



S11/L2

A N A L I S I S T A T I C A
A V A N Z A T A



Flavio Scognamiglio

Traccia

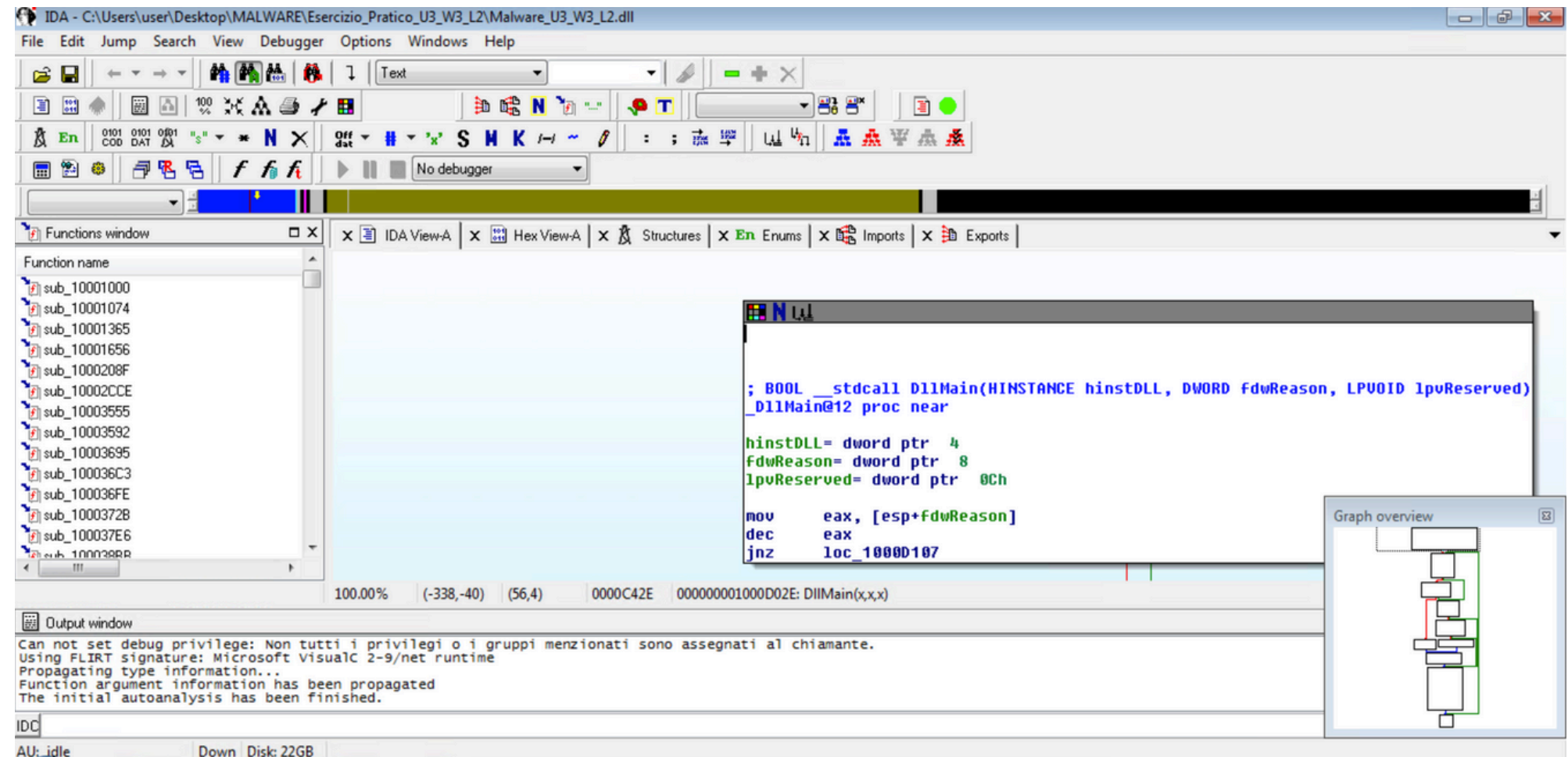
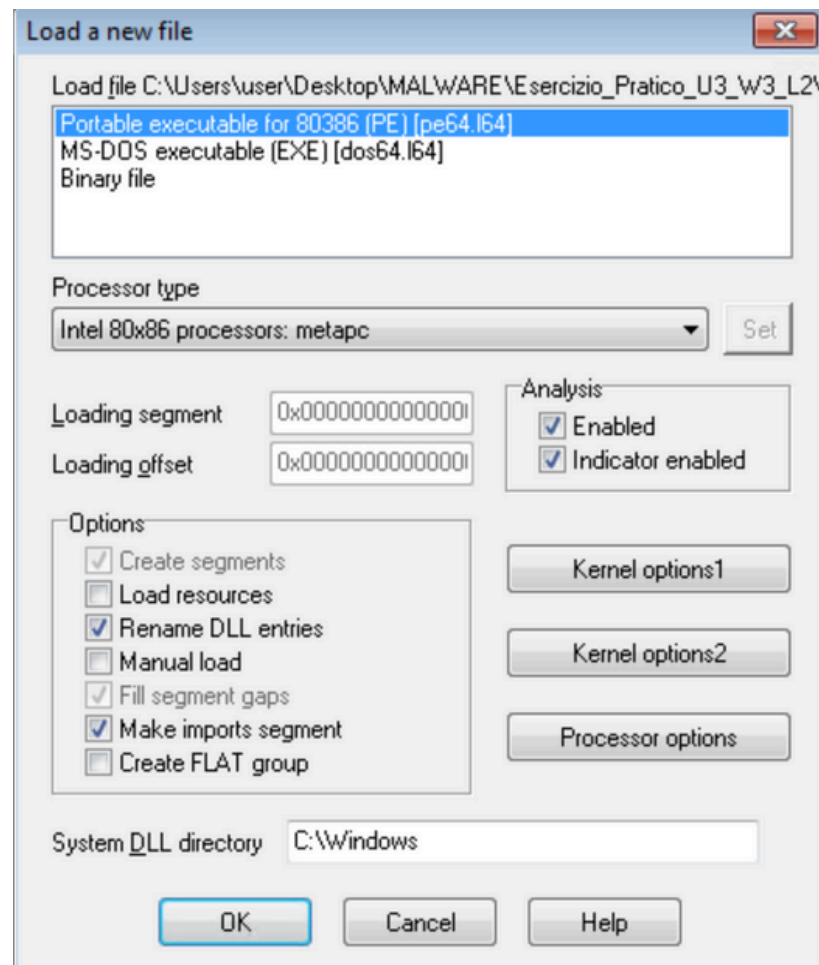
Lo scopo dell'esercizio di oggi è di acquisire esperienza con IDA, un tool fondamentale per l'analisi statica. A tal proposito, con riferimento al malware chiamato «**Malware_U3_W3_L2** » presente all'interno della cartella «**Esercizio_Pratico_U3_W3_L2**» sul Desktop della macchina virtuale dedicata all'analisi dei malware, rispondere ai seguenti quesiti, utilizzando IDA Pro.

1. Individuare l'indirizzo della funzione DLLMain (così com'è, in esadecimale)
 2. Dalla scheda «imports» individuare la funzione «gethostbyname ». Qual è l'indirizzo dell'import? Cosa fa la funzione?
 3. Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?
 4. Quanti sono, invece, i parametri della funzione sopra?
 5. Inserire altre considerazioni macro livello sul malware (comportamento)
-

IDA PRO

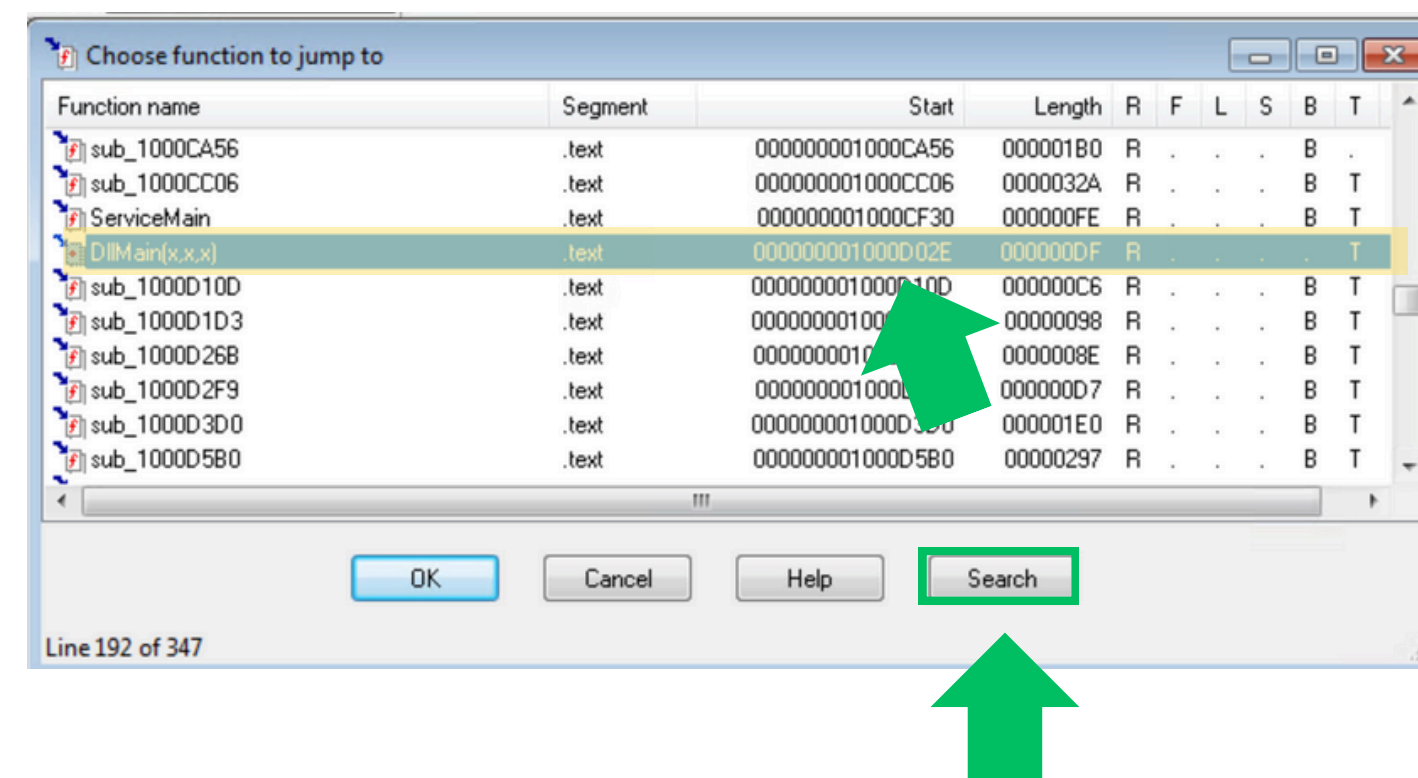
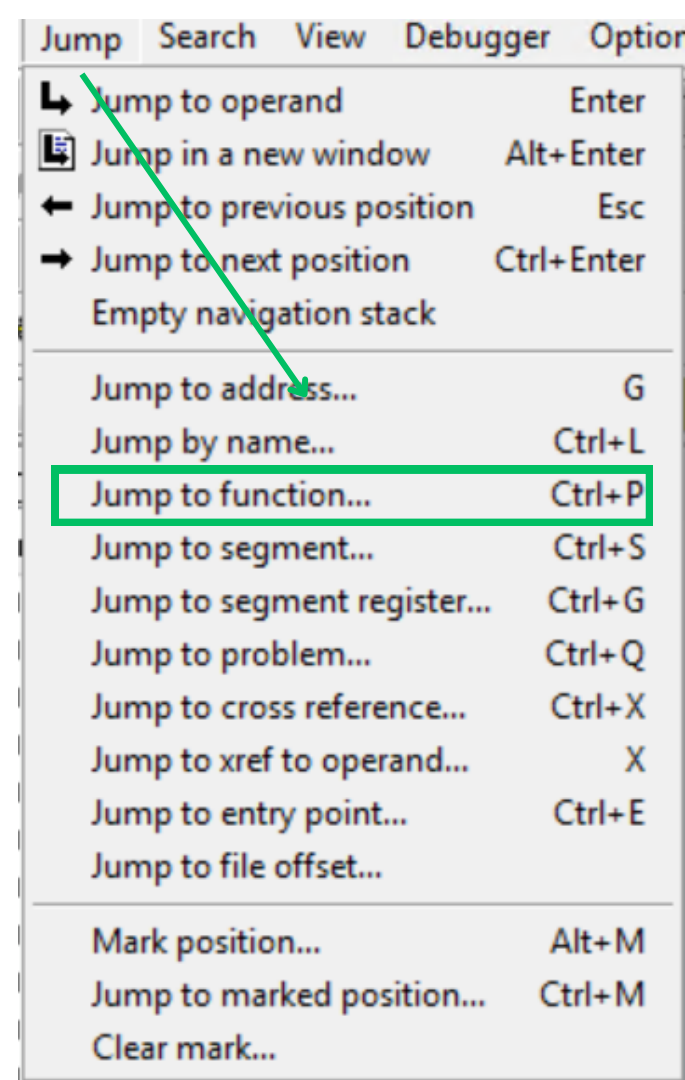
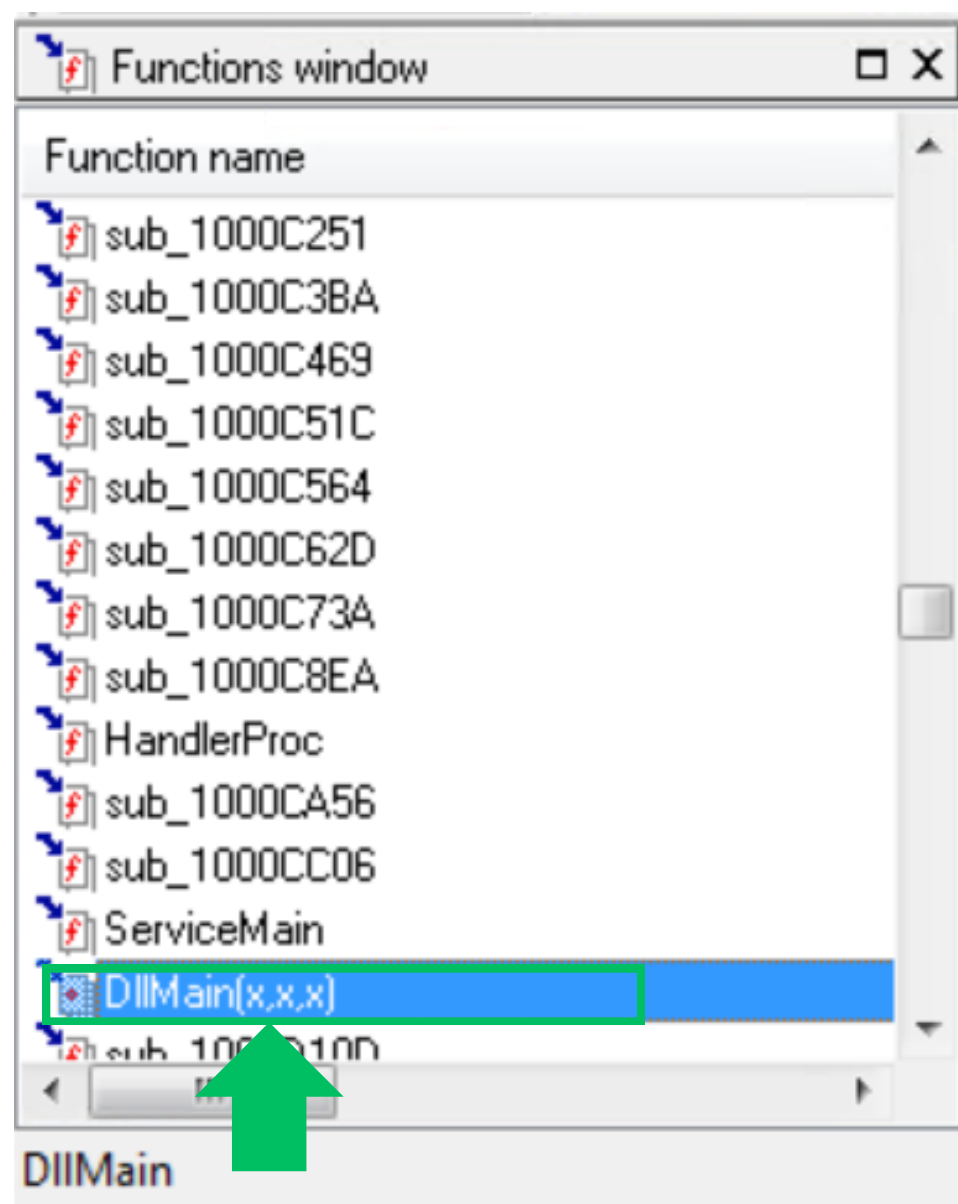
In questo esercizio userò IDA Pro, un potente disassembler che supporta diversi formati di file. Utilizzerò la versione 5.5.0 fornita dalle macchine virtuali del professore. **Tutti i risultati saranno basati su questa versione specifica.**

Come prima prima cosa, **avvio** il software e gli do in pasto il file in questione:



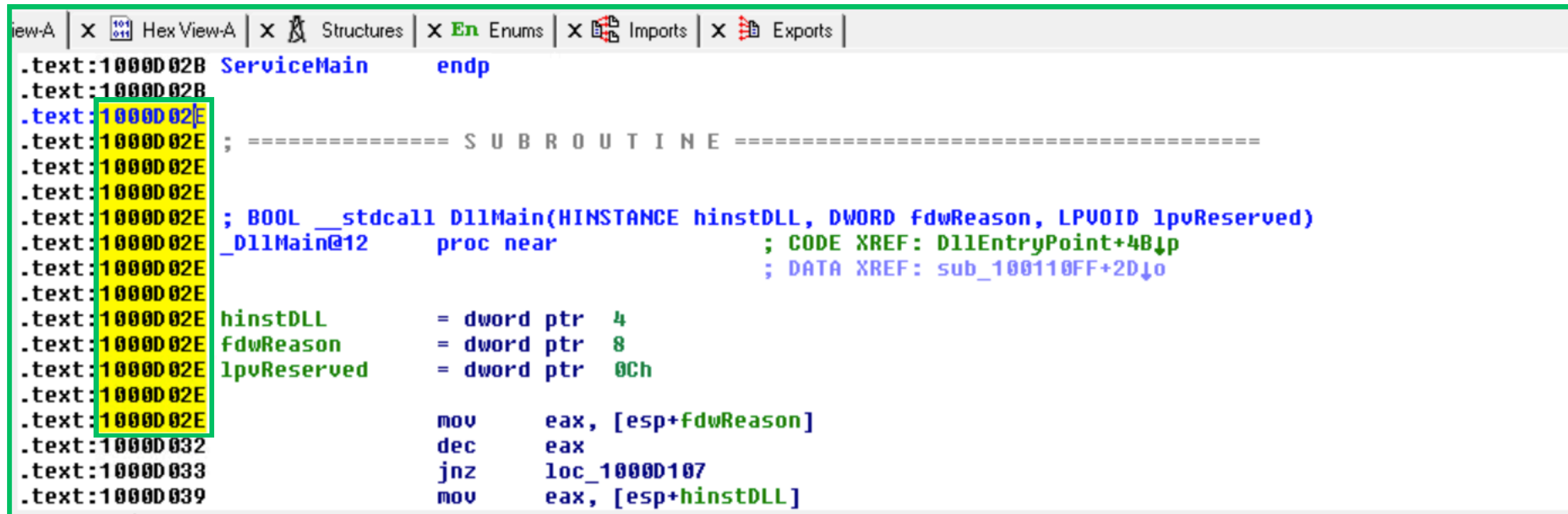
1 - Individuare l'indirizzo della funzione DLLMain

Grazie al grafico a blocchi, possiamo individuare facilmente la funzione **DllMain**. Tuttavia, esistono altri metodi per trovarla, come effettuare una ricerca diretta per il nome della funzione. Per fare ciò, è possibile utilizzare il comando '**Jump to Function**' dal menu, oppure la combinazione CTRL+F sul pannello della lista funzioni a sinistra:



1 - Individuare l'indirizzo della funzione DLLMain

L'indirizzo della funzione DLLMain quindi è **1000D02E**. Infatti, cliccando col destro del mouse sul riquadro relativo alla funzione Main nel Graph Overview, e digitando la barra spaziatrice, il programma ci porta **all'interfaccia di analisi testuale relativa a quella funzione**. E da come si può notare, tutte le istruzioni di quella funzione corrispondono all'indirizzo di cui sopra:



```
iew-A | X 101 Hex View-A | X Structures | X En Enums | X Imports | X Exports |
.text:1000D02B ServiceMain      endp
.text:1000D02B
.text:1000D02E ; ===== S U B R O U T I N E =====
.text:1000D02E
.text:1000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:1000D02E _DllMain@12      proc near      ; CODE XREF: DllEntryPoint+4B↓p
.text:1000D02E                                     ; DATA XREF: sub_100110FF+2D↓o
.text:1000D02E hinstDLL      = dword ptr  4
.text:1000D02E fdwReason    = dword ptr  8
.text:1000D02E lpvReserved  = dword ptr 0Ch
.text:1000D02E
.text:1000D02E      mov     eax, [esp+fdwReason]
.text:1000D032      dec     eax
.text:1000D033      jnz     loc_1000D107
.text:1000D039      mov     eax, [esp+hinstDLL]
```


2 - Indirizzo dell'Import e Funzione di gethostbyname

La traccia ci chiede di andare nella scheda imports, dove vi sono tutte le funzioni importate, e ricercare nello specifico l'indirizzo relativo all'import della funzione **gethostbyname**. In questo caso, con ALT+T o in alto nel menu search, ho ricercato la stringa in questione:

The image illustrates the steps to find the address of the `gethostbyname` function in a debugger's imports section.

1 The **Imports** tab is selected in the debugger window. The table below shows the list of imported functions and their libraries.

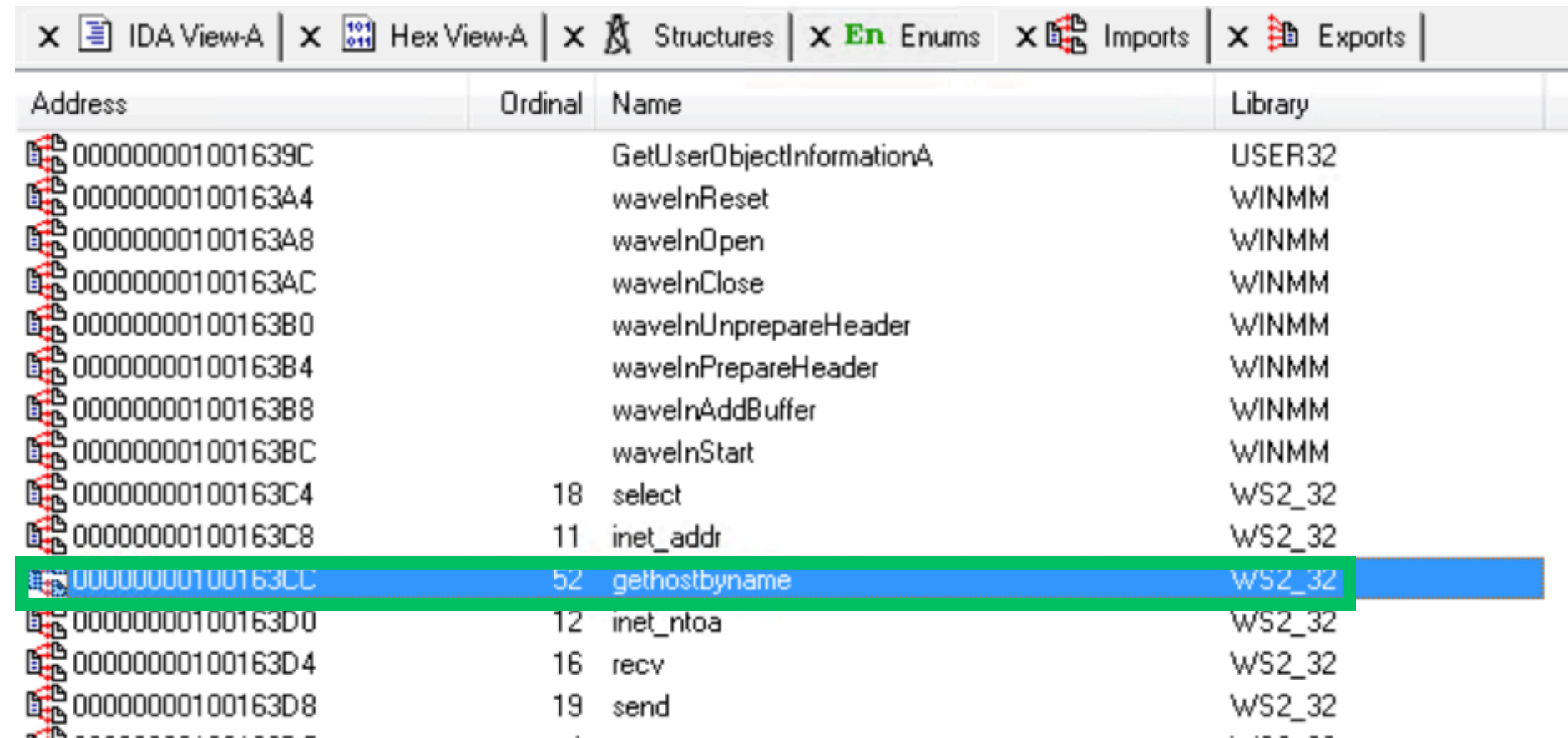
Address	Ordinal	Name	Library
000000...		LookupPrivilegeValueA	ADVAPI32
000000...		OpenProcessToken	ADVAPI32
000000...		RegCloseKey	ADVAPI32
000000...		RegQueryValueExA	ADVAPI32
000000...		RegOpenKeyExA	ADVAPI32
000000...		CreateProcessAsUserA	ADVAPI32
000000...		RegSetValueExA	ADVAPI32
000000...		RegDeleteValueA	ADVAPI32
000000...		RegEnumKeyA	ADVAPI32
000000...		RegOpenKeyA	ADVAPI32
000000...		SetTokenInformation	ADVAPI32
000000...		DuplicateTokenEx	ADVAPI32
000000...		RegEnumValueA	ADVAPI32
000000...		AdjustTokenPrivileges	ADVAPI32

2 The **Search** menu is open, showing the **Search...** option (Alt+T).

3 The **Enter the search substring** dialog box is shown, with the search string `gethostbyname` entered in the **String** field.

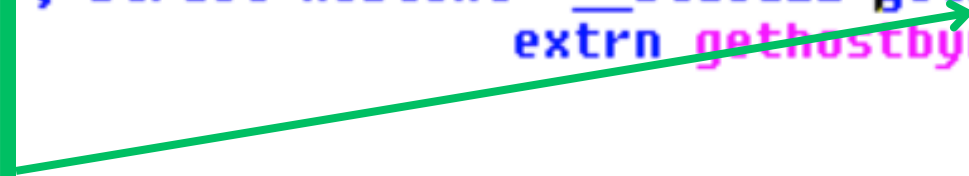
2 - Indirizzo dell'Import e Funzione di gethostbyname

A questo punto otteniamo l'indirizzo richiesto dalla traccia, ovvero 100163CC.



Address	Ordinal	Name	Library
000000001001639C		GetUserObjectInformationA	USER32
00000000100163A4		waveInReset	WINMM
00000000100163A8		waveInOpen	WINMM
00000000100163AC		waveInClose	WINMM
00000000100163B0		waveInUnprepareHeader	WINMM
00000000100163B4		waveInPrepareHeader	WINMM
00000000100163B8		waveInAddBuffer	WINMM
00000000100163BC		waveInStart	WINMM
00000000100163C4	18	select	WS2_32
00000000100163C8	11	inet_addr	WS2_32
00000000100163CC	52	gethostbyname	WS2_32
00000000100163D0	12	inet_ntoa	WS2_32
00000000100163D4	16	recv	WS2_32
00000000100163D8	19	send	WS2_32

```
.idata 100163CC ; struct hostent *__stdcall gethostbyname(const char *name)
.idata 100163CC      extrn gethostbyname:dword
.idata 100163CC      ; CODE XREF: sub_10001074:
.idata 100163CC      : sub 10001074+1D3↑p ...
```



2 - Cosa fa la funzione gethostbyname?

In questo ci viene in soccorso la documentazione Microsoft: <https://learn.microsoft.com/en-us/dotnet/api/system.net.dns.gethostbyname>

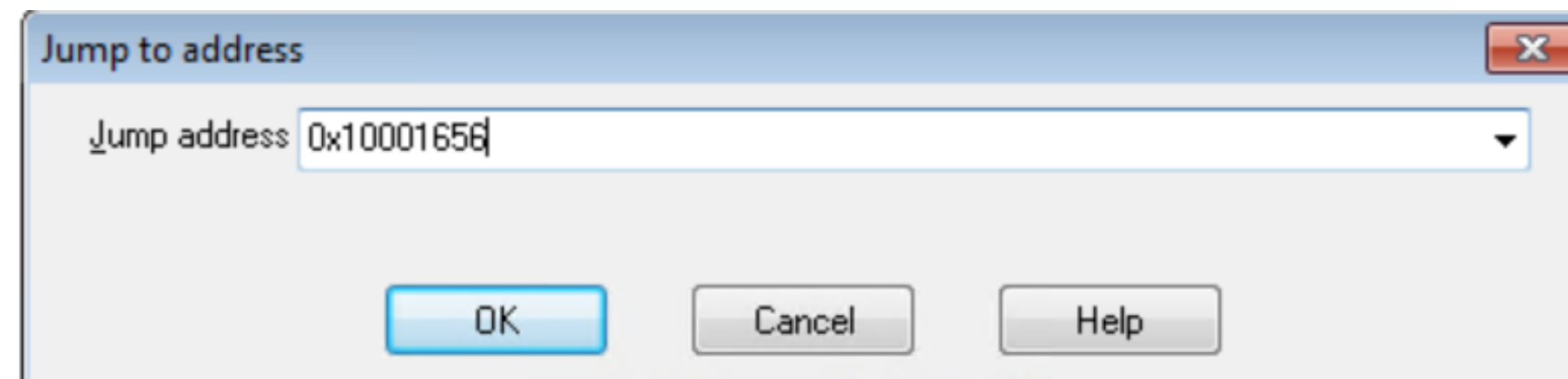
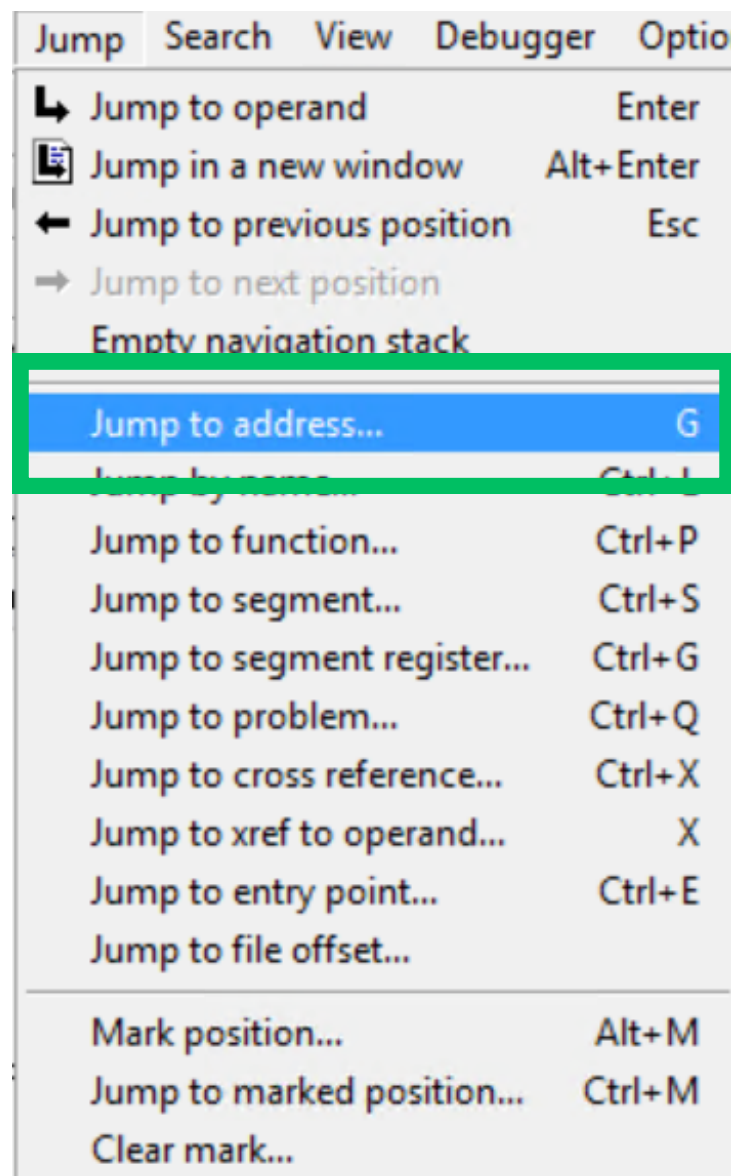
In sostanza la funzione **gethostbyname** è utilizzata per risolvere un nome di dominio in un indirizzo IP. **Accetta un nome di dominio come stringa** e restituisce un puntatore a una struttura hostent che **contiene l'indirizzo IP** associato al nome di dominio. È spesso utilizzata per la **risoluzione DNS nei programmi di rete**.

Nella sezione idata della slide precedente, cliccando col tasto destro sul nome della funzione, si può ottenere un elenco di tutte le funzioni chiamanti con i relativi indirizzi. Probabilmente queste funzioni fanno richieste ad indirizzi esterni, ecco perchè usufruiscono di gethostbyname. Andrebbero analizzate singolarmente.

xrefs to gethostbyname			
Direction	Type	Address	Text
Up	p	sub_10001074+1D3	call ds:gethostbyname
Up	r	sub_10001074+1D3	call ds:gethostbyname
Up	r	sub_10001074+268	call ds:gethostbyname
Up	p	sub_10001074+268	call ds:gethostbyname
Up	r	sub_10001074:loc_1...	call ds:gethostbyname
Up	p	sub_10001074:loc_1...	call ds:gethostbyname
Up	p	sub_10001365+1D3	call ds:gethostbyname
Up	r	sub_10001365+1D3	call ds:gethostbyname
Up	p	sub_10001365+268	call ds:gethostbyname
Up	r	sub_10001365+268	call ds:gethostbyname
Up	p	sub_10001365:loc_1...	call ds:gethostbyname
Up	r	sub_10001365:loc_1...	call ds:gethostbyname
Up	p	sub_10001656+101	call ds:gethostbyname
Up	r	sub_10001656+101	call ds:gethostbyname
Up	p	sub_1000208F+3A1	call ds:gethostbyname
Up	r	sub_1000208F+3A1	call ds:gethostbyname
Up	p	sub_10002CCE+4F7	call ds:gethostbyname

3 - Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?

In questo caso la traccia ci fornisce uno specifico indirizzo di memoria, e quindi, sempre tramite il menu Jump, saltiamo direttamente alla funzione tramite l'indirizzo per capire la situazione.



3 - Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?

4 - Quanti sono, invece, i parametri della funzione sopra?

In totale vi sono ben **23** variabili e un parametro. Si nota dal fatto che, come visto nella teoria, le variabili sono ad un **offset negativo** rispetto al registro EBP. Mentre Arg_0, ha un **offset positivo** rispetto ad EBP, quindi è un parametro.

```
.text:10001656  
.text:10001656 var_675  
.text:10001656 var_674  
.text:10001656 hLibModule  
.text:10001656 timeout  
.text:10001656 name  
.text:10001656 var_654  
.text:10001656 Dst  
.text:10001656 Parameter  
.text:10001656 var_640  
.text:10001656 CommandLine  
.text:10001656 Source  
.text:10001656 Data  
.text:10001656 var_637  
.text:10001656 var_544  
.text:10001656 var_50C  
.text:10001656 var_500  
.text:10001656 Buf2  
.text:10001656 readfds  
.text:10001656 phkResult  
.text:10001656 var_3B0  
.text:10001656 var_1A4  
.text:10001656 var_194  
.text:10001656 WSAData  
.text:10001656 arg_0
```

```
= byte ptr -675h  
= dword ptr -674h  
= dword ptr -670h  
= timeval ptr -66Ch  
= sockaddr ptr -664h  
= word ptr -654h  
= dword ptr -650h  
= byte ptr -644h  
= byte ptr -640h  
= byte ptr -63Fh  
= byte ptr -63Dh  
= byte ptr -638h  
= byte ptr -637h  
= dword ptr -544h  
= dword ptr -50Ch  
= dword ptr -500h  
= byte ptr -4FCh  
= fd_set ptr -4BCh  
= byte ptr -3B8h  
= dword ptr -3B0h  
= dword ptr -1A4h  
= dword ptr -194h  
= WSAData ptr -190h  
= dword ptr 4
```

23 VARIABILI

1 PARAMETRO

5 - Inserire altre considerazioni macro livello sul malware (comportamento)


Un'analisi completa richiederebbe molto tempo e competenze avanzate. Con i pochi elementi che ho esaminato nella giornata, posso notare che il malware utilizza numerose funzioni, tra cui `recv`, `send`, `connect` e `socket`. Queste, insieme alle funzioni che manipolano il filesystem e raccolgono informazioni sul sistema, suggeriscono fortemente la presenza di una backdoor.

Address	Ordinal	Name	Library
000000...		waveInAddBuffer	WINMM
000000...		waveInStart	WINMM
000000...	18	select	WS2_32
000000...	11	inet_addr	WS2_32
000000...	52	gethostbyname	WS2_32
000000...	12	inet_ntoa	WS2_32
000000...	16	recv	WS2_32
000000...	19	send	WS2_32
000000...	4	connect	WS2_32
000000...	15	ntohs	WS2_32
000000...	9	htons	WS2_32
000000...	21	setsockopt	WS2_32
000000...	116	WSACleanup	WS2_32
000000...	115	WSAStartup	WS2_32
000000...	3	closesocket	WS2_32
000000...	23	socket	WS2_32
000000...	111	WSAGetLastError	WS2_32
000000...		GetAdaptersInfo	iphlpapi
000000...		CoTaskMemFree	ole32
000000...		ColInitialize	ole32
000000...		ColInitializeEx	ole32

5 - Inserire altre considerazioni macro livello sul malware (comportamento)

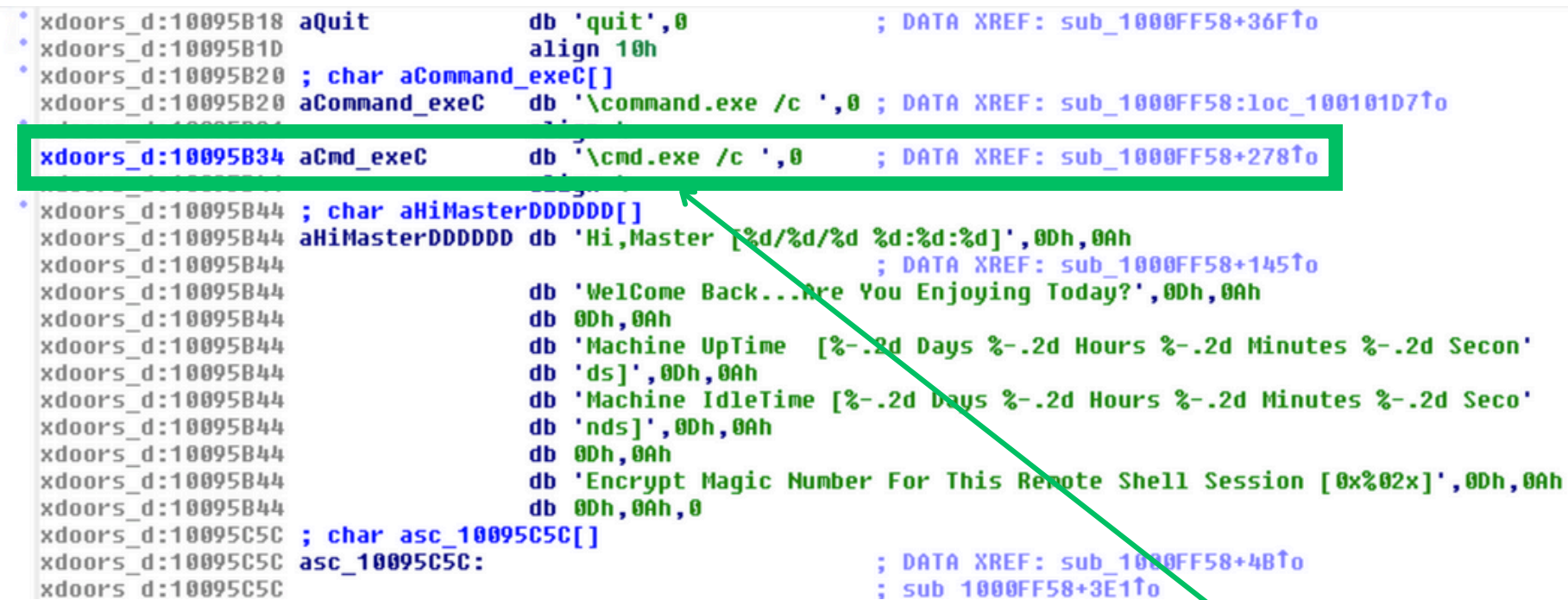
Una sezione interessante esplorata su IDA è quella delle stringhe contenute nell'eseguibile, visibile tramite il menu **View > Open Subview > Strings**. Scorrendo ho trovato stringhe che hanno confermato la mia impressione che si trattasse di una backdoor, tra cui alcune incomprensibili. Ho utilizzato il tool **Strings** in tutte le analisi di malware e nelle attività di OSINT delle precedenti esercitazioni, quindi ho pensato bene di esplorare tale procedura anche in questo caso.

Length	Type	String
0000000C	C	sethostinfo
00000011	C	
00000022	C	socket() GetLastError reports %d\n
00000014	C	Plug_KeyLog_Restart
00000009	C	xkey.dll
00000018	C	WSAStartup() error: %d\n
00000004	C	RETR %s\n\n
00000007	C	PASV\n\n
00000004	C	SIZE %s\n\n
00000009	C	TYPE I\n\n
00000009	C	CWD %s\n\n
00000004	C	PASS %s\n\n
00000004	C	USER %s\n\n
00000012	C	Socket error....\n
00000008	C	IEuser@
00000004	C	anonymous
00000007	C	FTP://
00000007	C	ftp://
00000010	C	Content-Length:
00000008	C	HTTP/1.1 5
00000008	C	HTTP/1.1 3
00000011	C	\\command.exe /c
0000000D	C	\\cmd.exe /c
00000118	C	Hi,Master [%d/%d/%d %d:%d:%d]\n\nWelCome Back...Are You Enjoying Today?\n\nMachine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seconds]\n\nMachine IdleTim...



5 - Inserire altre considerazioni macro livello sul malware (comportamento)

Cliccando due volte su **cmd.exe**, ad esempio, è uscita fuori la parte relativa al codice assembly.



```
* xdoors_d:10095B18 aQuit          db 'quit',0          ; DATA XREF: sub_1000FF58+36F↑to
* xdoors_d:10095B1D             align 10h
* xdoors_d:10095B20 ; char aCommand_exeC[]
xdoors_d:10095B20 aCommand_exeC db '\command.exe /c ',0 ; DATA XREF: sub_1000FF58:loc_100101D7↑to
xdoors_d:10095B34 aCmd_exeC      db '\cmd.exe /c ',0      ; DATA XREF: sub_1000FF58+278↑to
* xdoors_d:10095B44 ; char aHiMasterDDDDDD[]
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
xdoors_d:10095B44             ; DATA XREF: sub_1000FF58+145↑to
xdoors_d:10095B44             db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
xdoors_d:10095B44             db 0Dh,0Ah
xdoors_d:10095B44             db 'Machine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Secon'
xdoors_d:10095B44             db 'ds]',0Dh,0Ah
xdoors_d:10095B44             db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44             db 'nds]',0Dh,0Ah
xdoors_d:10095B44             db 0Dh,0Ah
xdoors_d:10095B44             db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095B44             db 0Dh,0Ah,0
xdoors_d:10095C5C ; char asc_10095C5C[]
xdoors_d:10095C5C asc_10095C5C: ; DATA XREF: sub_1000FF58+4B↑to
xdoors_d:10095C5C             ; sub_1000FF58+3E1↑to
```

La riga sottolineata in verde sembra faccia riferimento a una stringa **\cmd.exe /c** (scovata nella slide precedente) memorizzata all'indirizzo 0x10095B34. Questo comando solitamente viene utilizzato per eseguire comandi nel prompt dei comandi di Windows, quindi potrebbe essere un suggerimento che la funzione **sub_1000FF58** in qualche modo sia coinvolta nell'esecuzione di comandi sulla macchina compromessa, tipico di una backdoor.



GRAZIE

FLAVIO SCOGNAMIGLIO
