

S11L2

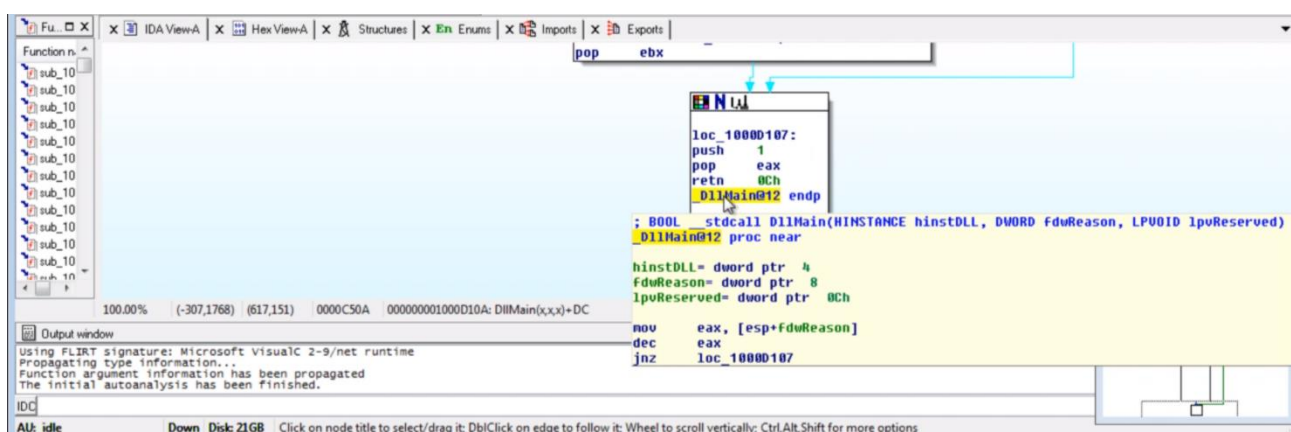
Consegna:

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 macrolivello sul malware (comportamento)

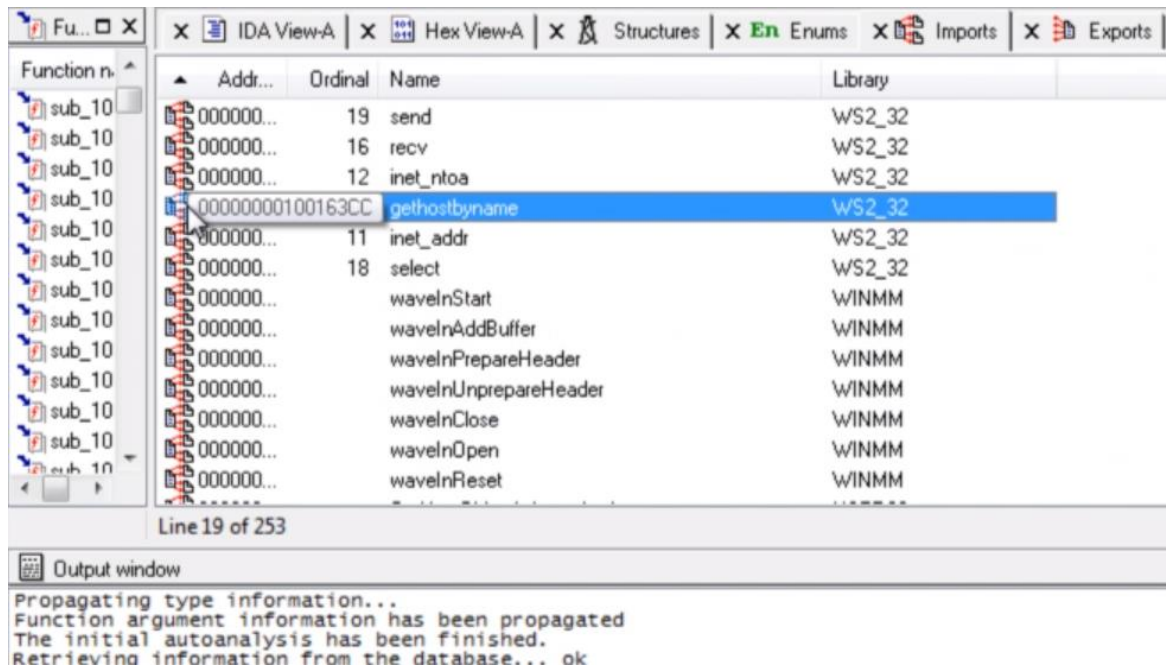
Risoluzione:

1. Per individuare l'indirizzo di memoria di DLLMain basta trovare la chiamata di funzione nell'interfaccia grafica fornita da IDA Pro, una volta fatto ciò basterà spostare il cursore del mouse sulla funzione ed è possibile osservare che per attingere alla funzione bisogna compiere un salto all'indirizzo di memoria 1000D107, e se si prova a controllare queste coordinate nell'interfaccia testuale di IDA Pro, si può notare che poco dopo, più precisamente all'indirizzo 1000D10A, è presente la funzione DLLMain.



```
.text:1000D107 loc_1000D107:  
.text:1000D107  
.text:1000D109  
.text:1000D10A  
.text:1000D10A _DllMain@12  
.text:1000D10A  
push 1  
pop eax  
retn 0Ch  
endp
```

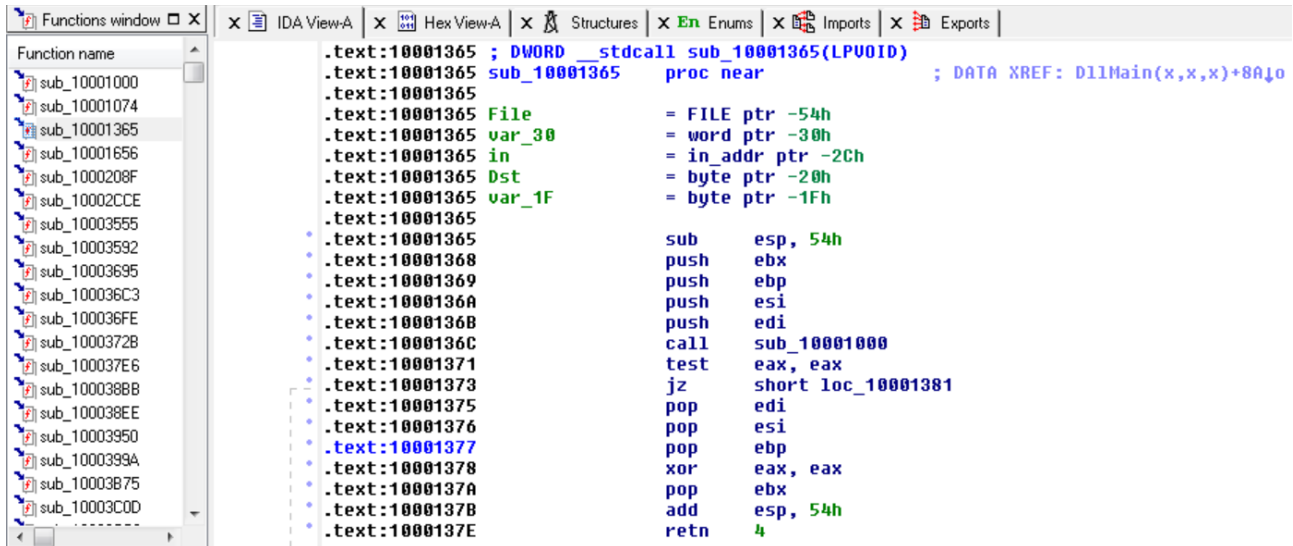
2. Per controllare l'indirizzo della funzione "gethostbyname" dalle scheda import, bisogna cercare la funzione, una volta fatto ciò basterà sportare il cursore del mouse sulla funzione interessata per controllarne l'indirizzo.



3. Per controllare quante variabili sono presenti nella funzione che sta nell'indirizzo di memoria 0x10001656 basta andare al suddetto indirizzo dalla funzione testuale per poi contare quanti parametri var_x (dove x è un qualunque numero intero). In questo caso se ne possono contare 11

```
.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 readfds      = fd_set ptr -4BCh
.text:10001656 phkResult     = byte ptr -3B8h
.text:10001656 var_3B0      = dword ptr -3B0h
.text:10001656 var_1A4      = dword ptr -1A4h
.text:10001656 var_194      = dword ptr -194h
.text:10001656 WSADATA      = WSADATA ptr -190h
```

4. Per capire quanti parametri fossero presenti nella funzione sopra nella sezione Functions windows alla funzione analizzata in precedenza, basterà cliccarci sopra per avere una panoramica del codice assembly della funzione stessa, per poi contarne i parametri. In questo caso sono 2, ovvero File e Dst.



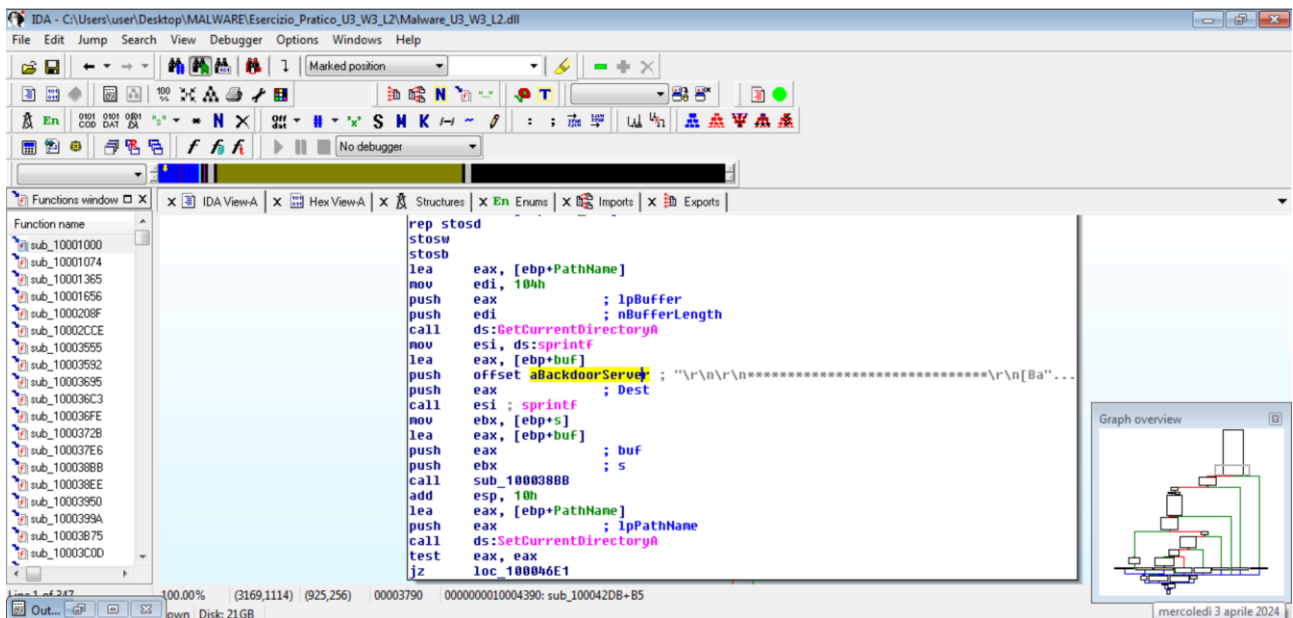
The screenshot shows the IDA Pro interface. On the left, the 'Functions window' lists several subroutines. The main window displays the assembly code for the selected function, `sub_10001365`. The code is as follows:

```

.text:10001365 ; DWORD __stdcall sub_10001365(LPVOID)
.text:10001365 sub_10001365 proc near ; DATA XREF: DllMain(x,x,x)+8A40
.text:10001365
.text:10001365 File = FILE ptr -54h
.text:10001365 var_30 = word ptr -30h
.text:10001365 in = in_addr ptr -2Ch
.text:10001365 Dst = byte ptr -20h
.text:10001365 var_1F = byte ptr -1Fh
.text:10001365
.text:10001365 sub esp, 54h
.text:10001368 push ebx
.text:10001369 push ebp
.text:1000136A push esi
.text:1000136B push edi
.text:1000136C call sub_10001000
.text:10001371 test eax, eax
.text:10001373 jz short loc_10001381
.text:10001375 pop edi
.text:10001376 pop esi
.text:10001377 pop ebp
.text:10001378 xor eax, eax
.text:1000137A pop ebx
.text:1000137B add esp, 54h
.text:1000137E retn 4

```

5. Sulla base del codice analizzato, questo malware probabilmente è una backdoor poiché, come riportato di sotto, ci sono delle funzioni che fanno riferimento ad un Backdoor server.



The screenshot shows the IDA Pro interface with the assembly code for `sub_10001365` displayed. The code is as follows:

```

rep stosd
stosw
stosb
lea eax, [ebp+PathName]
mov edi, 104h
push eax ; lpBuffer
push edi ; nBufferLength
call ds:GetCurrentDirectoryA
mov esi, ds:sprintf
lea eax, [ebp+buf]
offset aBackdoorServer ; "\r\n\r\n*****\r\n[Ba"...
push eax ; Dest
call esi ; sprintf
mov ebx, [ebp+s]
lea eax, [ebp+buf]
push eax ; buf
push ebx ; s
call sub_1000388B
add esp, 10h
lea eax, [ebp+PathName]
push eax ; lpPathName
call ds:SetCurrentDirectoryA
test eax, eax
jz loc_100046E1

```

The code includes a comment `offset aBackdoorServer ; "\r\n\r\n*****\r\n[Ba"...`, which indicates a reference to a backdoor server. A 'Graph overview' window is also visible on the right side of the interface.