProcess
0000669E	0000669E	029E	TerminateProcess
000066B2	000066B2	00F7	GetCurrentProcess

CFF Explorer VIII - [Malware_U3_W2_L5.exe]

File Settings ?

Malware_U3_W2_L5.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderCh
00006664	N/A	000064F0	000064F4	000064F8
szAnsi	(nFunctions)	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000
WININET.dll	5	000065CC	00000000	00000000

File: Malware_U3_W2_L5.exe

- Dos Header
- Nt Headers
 - File Header
 - Optional Header
 - Data Directories [x]
- Section Headers [x]
- Import Directory
- Address Converter
- Dependency Walker
- Hex Editor
- Identifier
- Import Adder
- Quick Disassembler
- Rebuilder
- Resource Editor
- UPX Utility

OFTs FTs (IAT) Hint Name

OFTs	FTs (IAT)	Hint	Name
000065CC	000060B4	00006640	00006642
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

Traccia n.2

Sezioni del malware

Sezioni del malware



Per ottenere le sezioni dell'eseguibile malevolo si possono usare 2 tools:

- **CFF Explorer**
- **Exeinfo PE**

In entrambi i casi si ottengono i dettagli necessari a procedere con l'analisi.

The screenshot shows the CFF Explorer interface with the file 'Malware_U3_W2_L5.exe' loaded. On the left, a tree view shows the file structure with nodes for Dos Header, Nt Headers, File Header, Optional Header, Data Directories, and Section Headers. The 'Section Headers [x]' node is highlighted with a red box. On the right, a table displays section details:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000

The screenshot shows the Sections viewer interface with the file 'Malware_U3_W2_L5.exe' loaded. It displays 3 sections with the following details:

Nr	Virtual ...	Virtual s...	RAW D...	RAW size	Flags	Name	First bytes (hex)	Fir...	sect. Stats
01 ep	00001000	00004A78	00001000	00005000	60000020	.text	55 8B EC 51 6A 00 6A 00 FF	U ...	Not packed - 17,959 % ZERO
02 im	00006000	0000095E	00006000	00001000	40000040	.rdata	E4 65 00 00 40 69 00 00 2E	e ...	Very not packed - 53,6621 % ZERO
03	00007000	00003F08	00007000	00003000	C0000040	.data	00 00 00 00 00 00 00 00 00	...	Very not packed - 93,9616 % ZERO

Sezioni del malware



Le sezioni sono tre:

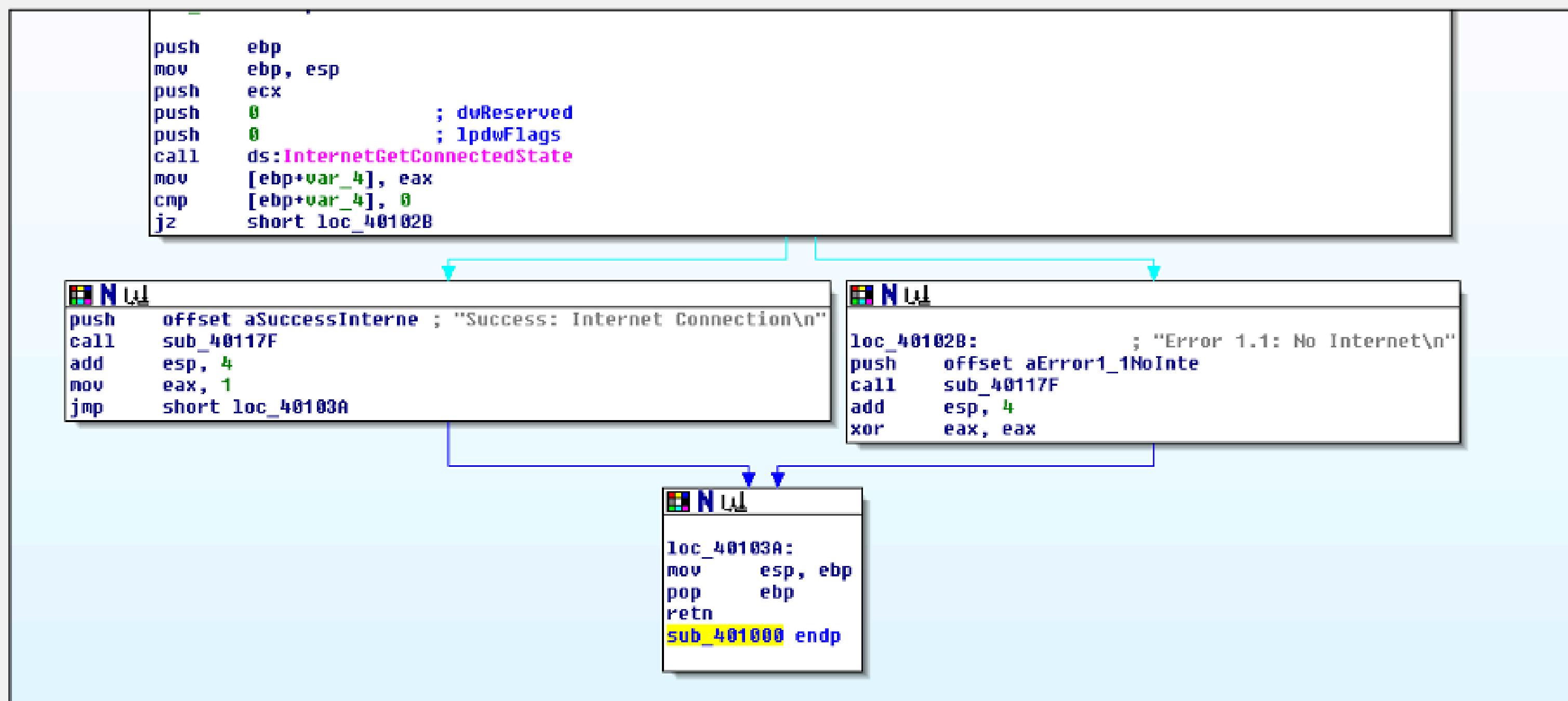
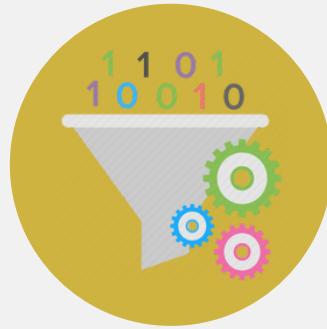
- **.text:** contiene il codice del programma che verrà eseguito dalla CPU.
- **.data:** contiene le variabili globali dell'eseguibile
- **.rdata:** contiene le librerie importate ed esportate dal programma nonché le funzioni importate dalle librerie. Quindi ad esempio, le librerie viste pocanzi: **KERNEL32.dll** e **WININET.dll**.

Si notano anche due colonne chiamate **Virtual Size** e **Raw Size**. La prima indica la dimensione della sezione del malware dopo il caricamento nella memoria di sistema (RAM), mentre la seconda ci dà informazioni sulla dimensione della sezione sul disco di sistema.

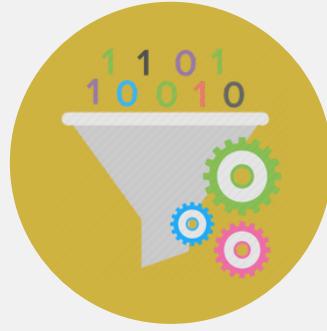
In questo caso le varie sezioni sono ben visibili. In altri casi, capita che per non permetterne l'analisi, le sezioni vengano compresse e criptate con vari strumenti (Morphin, UPX ...). E' possibile poi estrarle con un procedimento più complesso.

**Traccia n.3-5
Identificazione
costrutti e analisi
codice**

Identificazione costrutti



Identificazione costrutti



Creazione dello stack

- Viene eseguito il push del **Base Pointer (puntatore alla base dello stack)** sullo stack.
- Viene copiato lo **Stack Pointer (puntatore alla cima dello stack)** nel **Base Pointer**.

push
mov

ebp
ebp, esp

Caricamento parametri di funzione

- Viene eseguito il **push** del **registro ecx (count register, tiene il conto durante le operazioni che si reiterano più volte, es. cicli for)** sullo stack.
- Viene eseguito il **push** del **valore 0** sullo **stack**.
- Viene eseguito il **push** di un secondo **valore 0** sullo **stack**.

push
push
push
ecx
0
0

; duReserved
; lpdwFlags

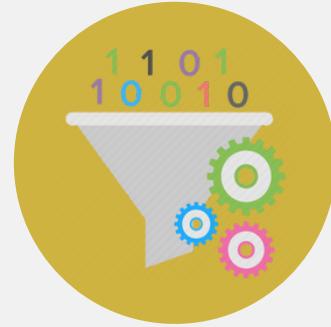
Chiamata di funzione

Viene chiamata la funzione **InternetGetConnectedState** che si occupa di verificare lo stato della connessione del sistema.

ds è un **segment register**: esso contiene dati e costanti.

call ds:InternetGetConnectedState

Identificazione costrutti



Operazione di copia

Viene eseguita la copia del contenuto del **registro eax** (**primary accumulator**, usato per l'Input/Output delle operazioni aritmetiche) all'interno della **variabile var_4**.

mov [ebp+var_4], eax

Costrutto IF in assembly

- Viene eseguito il confronto tra il contenuto della **variabile var_4** e il **valore 0**. L'operazione eseguita nella realtà è **sub [ebp+var_4], 0**, cioè la differenza tra la **destinazione**, ovvero la variabile, e la **sorgente**, ovvero il valore numerico. Se **destinazione = sorgente** viene settato **ZF = 1** mentre **CF = 0**. Se **destinazione > sorgente** allora **ZF = 0** e **CF = 0**. Se **destinazione < sorgente** allora **ZF = 0** e **CF = 1**. **ZF** e **CF** sono detti **EFLAGS** e sono utili nel caso di costrutti di questo tipo per poi implementare i **salti condizionali**.
- **jz** esegue i **salti condizionali**. Serve per saltare ad uno specifico indirizzo di memoria al verificarsi di una determinata condizione. In questo caso il salto all'indirizzo di memoria **40102B** avviene se **var_4 = 0**.

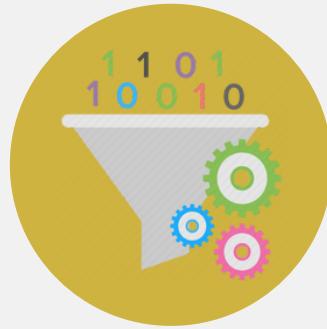
cmp [ebp+var_4], 0
jz short loc_40102B

Stampa a schermo di una stringa (salto all'indirizzo 40102B – primo blocco if)

- Viene seguito il **push** sullo **stack** dell'offset della **variabile stringa aError1_1NoInte** da passare come parametro alla funzione di stampa.
- Poi viene chiamata la funzione di stampa **sub_40117F**, per scrivere a schermo il messaggio di errore per l'utente.

loc_40102B: ; "Erro
push offset aError1_1NoInte
call sub_40117F

Identificazione costrutti



Eliminazione variabili dallo stack e pulizia dei registri

- La prima istruzione somma il **valore 4** al valore del registro dello **Stack Pointer** per eliminare dallo **stack** la variabile stringa precedentemente pushata. Il **valore 4** corrisponde ai **4 byte** di spazio che sono la dimensione della variabile, ovvero **32 bits**.
- La seconda istruzione esegue l'operazione **xor (exclusive or)** tra il **registro eax** ed il registro stesso per azzerarne il contenuto. Si usa lo **xor** per velocizzare l'esecuzione dato che copiare il **valore 0** nel registro richiederebbe qualche ciclo di clock di troppo. La seconda ragione consiste nel fatto che lo **xor** tra due valori uguali fa sempre **0**.

add esp, 4
xor eax, eax

Caricamento parametri di funzione e stampa a schermo (codice eseguito se non avviene il salto - secondo blocco dopo if)

- Si effettua il **push** della variabile stringa **aSuccessInterne** sullo **stack** che servirà da parametro alla funzione di stampa che verrà successivamente chiamata.
- Viene effettuata la chiamata alla funzione di stampa **sub_40117F** che stamperà il messaggio di avvenuta connessione a schermo.

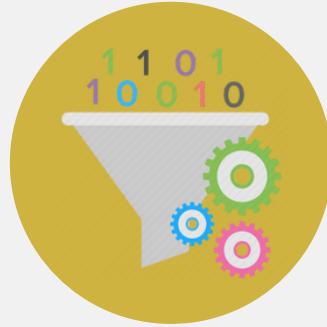
push call offset aSuccessInterne
sub_40117F

Eliminazione variabili dallo stack

- Si esegue l'**add** del **valore 4** allo **Stack Pointer** per far sì che dallo **stack** venga eliminata la variabile stringa **aSuccessInterne** precedentemente usata come parametro della funzione di stampa.
- Poi viene copiato il **valore 1** all'interno del **registro eax**.

add esp, 4
mov eax, 1

Identificazione costrutti



Salto al successivo blocco di codice da eseguire

- Viene eseguito il **jmp** all'indirizzo di memoria **40103A** per l'esecuzione delle successive istruzioni del programma.

```
jmp short loc_40103A
```

Ripristino dello stack post-esecuzione (blocco finale)

- Qui viene eseguita la pulizia finale dello **stack** per creare lo spazio alle prossime istruzioni che verranno eseguite. Praticamente viene copiato il registro del **Base Pointer** nel registro dello **Stack Pointer**.
- Nella seconda istruzione viene eseguito il **pop** del registro del **Base Pointer**, ovvero viene eliminato dallo **stack**.

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

Fine dell'esecuzione

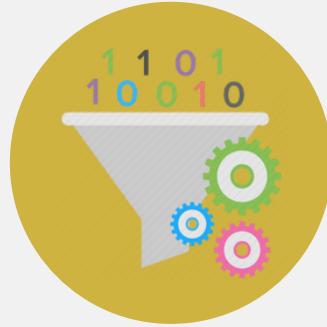
- L'istruzione **ret_n** permette di ritornare al programma chiamante al termine di una procedura, cioè una funzione chiamata per mezzo dell'istruzione **call**.
- L'istruzione **endp** è detta **direttiva**, e serve per terminare la funzione (o procedura) **sub_401000** dopo che ha finito la sua esecuzione.

```
retn  
sub_401000 endp
```

Traccia n.4

Ipotesi sul comportamento

Ipotesi sul comportamento



Dopo aver eseguito l'analisi completa del codice, riga per riga, ed aver riconosciuto i costrutti del linguaggio C all'interno di esso, si può fare una ipotesi sul funzionamento globale del programma.

Il codice analizzato si occupa di chiamare una funzione che serve per effettuare un controllo sullo stato di connessione alla rete del sistema. Questa funzione lavora con tre parametri. All'interno si trova un **costrutto if** che a seconda del valore di ritorno della funzione eseguirà l'una o l'altra porzione di codice che segue.

Se la connessione è stata stabilita correttamente, il programma stamperà a schermo il messaggio per avvisare l'utente che è avvenuta con successo, altrimenti stamperà un messaggio di errore.

Infine viene terminata l'esecuzione.

Grazie

Gianluca Sansone