Esercizio Epicode 05.12.2023

Analisi statica avanzata con IDA

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.

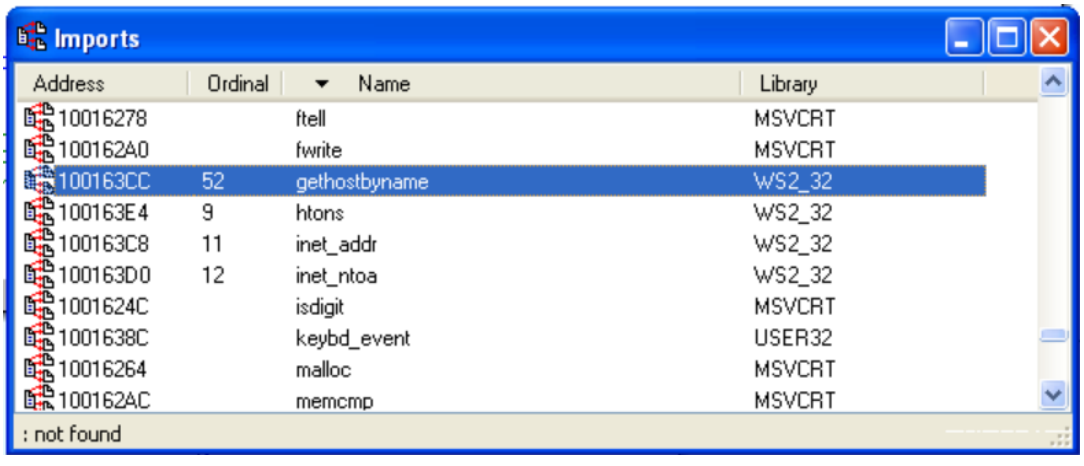
1. Individuare l'indirizzo della funzione DLLMain

Passando alla versione del codice assembly possiamo risalire all'indirizzo della funzione DLLMain, che equivale a **1000D02E**

```
.text:1000D02E
.text:1000D02E ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.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
```

2. Dalla scheda «imports» individuare la funzione «gethostbyname». Qual è l'indirizzo dell'import? Cosa fa la funzione?

In questo malware la funzione Gethostbyname potrebbe essere utilizzata per ottenere l'indirizzo IP di un server remoto, e tramite la scheda degli imports possiamo individuare l'indirizzo della funzione: **100163CC**




```
.idata:100163CC ; struct hostent *__stdcall gethostbyname(const char *name)
.idata:100163CC                 extrn gethostbyname:dword
.idata:100163CC                                         ; DATA XREF: sub_10001074:loc_100011AF↑r
.idata:100163CC                                         ; sub_10001074+1D3↑r ...
```

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

Risultano 20 variabili locali, con offset negativo, sull'allocazione di memoria 0x10001656

```
.text:10001656 var_675          = byte ptr -675h
.text:10001656 var_674          = dword ptr -674h
.text:10001656 hModule          = dword ptr -670h
.text:10001656 timeout          = timeval ptr -66Ch
.text:10001656 name             = sockaddr ptr -664h
.text:10001656 var_654          = word ptr -654h
.text:10001656 in               = in_addr ptr -650h
.text:10001656 Parameter        = byte ptr -644h
.text:10001656 CommandLine      = byte ptr -63Fh
.text:10001656 Data             = byte ptr -638h
.text:10001656 var_544          = dword ptr -544h
.text:10001656 var_50C          = dword ptr -50Ch
.text:10001656 var_500          = dword ptr -500h
.text:10001656 var_4FC          = dword ptr -4FCh
.text:10001656 readfds          = fd_set ptr -4BCh
.text:10001656 phkResult        = HKEY__ 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
.text:10001656 arg_0            = dword ptr  4
```

4. Quanti sono, invece, i parametri della funzione sopra?

L'unico parametro sull'allocazione di memoria precedente è arg_0

```
.text:10001656 arg_0            = dword ptr  4
.text:10001656
.text:10001656                  sub     esp, 678h
```

5. Inserire altre considerazioni macro-livello sul malware

Controllando sia il codice assembly in formato testo che verificando tramite virus total il codice hash del malware, possiamo dedurre il file sia una backdoor

```
xdoors_d:10093D74 ; char aBackdoorServer[]
xdoors_d:10093D74 aBackdoorServer db 0Dh,0Ah             ; DATA XREF: sub_100042DB+B5↑o
xdoors_d:10093D74                 db 0Dh,0Ah
xdoors_d:10093D74                 db '*****************************',0Dh,0Ah
xdoors_d:10093D74                 db '[BackDoor Server Update Setup]',0Dh,0Ah
xdoors_d:10093D74                 db '*****************************',0Dh,0Ah
xdoors_d:10093D74                 db 0Dh,0Ah,0
```