Esercizio S11/L2

Malware analysis avanzata con IDA

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)

Punto 1

L’interactive disassembler, meglio conosciuto come IDA, è un software disassemblatore usato per il reverse engineering.

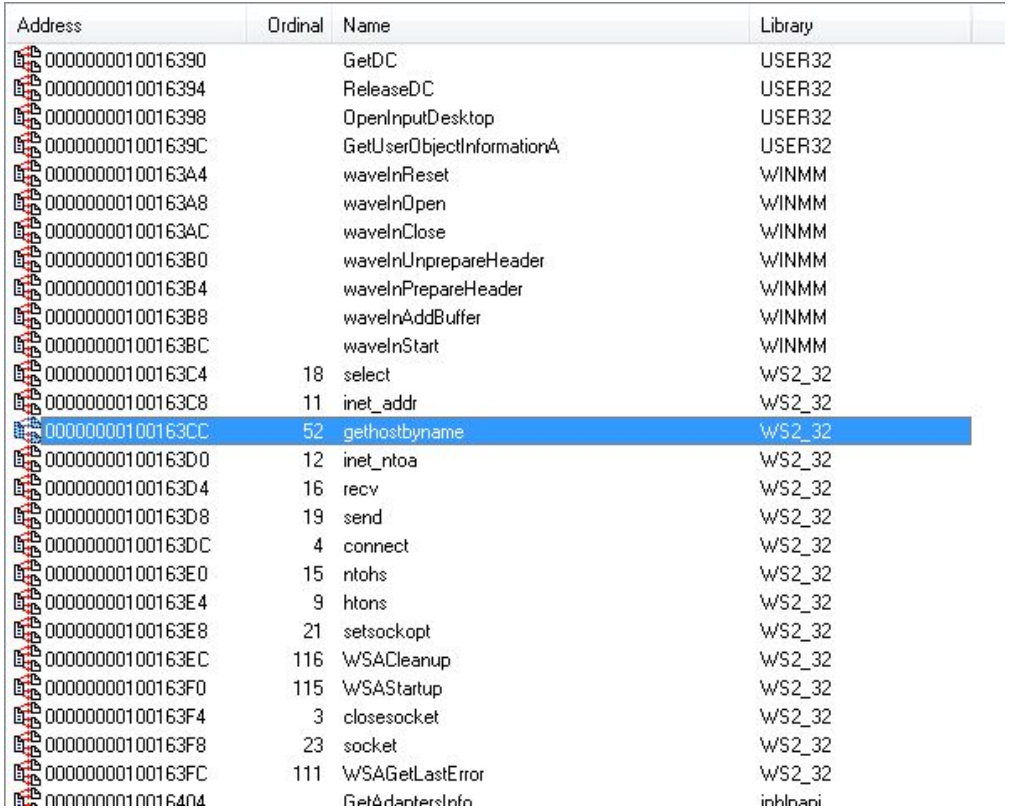
IDA ci fornisce il comando “jump” o “search” per trovare quello che che stiamo cercando nel codice esaminato. Come possiamo vedere dall’immagine sotto ci viene restituita la parte di codice cercata, nel nostro caso la funzione DLLMain si trova all’indirizzo 0x1000D02E.

```
.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
```

Punto 2

Eseguiamo l’operazione di ricerca della funzione “gethostbyname” come nel punto precedente. La funzione si trova all’indirizzo 100163CC nella sezione dei dati dati inizializzati (idata).

La funzione recupera le informazioni di un host, come ad esempio l’indirizzo IP, a partire da un nome host.



Address	Ordinal	Name	Library
0000000010016390		GetDC	USER32
0000000010016394		ReleaseDC	USER32
0000000010016398		OpenInputDesktop	USER32
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
00000000100163DC	4	connect	WS2_32
00000000100163E0	15	ntohs	WS2_32
00000000100163E4	9	htons	WS2_32
00000000100163E8	21	setsockopt	WS2_32
00000000100163EC	116	WSACleanup	WS2_32
00000000100163F0	115	WSAStartup	WS2_32
00000000100163F4	3	closesocket	WS2_32
00000000100163F8	23	socket	WS2_32
00000000100163FC	111	WSAGetLastError	WS2_32
0000000010016404		GetAdaptersInfo	iphlpapi

Punto 3

Variabili e parametri si distinguono in base all'offset rispetto al registro EBP. Le prime hanno un offset negativo, mentre le seconde hanno un offset positivo. Dalla figura accanto possiamo notare che sono presenti 23 variabili, mentre arg_0 è un parametro.

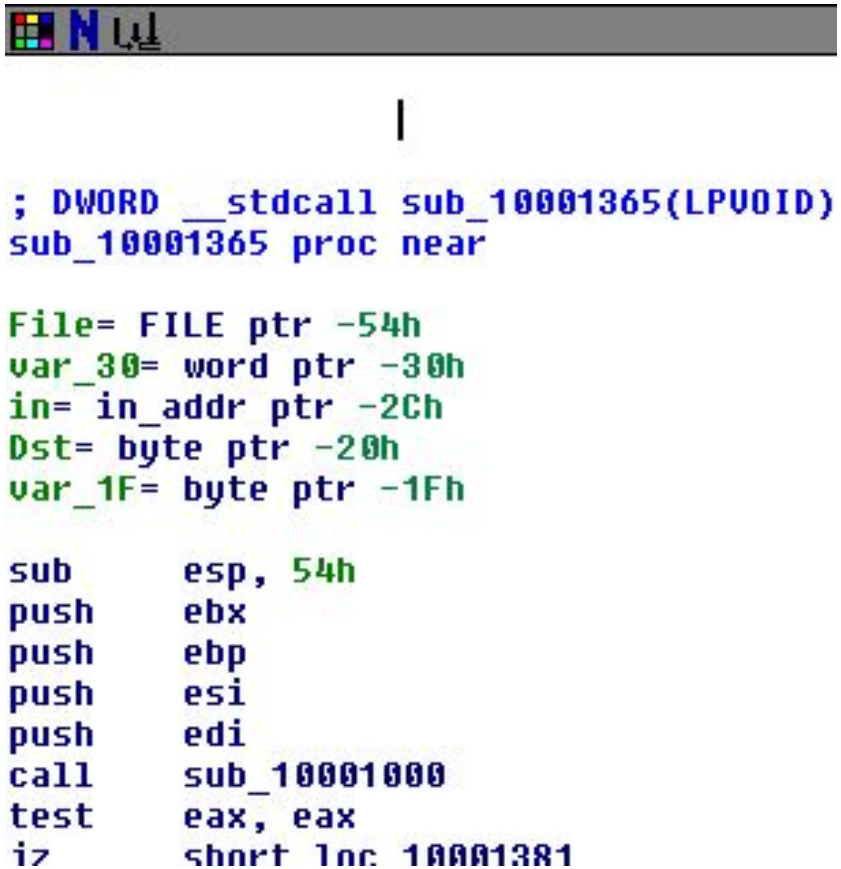
```
; DWORD __stdcall sub_10001656(LPVOID)
sub_10001656 proc near

var_675= byte ptr -675h
var_674= dword ptr -674h
hLibModule= dword ptr -670h
timeout= timeval ptr -66Ch
name= sockaddr ptr -664h
var_654= word ptr -654h
Dst= dword ptr -650h
Parameter= byte ptr -644h
var_640= byte ptr -640h
CommandLine= byte ptr -63Fh
Source= byte ptr -63Dh
Data= byte ptr -638h
var_637= byte ptr -637h
var_544= dword ptr -544h
var_50C= dword ptr -50Ch
var_500= dword ptr -500h
Buf2= byte ptr -4FCh
readfds= fd_set ptr -4BCh
phkResult= byte ptr -3B8h
var_3B0= dword ptr -3B0h
var_1A4= dword ptr -1A4h
var_194= dword ptr -194h
WSAData= WSAData ptr -190h
arg_0= dword ptr  4

sub     esp, 678h
push    ebx
push    ebp
push    esi
push    edi
call    sub_10001000
test    eax, eax
jnz     short loc_100016BC
```

Punto 4

La funzione all’indirizzo 10001365, sopra alla funzione precedente, in figura non presenta parametri, ma solo 5 variabili.



Punto 5

Dalle varie funzioni presenti nel malware, tra cui la raccolta informazioni, possiamo ipotizzare che si tratti di un RAT o un trojan. Inoltre come possiamo vedere è in grado di caricare librerie e creare thread remoti, che avvalora questa ipotesi.

```
call    ds:WriteProcessMemory
push    offset aLoadlibraryw ; "LoadLibraryW"
push    offset aKernel32_0 ; "Kernel32"
call    ds:GetModuleHandleA
push    eax             ; hModule
call    ds:GetProcAddress
push    esi             ; lpThreadId
push    esi             ; dwCreationFlags
push    edi             ; lpParameter
push    eax             ; lpStartAddress
push    esi             ; dwStackSize
push    esi             ; lpThreadAttributes
push    [ebp+hProcess]  ; hProcess
call    ds:CreateRemoteThread
```