



CONSEGNA

S11/L2

DI GIUSEPPE LUPOI



Traccia

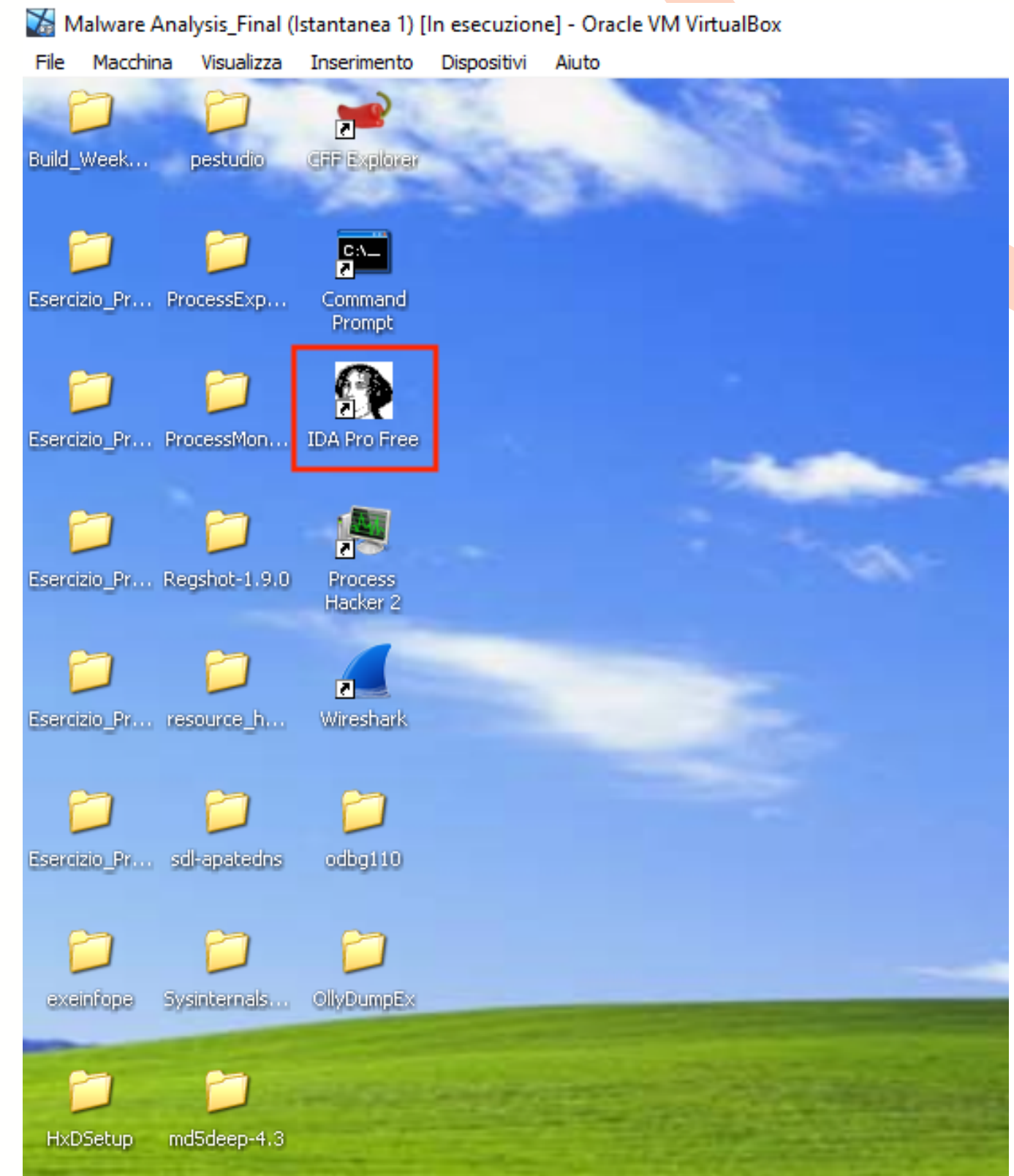


Lo scopo dell'esercizio di oggi è di acquisire esperienza con IDA, un tool fondamentale per l'analisi statica. A tal proposito, con riferimento al malware chiamato «Malware_U3_W3_L2» presente all'interno della cartella «Esercizio_Pratico_U3_W3_L2» sul desktop della macchina virtuale dedicata all'analisi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

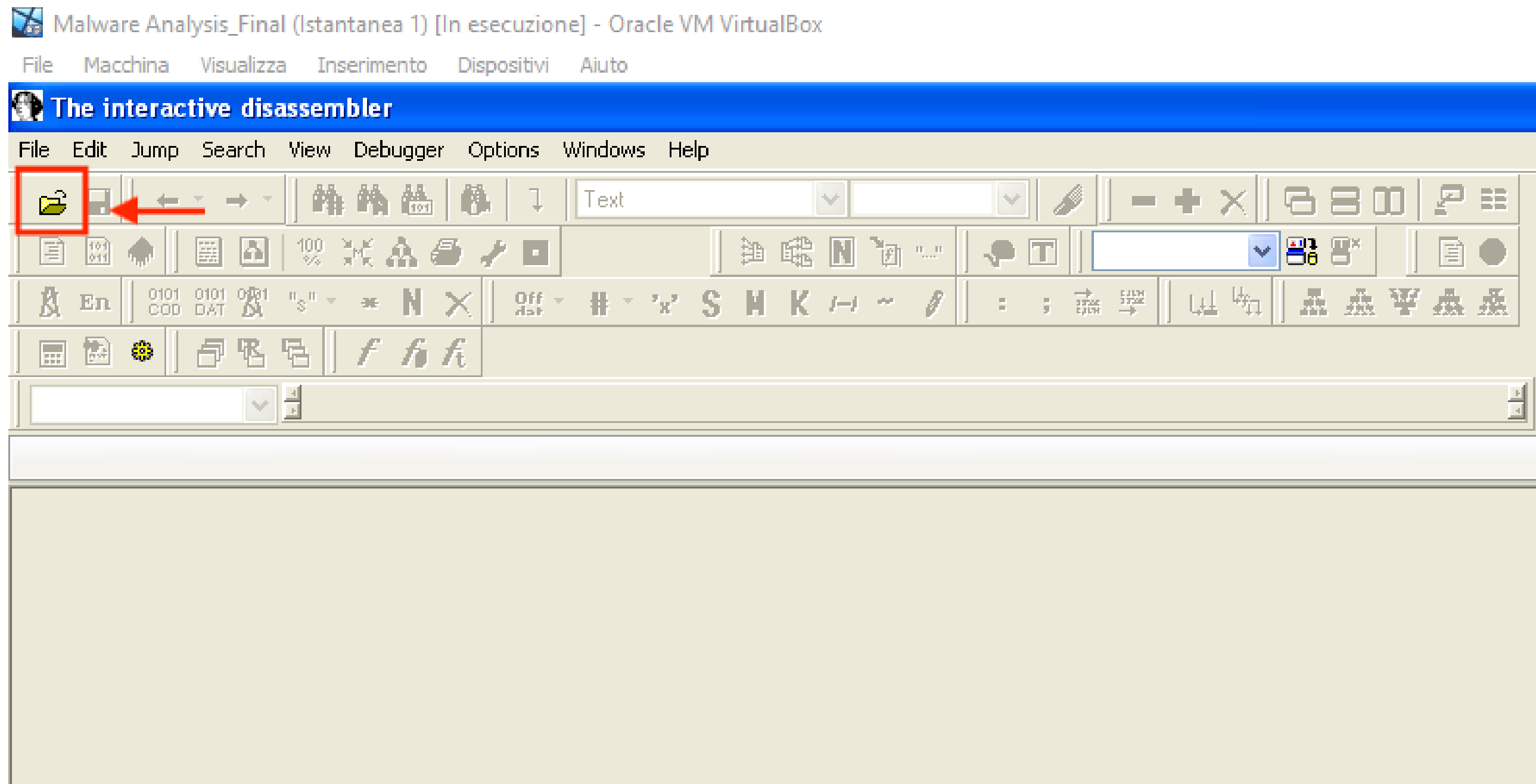
1. Individuare l'indirizzo della funzione DLLMain
2. Dalla scheda «imports» individuare la funzione «gethostbyname». Qual è l'indirizzo dell'import?
3. Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?
4. Quanti sono, invece, i parametri della funzione sopra?



Per lo svolgimento del primo punto della traccia di oggi, una volta recati nella macchina a noi fornita per l'analisi dei malware, dal desktop clicchiamo su "IDA Pro Free" e come abbiamo visto nella lezione di oggi ci verrà proposta l'interfaccia del tool.

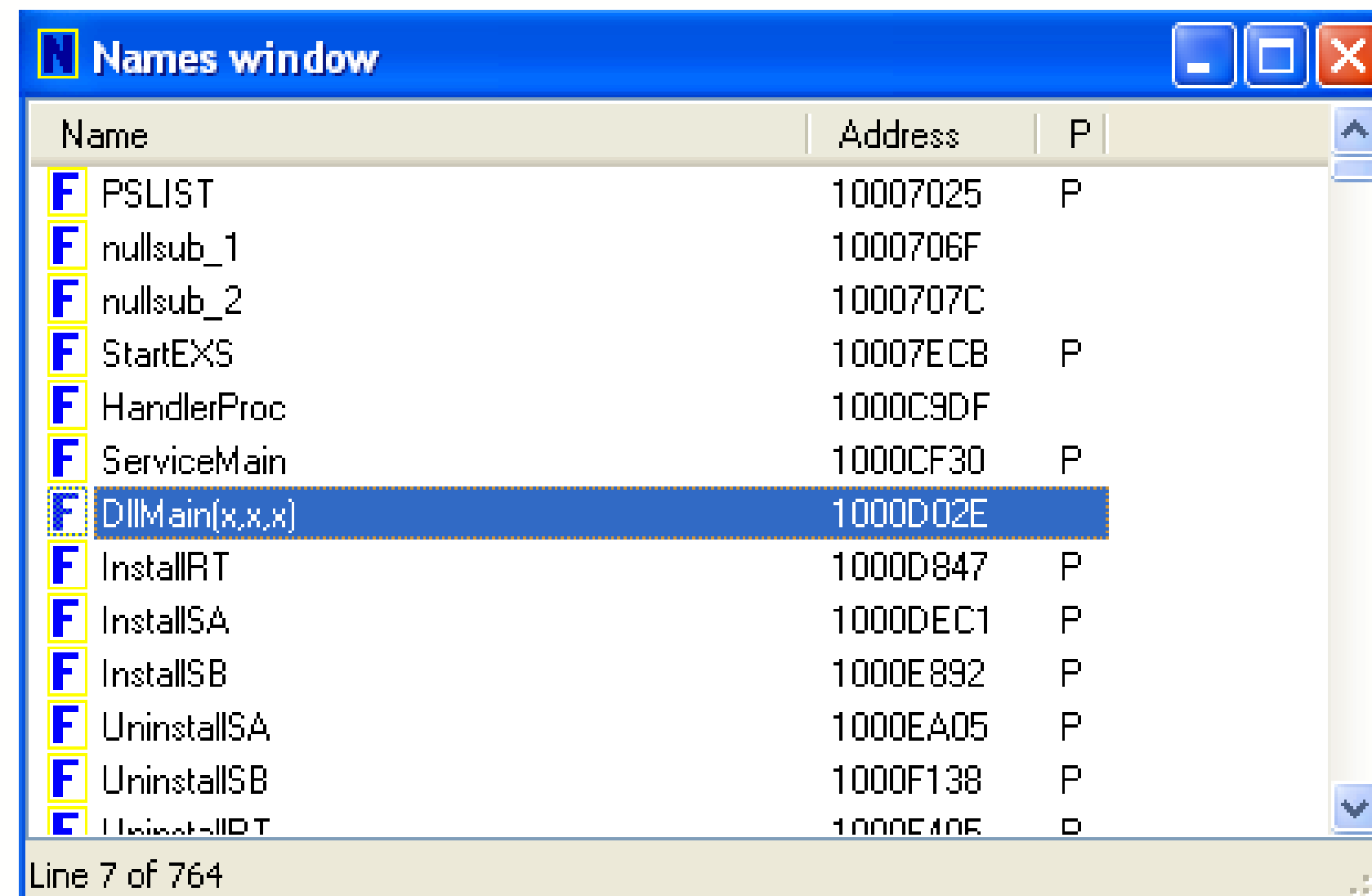


Clicchiamo nella cartella in alto a sinistra come mostrato dal quadrato in rosso e selezioniamo il malware di nostro interesse, oggi sarà Malware_U3_W3_L2.



Iniziamo quindi, come richiesto dalla traccia, l'indirizzo della funzione **DDLMain**.

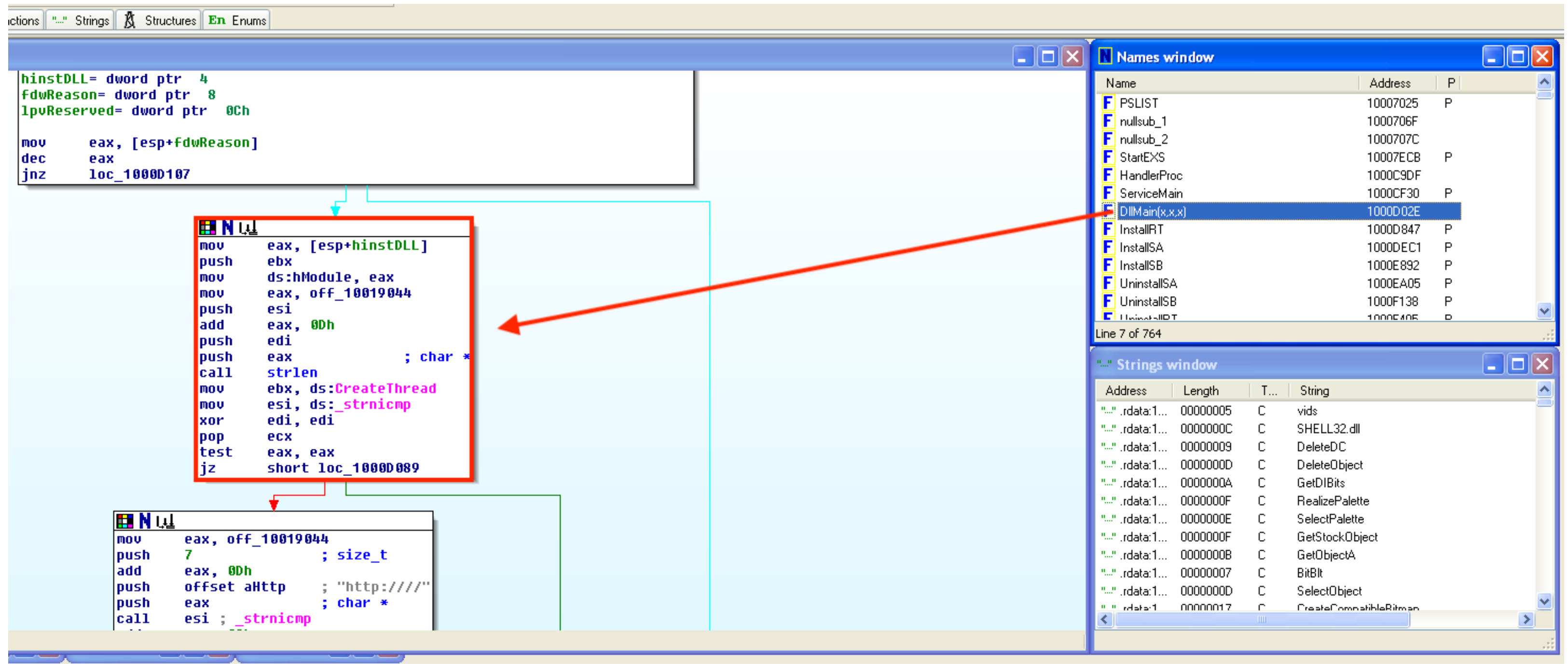
Per facilitarci il compito, sulla destra **IDA Pro Free** ci permette di individuare le funzioni da una lista nella tabella di nome **Names Windows**.



Name	Address	P
PSLIST	10007025	P
nullsub_1	1000706F	
nullsub_2	1000707C	
StartEXS	10007ECB	P
HandlerProc	1000C9DF	
ServiceMain	1000CF30	P
DllMain(x,x,x)	1000D02E	
InstallRT	1000D847	P
InstallSA	1000DEC1	P
InstallSB	1000E892	P
UninstallSA	1000EA05	P
UninstallSB	1000F138	P
UninstallRT	1000E40E	P

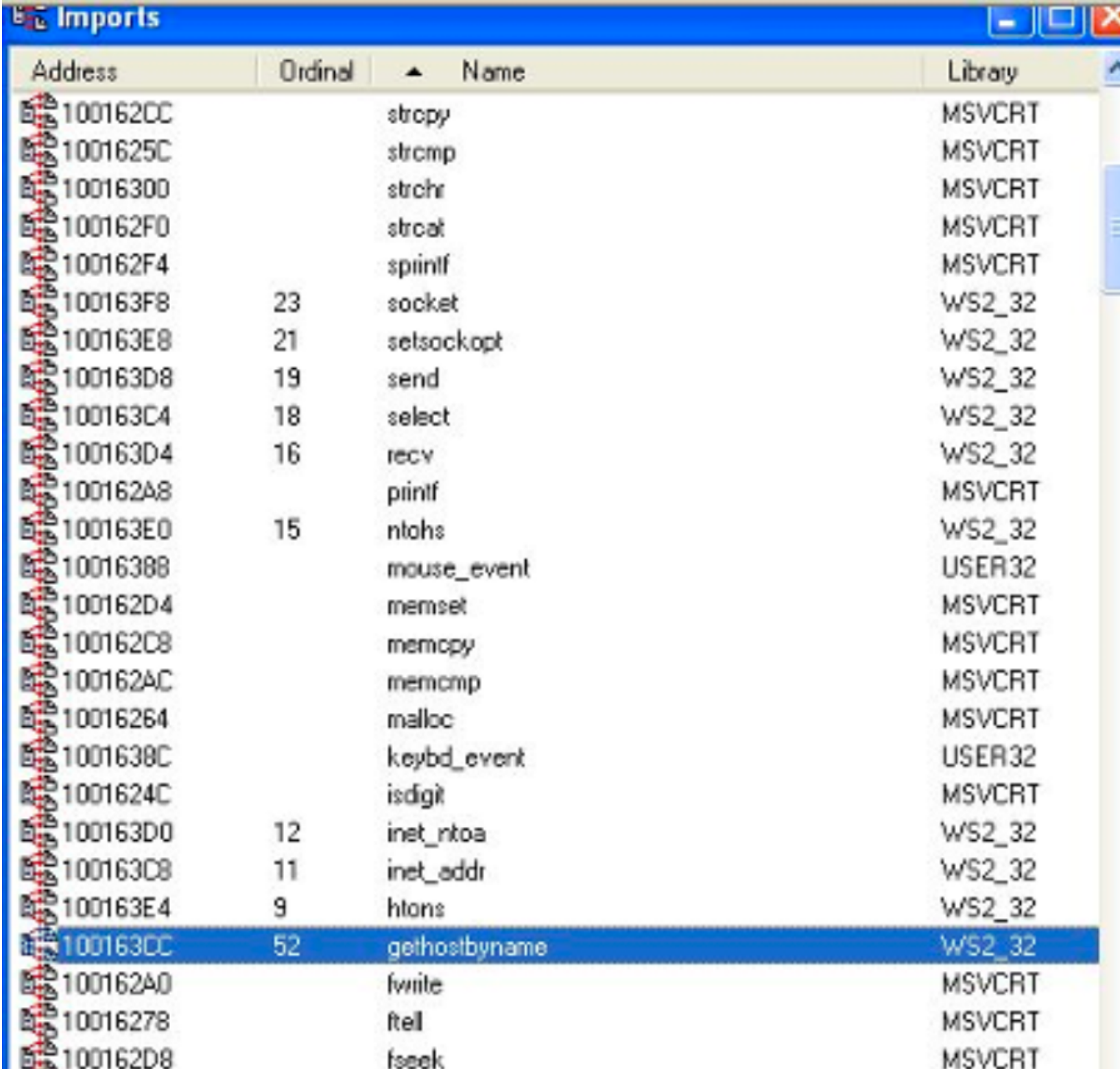
Line 7 of 764

Cliccando quindi sul nome della funzione che ci interessa verrà identificata sulla finestra principale in quello che possiamo chiamare uno schema da codice tradotto da **IDA Pro Free**.

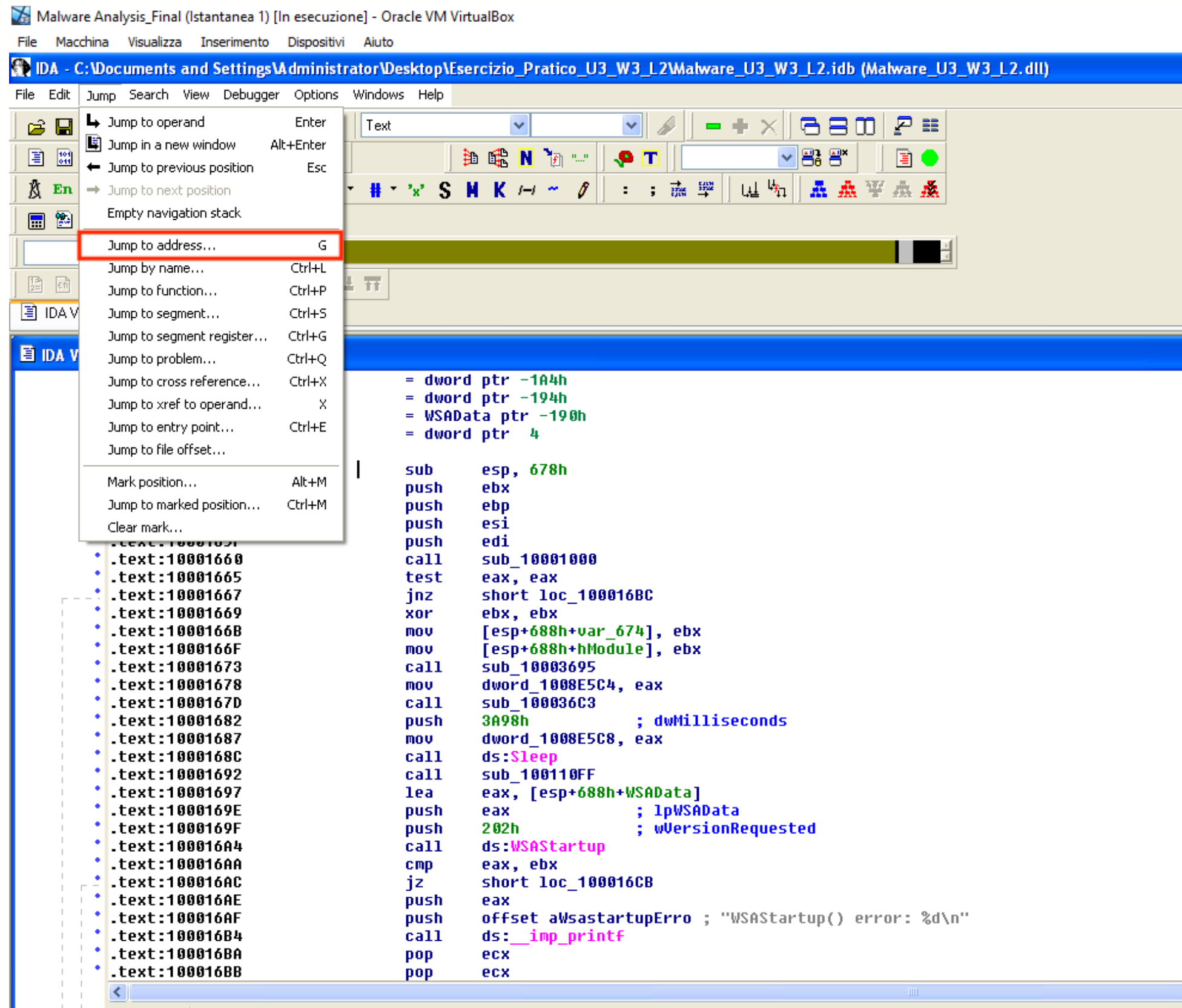


Per il secondo punto della traccia individueremo la funzione “**gethostbyname**” e l’indirizzo dell’import.

Ci basterà navigare nella barra degli strumenti “**Imports**” posizionata in alto, scorrendo troveremo la funzione che cerchiamo ed alla sua sinistra il relativo indirizzo che notiamo essere “**100163CC**”.



Address	Ordinal	Name	Library
100162CC		strcpy	MSVCRT
1001625C		strcmp	MSVCRT
10016300		strchr	MSVCRT
100162F0		strcat	MSVCRT
100162F4		sprintf	MSVCRT
100163F8	23	socket	WS2_32
100163E8	21	setsockopt	WS2_32
100163D8	19	send	WS2_32
100163C4	18	select	WS2_32
100163D4	16	recv	WS2_32
100162A8		printf	MSVCRT
100163E0	15	ntohs	WS2_32
10016388		mouse_event	USER32
100162D4		memset	MSVCRT
100162C8		memcpy	MSVCRT
100162AC		memcmp	MSVCRT
10016264		malloc	MSVCRT
1001638C		keybd_event	USER32
1001624C		isdigit	MSVCRT
100163D0	12	inet_ntoa	WS2_32
100163C8	11	inet_addr	WS2_32
100163E4	9	htons	WS2_32
100163CC	52	gethostbyname	WS2_32
100162A0		fwrite	MSVCRT
10016278		fread	MSVCRT
100162D8		fseek	MSVCRT



Al quarto punto della traccia per trovare direttamente l'indirizzo di locazione richiesto possiamo usare la voce **Jump** dalla barra degli strumenti in alto ed infine cliccando su “**Jump to address**” ed inserendo l'indirizzo verremo reindirizzati alla locazione desiderata.

Una volta davanti al pezzo di codice che vogliamo analizzare dando un'occhiata possiamo notare 20 funzioni in questa locazione di memoria.

```
View-A
.text:10001656 arg_0 = dword ptr 4
.text:10001656
.text:10001656 sub esp, 678h
.text:1000165C push ebx
.text:1000165D push ebp
.text:1000165E push esi
.text:1000165F push edi
.text:10001660 call sub_10001000
.text:10001665 test eax, eax
-- .text:10001667 jnz short loc_100016BC
.text:10001669 xor ebx, ebx
.text:1000166B mov [esp+688h+var_674], ebx
.text:1000166F mov [esp+688h+hModule], ebx
.text:10001673 call sub_10003695
.text:10001678 mov dword_1000E5C4, eax
.text:1000167D call sub_100036C3
.text:10001682 push 3A98h ; dwMilliseconds
.text:10001687 mov dword_1000E5C8, eax
.text:1000168C call ds:Sleep
.text:10001692 call sub_100110FF
.text:10001697 lea eax, [esp+688h+WSAData]
.text:1000169E push eax ; lpWSAData
.text:1000169F push 202h ; wVersionRequested
.text:100016A4 call ds:WSAStartup
.text:100016AA cmp eax, ebx
.text:100016AC jz short loc_100016CB
.text:100016AE push eax
.text:100016AF push offset aWsastartupErro ; "WSAStartup() error: %d\n"
.text:100016B4 call ds:__imp_printf
.text:100016BA pop ecx
.text:100016BB pop ecx
.text:100016BC
.text:100016BC loc_100016BC: ; CODE XREF: sub_10001656+11↑j
-- .text:100016BC pop edi

44 32 8192 allocating memory for name pointers...
```



Quindi continuando l'analisi del malware è stata rilevata che questa funzione accetta un unico parametro in ingresso, ovvero **arg_0**.

Il parametro arg_0 rappresenta un'interfaccia critica per la funzione, fungendo da canale di ingresso per i dati che saranno manipolati o analizzati dalla funzione stessa. Nella programmazione, i parametri di una funzione sono fondamentali per la modularità e la riutilizzabilità del codice, permettendo alle funzioni di operare su dati diversi senza la necessità di modificare il corpo della funzione.