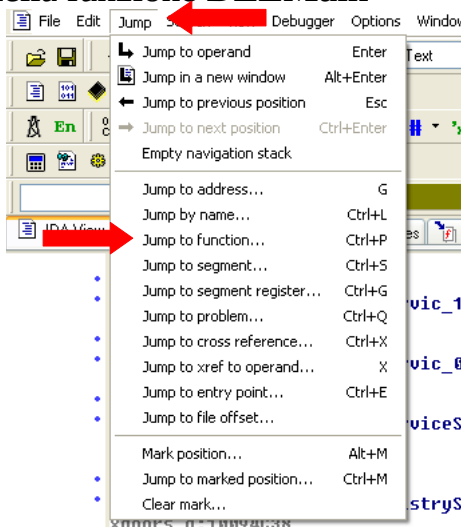


# Analisi Malware avanzata dinamica

## 1) Individuare l'indirizzo della funzione **DLLMain**

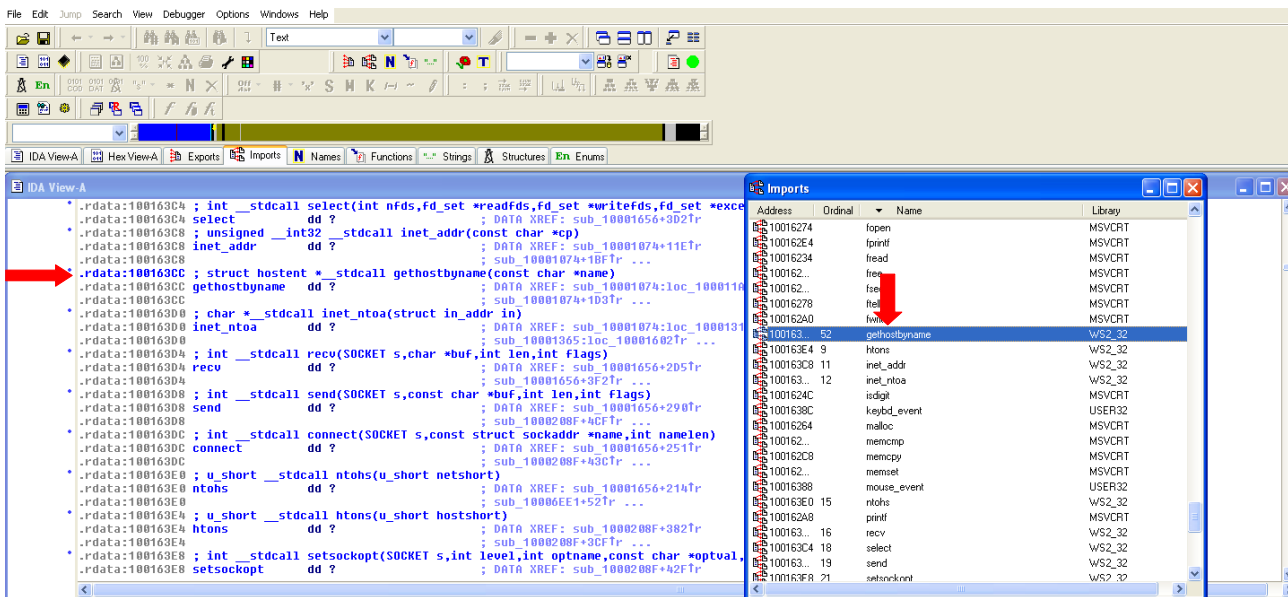
Per cercare un indirizzo di una specifica funzione **IDA** mette a disposizione la funzione **jump** che permette di cercare nel codice secondo alcune possibilità. In questo caso noi cercheremo per **function**.



Ed ecco che cercando **DLLMain** verremo indirizzati all'indirizzo **100D02E** che è l'indirizzo della funzione.

```
.text:1000002E
.text:1000D02E ; 800L __stdcall DllMain(HINSTANCE hinstDLL,DWORD fdwReason,LPVOID lpvReserved)
.text:1000D02E _DllMain@12      proc near
.text:1000D02E ; 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]
* .text:1000D03D      push    ebx
* .text:1000D03E      mov     ds:hModule, eax
* .text:1000D043      mov     eax, off_10019044
* .text:1000D048      push    esi
* .text:1000D049      add     eax, 0Dh
* .text:1000D04C      push    edi
* .text:1000D04D      push    eax
* .text:1000D04E      call   j_strlen
* .text:1000D053      mov     ebx, ds:CreateThread
* .text:1000D059      mov     esi, ds:_strnicmp
* .text:1000D05F      xor     edi, edi
* .text:1000D061      pop     ecx
* .text:1000D062      test    eax, eax
* .text:1000D064      jz      short loc_1000D089
* .text:1000D066      mov     eax, off_10019044
```

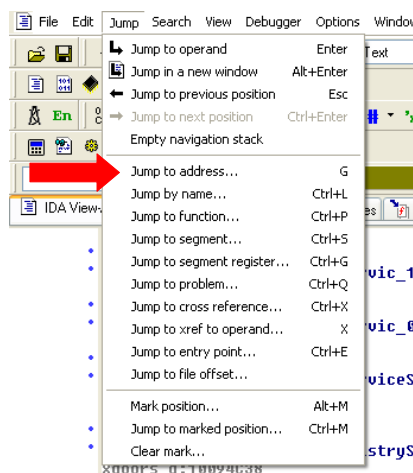
2) Individuare nella sezione **imports** l'indirizzo della funzione **<gethostbyname>**.  
Per cercare l'indirizzo di una funzione in **IDA** bisogna andare nella funzione **imports**.



E scorrendo nelle voci possiamo individuare quella che interessa a noi e facendo doppio click su di esso potremmo essere indirizzati in alla locazione di memoria della funzione : **100163CC**.

3) Quante sono le variabili locali della funzione alla locazione di memoria **0x10001656**.

Per cercare ad una specifica locazione di memoria bisogna usare sempre la funzione **jump** in questo caso nella sezione **address**.



```
.text:10001656
.text:10001656
.text:1000165C
.text:1000165D
.text:1000165E
.text:1000165F

sub esp, 678h
push ebx
push ebp
push esi
push edi
```

Ed ecco che alla locazione di memoria troveremo 4 variabili.

4) Quanti sono invece i parametri della funzione sopra.

Ed Ecco che scorrendo i parametri della funzione sopra che sono 35.

```
.text:10001656 sub_10001656 proc near ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
.text:10001656 var_6F6 = byte ptr -6F6h
.text:10001656 var_6F5 = byte ptr -6F5h
.text:10001656 var_6F4 = dword ptr -6F4h
.text:10001656 hModule = dword ptr -6F0h
.text:10001656 var_6E6 = dword ptr -6E6h
.text:10001656 Parameter = timeval ptr -6C8h
.text:10001656 CommandLine = byte ptr -6BFh
.text:10001656 Data = byte ptr -6B8h
.text:10001656 var_6A0 = dword ptr -6A0h
.text:10001656 var_682 = word ptr -682h
.text:10001656 var_67C = dword ptr -67Ch
.text:10001656 var_674 = dword ptr -674h
.text:10001656 var_670 = dword ptr -670h
.text:10001656 var_66C = word ptr -66Ch
.text:10001656 in = in_addr ptr -668h
.text:10001656 var_590 = dword ptr -590h
.text:10001656 var_558 = dword ptr -558h
.text:10001656 var_54C = dword ptr -54Ch
.text:10001656 var_548 = dword ptr -548h
.text:10001656 buf = byte ptr -53Ch
.text:10001656 readfds = fd_set ptr -518h
.text:10001656 var_3FC = dword ptr -3FCh
.text:10001656 var_3E4 = dword ptr -3E4h
.text:10001656 var_1F0 = dword ptr -1F0h
.text:10001656 var_1C0 = dword ptr -1C0h
.text:10001656 WSADATA = WSADATA ptr -190h
.text:10001656
.text:10001656 sub esp, 678h
```

5) Considerazioni macro sul malware.

Inserendo il file hash su virus total avremmo il seguente risultato:

50 / 67

50 security vendors and no sandboxes flagged this file as malicious

eb1079bdd96bc9cc19c38b76342113a09666aad47518ff1a7536eefb8aad4a

130.94 KB Size

2023-01-17 09:14:27 UTC 6 hours ago

X-doorc

pedll corrupt armadillo overlay

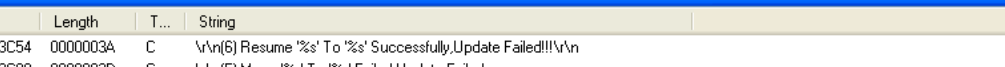
DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 19

Security vendors' analysis

AhnLab-V3	Backdoor:Win32.Agent.R9408	Alibaba	Backdoor:Win32/Idicaf.9f3a5556
ALYac	Backdoor.XIW	Antiy-AVL	Trojan[Backdoor]/Win32.Agent
Arcabit	Backdoor.XIW	Avast	Win32:Agent-OLH [Trj]
AVG	Win32:Agent-OLH [Trj]	Avira (no cloud)	BDS/Agen.twe.134160

Come si può vedere più provider segnalano questo file come malevolo e nello specifico come una **backdoor**.

Avendo questa informazione a nostra disposizione usando sempre la funzione **jump** su **IDA** in particolare cercando **backdoor** avremo un risultato, più precisamente nella sezione strings la seguente sigla: **BackDoor Server Update Setup**.



The screenshot shows the 'Strings window' of a debugger. It contains a table with columns: Address, Length, Type (T...), and String. The strings listed are various system messages and file names. A red arrow points to the string '\\\\BackDoor Server Update Setup\\' located at address xdoors\_d:10093D74.

Address	Length	T...	String
xdoors_d:10093C54	0000003A	C	\\n(6) Resume '%s' To '%s' Successfully.Update Failed!!!\\n\\n
xdoors_d:10093C90	0000002D	C	\\n(5) Move '%s' To '%s' Failed.Update Failed
xdoors_d:10093CC0	00000025	C	\\n(5) Copy '%s' To '%s' Successfully
xdoors_d:10093CE8	0000001C	C	\\n(4) Get New FileName '%s'
xdoors_d:10093D04	00000025	C	\\n(3) Move '%s' To '%s' Successfully
xdoors_d:10093D2C	00000006	C	.ubak
xdoors_d:10093D34	0000001C	C	\\n(2) Get DLL FileName '%s'
xdoors_d:10093D50	00000023	C	\\n(1) Enter Current Directory '%s'
xdoors_d:10093D74	00000067	C	\\n\\n\\\\BackDoor Server Update Setup\\'\\n\\n...
xdoors_d:10093DDC	00000006	C	-warn
xdoors_d:10093DE4	00000006	C	-erro
xdoors_d:10093DEC	00000006	C	-stop
xdoors_d:10093DF4	0000000A	C	-shutdown

Confermando il fatto che il nostro malware sia di fatto una backdoor.