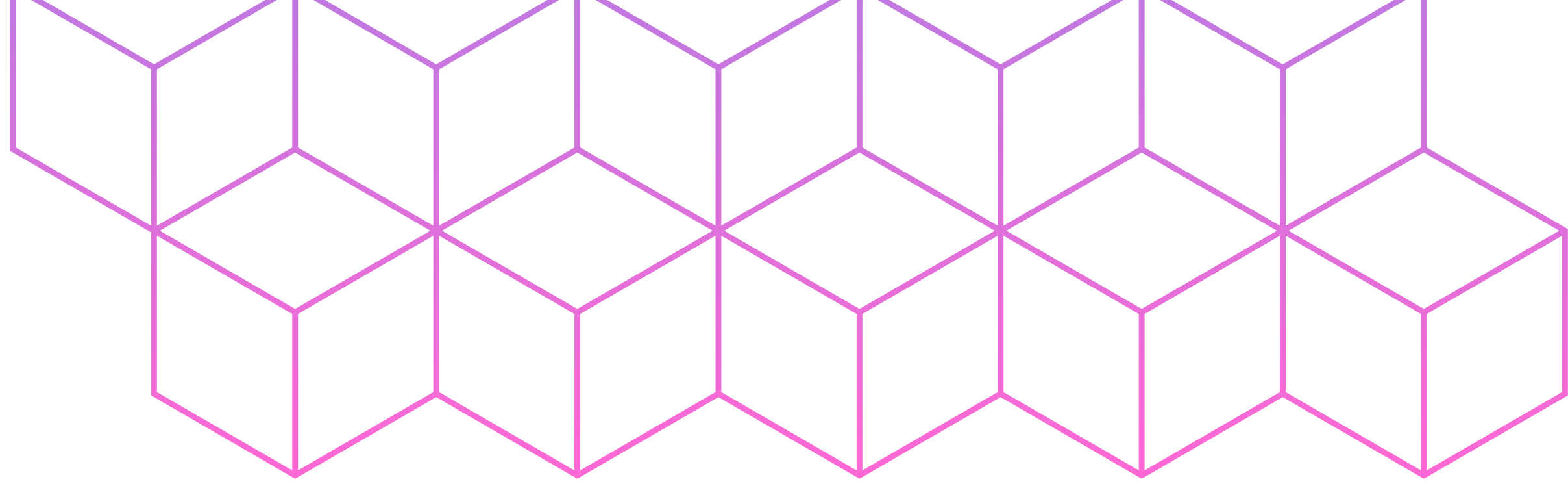
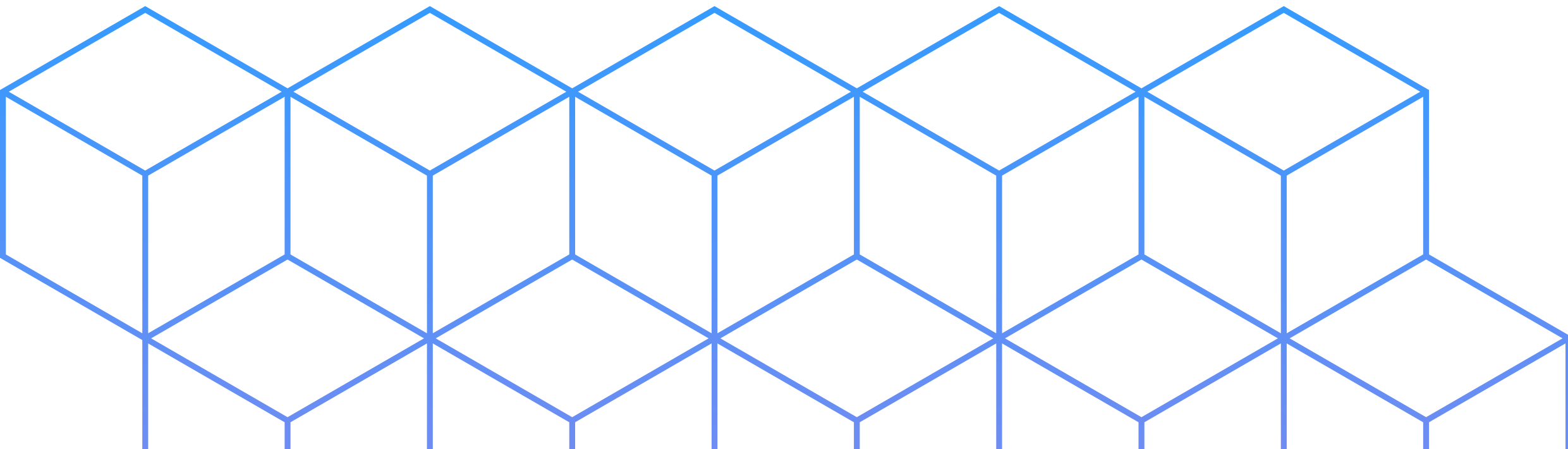


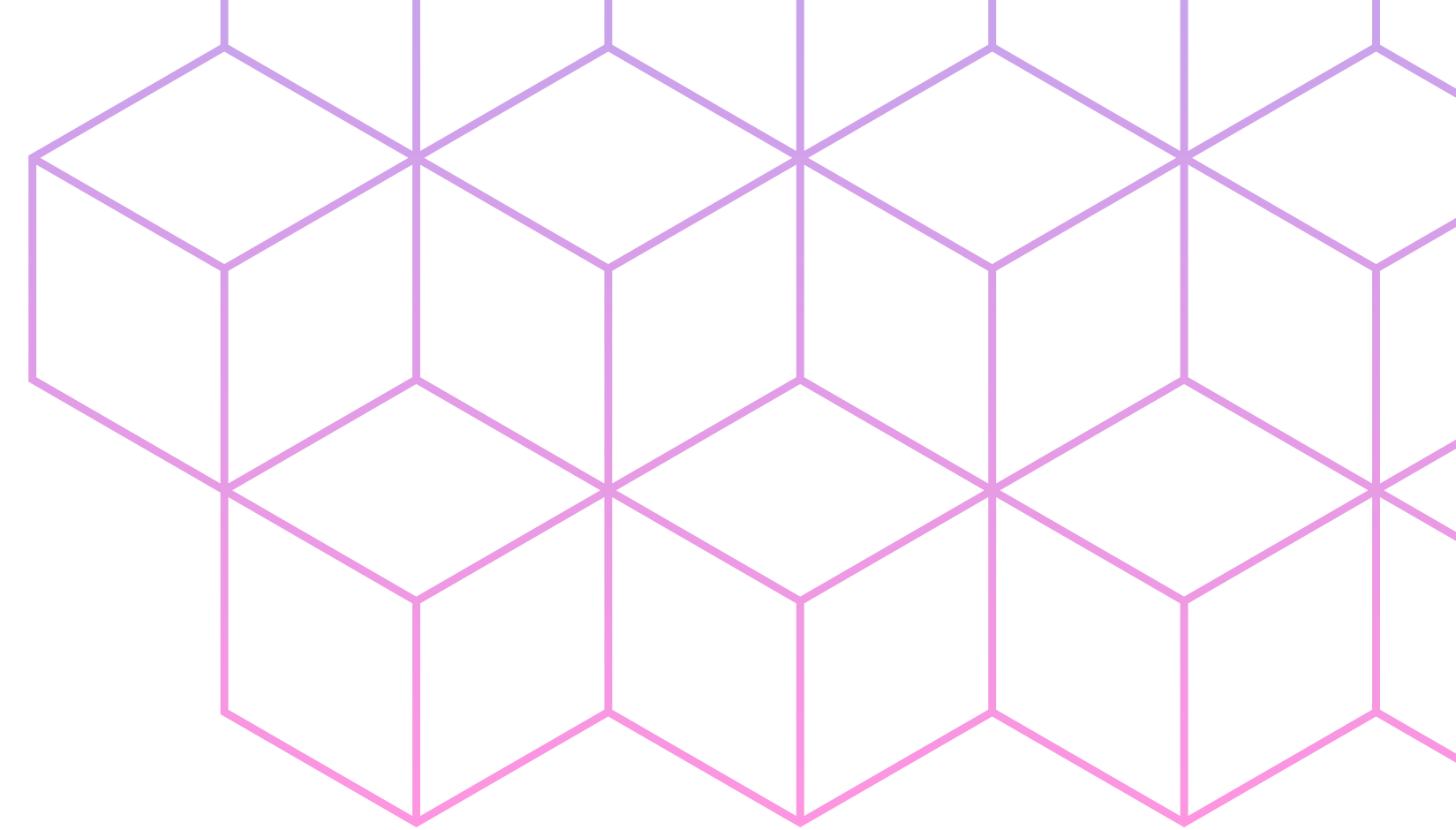
Epicode



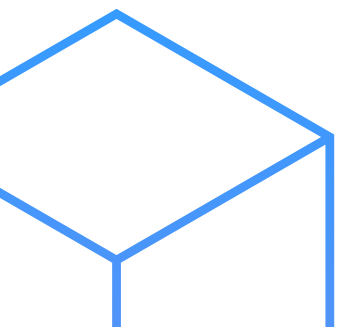
Malware Analysis con IDA



- Traccia
- Individuare l'indirizzo della funzione DLLMain
- La funzione “gethostbyname”
- Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?
- Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?
- Quanti sono, invece, i parametri della funzione sopra?
- Altre considerazioni macro livello sul malware (comportamento)



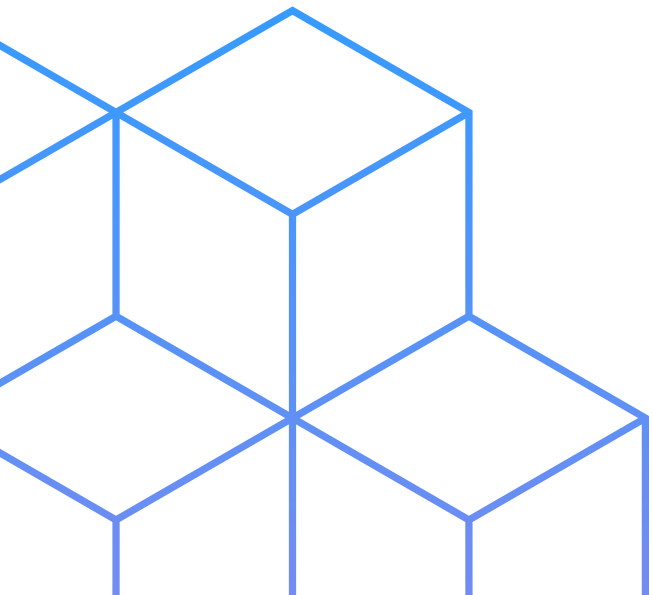
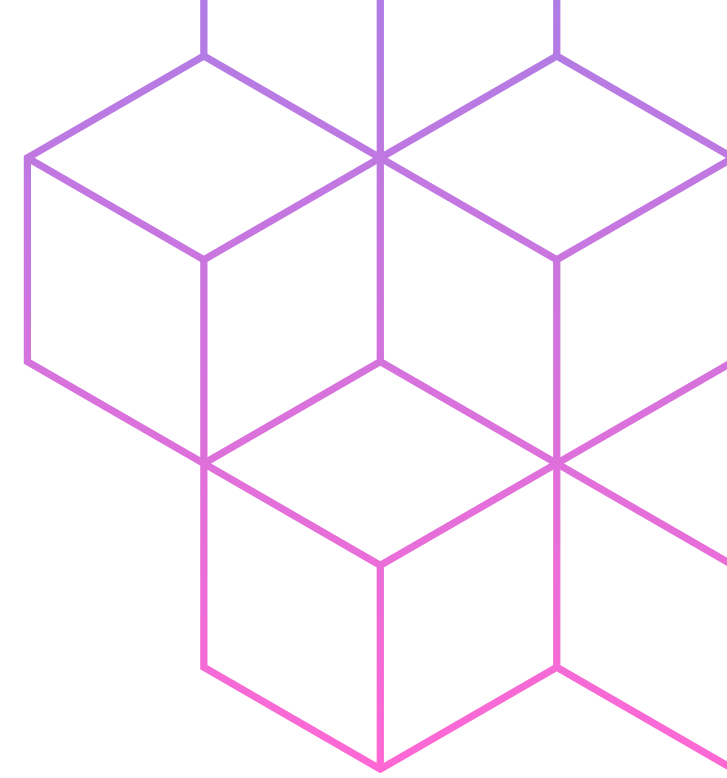
Indice



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.

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

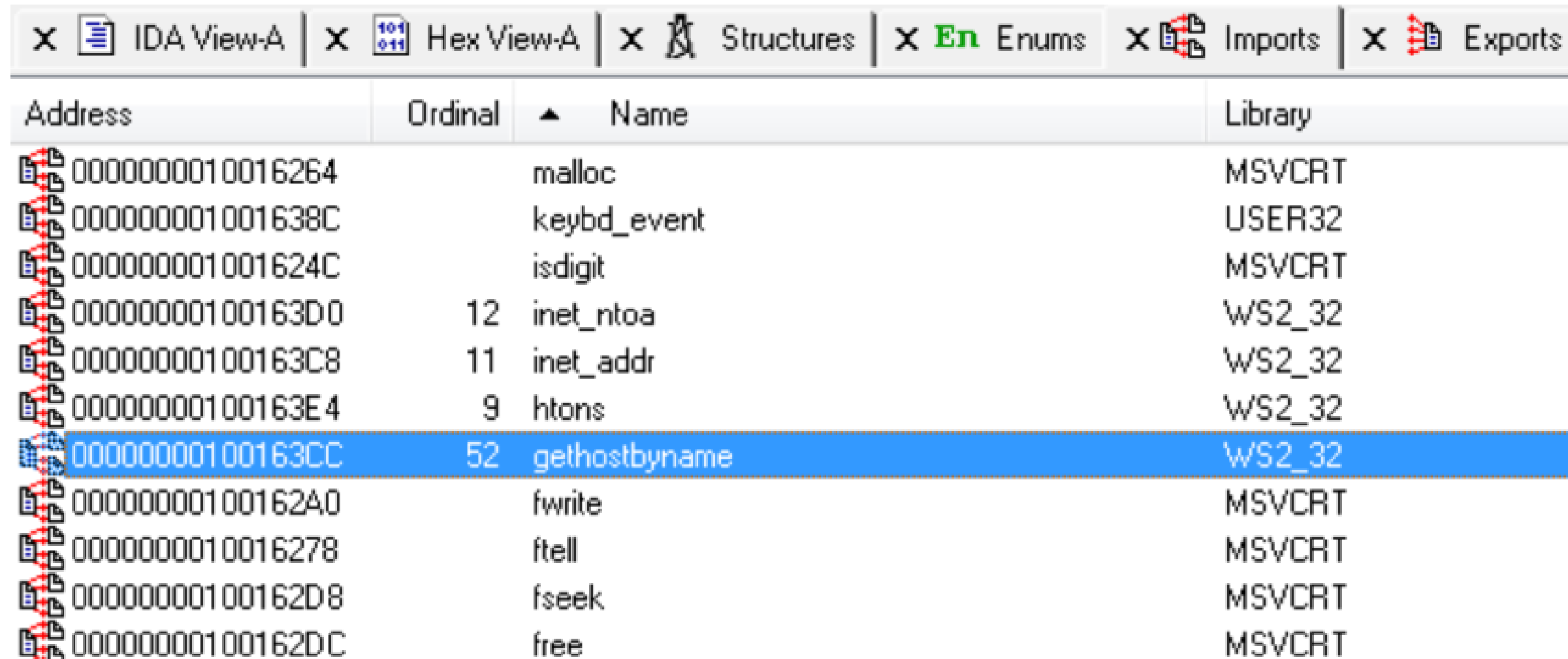


Individuare l'indirizzo della funzione DLLMain

```
.text:1000D02E
.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
.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]
.text:1000D03D      push    ebx
.text:1000D03E      mov     ds:hModule, eax
.text:1000D043      mov     eax, off_10019044
.text:1000D048      push    esi
```

Nella prima parte di questa esercitazione attraverso il software disassembler *IDA Pro* analizziamo il malware così da tradurre il codice macchina in Assembly. Dopo aver caricato il malware su IDA esamineremo il codice tradotto. La porzione di codice e l'indirizzo della funzione DLLMain sono facilmente individuabili: si trova all'indirizzo di memoria 1000D02E.

La funzione “gethostbyname”



Address	Ordinal	Name	Library
0000000010016264		malloc	MSVCRT
000000001001638C		keybd_event	USER32
000000001001624C		isdigit	MSVCRT
00000000100163D0	12	inet_ntoa	WS2_32
00000000100163C8	11	inet_addr	WS2_32
00000000100163E4	9	htons	WS2_32
00000000100163CC	52	gethostbyname	WS2_32
00000000100162A0		fwrite	MSVCRT
0000000010016278		ftell	MSVCRT
00000000100162D8		fseek	MSVCRT
00000000100162DC		free	MSVCRT

Proseguiamo nell'analisi andando ad esaminare la scheda degli *imports* per identificare le funzioni importate dal malware. In questo modo, individuiamo anche l'indirizzo di memoria della funzione richiesta, *gethostbyname*, che è 100163CC.

La funzione *gethostbyname* recupera le informazioni host corrispondenti a un nome host da un database host.

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

```
.text:10001656 ; ===== S U B R O U T I N E =====
.text:10001656
.text:10001656
.text:10001656 ; DWORD __stdcall sub_10001656(LPVOID)
.text:10001656 sub_10001656 proc near ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
.text:10001656 var_675 = byte ptr -675h
.text:10001656 var_674 = dword ptr -674h
.text:10001656 hLibModule = dword ptr -670h
.text:10001656 timeout = timeval ptr -66Ch
.text:10001656 name = sockaddr ptr -664h
.text:10001656 var_654 = word ptr -654h
.text:10001656 Dst = dword ptr -650h
.text:10001656 Parameter = byte ptr -644h
.text:10001656 var_640 = byte ptr -640h
.text:10001656 CommandLine = byte ptr -63Fh
.text:10001656 Source = byte ptr -63Dh
.text:10001656 Data = byte ptr -638h
.text:10001656 var_637 = byte ptr -637h
.text:10001656 var_544 = dword ptr -544h
.text:10001656 var_50C = dword ptr -50Ch
.text:10001656 var_500 = dword ptr -500h
.text:10001656 Buf2 = byte ptr -4FCh
```

```
.text:10001656 var_194 = dword ptr -194h
.text:10001656 WSADATA = WSADATA ptr -190h
.text:10001656 arg_0 = dword ptr 4
```

Focalizziamoci sulla locazione di memoria 0x10001656. Nella casella search inseriamo 10001656 per filtrare i risultati. Qui abbiamo individuato 20 variabili con offset negativo rispetto ad EBP. Tuttavia, abbiamo notato che c'è un solo parametro associato a un offset positivo

Quanti sono, invece, i parametri della funzione sopra?

```
.text:10001656 ; ===== S U B R O U T I N E =====
.text:10001656
.text:10001656
.text:10001656 ; DWORD __stdcall sub_10001656(LPVOID)
.text:10001656 sub_10001656 proc near ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
.text:10001656 var_675 = byte ptr -675h
.text:10001656 var_674 = dword ptr -674h
.text:10001656 hLibModule = dword ptr -670h
.text:10001656 timeout = timeval ptr -66Ch
.text:10001656 name = sockaddr ptr -664h
.text:10001656 var_654 = word ptr -654h
.text:10001656 Dst = dword ptr -650h
.text:10001656 Parameter = byte ptr -644h
.text:10001656 var_640 = byte ptr -640h
.text:10001656 CommandLine = byte ptr -63Fh
.text:10001656 Source = byte ptr -63Dh
.text:10001656 Data = byte ptr -638h
.text:10001656 var_637 = byte ptr -637h
.text:10001656 var_544 = dword ptr -544h
.text:10001656 var_50C = dword ptr -50Ch
.text:10001656 var_500 = dword ptr -500h
.text:10001656 Buf2 = byte ptr -4FCh
```

```
.text:10001656 var_194 = dword ptr -194h
.text:10001656 WSADATA = WSADATA ptr -190h
.text:10001656 arg_0 = dword ptr 4
```

Dalla stessa immagine, possiamo osservare che solo un parametro viene passato alla funzione.

Questo argomento ha un offset positivo rispetto al registro EBP.

IDA Pro ha etichettato questo parametro come "arg_0".

Altre considerazioni macro livello sul malware (comportamento)

```
.text:1000437D  
.text:1000437E  
.text:10004384  
.text:1000438A  
.text:10004390  
.text:10004395  
.text:10004396  
.text:10004398  
.text:1000439B
```

```
push    edi                ; nBufferLength  
call    ds:GetCurrentDirectoryA  
mov     esi, ds:sprintf  
lea     eax, [ebp+buf]  
push    offset aBackdoorServer ; "\\r\n\r\n*****\r\n[Ba"  
push    eax                ; Dest  
call    esi ; sprintf  
mov     ebx, [ebp+s]  
lea     eax, [ebp+buf]
```

Potremmo dedurre che il comportamento del malware sia simile a quello di una backdoor, infatti possiamo notare funzioni e variabili inerenti a “backdoorserver”