**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)



## 1. Individuazione dell'indirizzo della funzione DLLMain

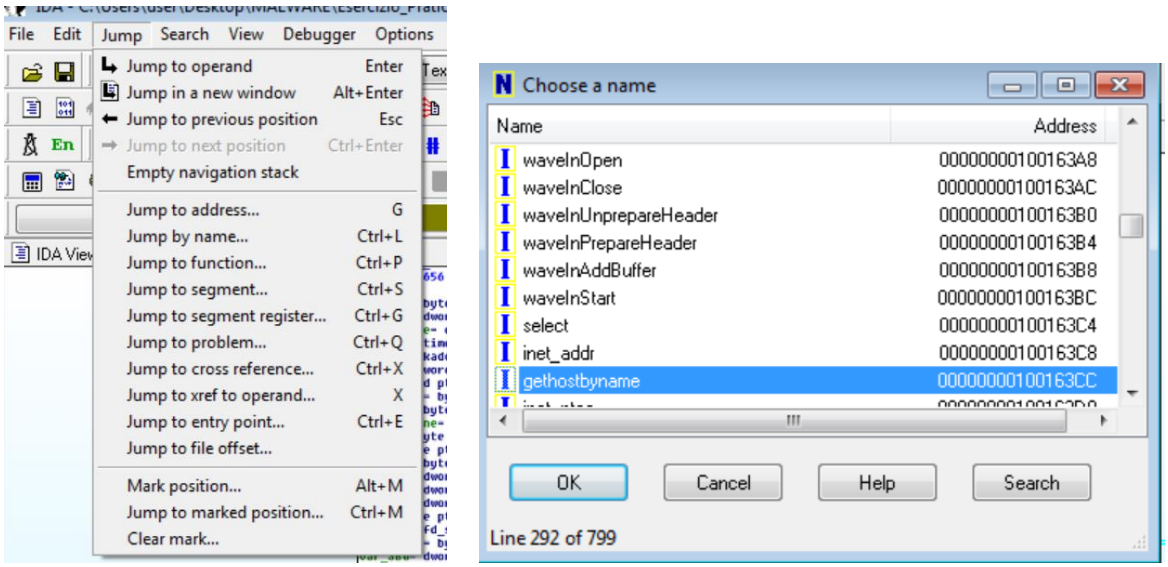
Oggi ci occuperemo di Analisi Statica Avanzata nel settore della malware analysis. Analizzeremo un malware chiamato Malware_U3_W3_L2.dll, partendo dalle sue istruzioni in Assembly. Per fare ciò, utilizzeremo il tool IDA Pro, che è un disassembler che traduce il linguaggio macchina di un eseguibile in linguaggio Assembly e ha molte funzionalità. Analizzeremo il malware all'interno della VM Malware_Analysis, utilizzando il sistema operativo Windows XP SP3.

| Function name | Segment | Start | Length | R | F | L | S | B | T | = |
|---|---|---|---|---|---|---|---|---|---|---|
| sub_1000C62D | .text | 000000001000C62D | 0000010D | R | . | . | . | B | T | . |
| sub_1000C73A | .text | 000000001000C73A | 000001B0 | R | . | . | . | B | T | . |
| sub_1000C8EA | .text | 000000001000C8EA | 000000F5 | R | . | . | . | B | T | . |
| HandlerProc | .text | 000000001000C9DF | 00000077 | R | . | . | . | . | T | . |
| sub_1000CA56 | .text | 000000001000CA56 | 000001B0 | R | . | . | . | B | . | . |
| sub_1000CC06 | .text | 000000001000CC06 | 0000032A | R | . | . | . | B | T | . |
| ServiceMain | .text | 000000001000CF30 | 000000FE | R | . | . | . | B | T | . |
| DllMain(x,x,x) | .text | 000000001000D02E | 000000DF | R | . | . | . | . | T | . |
| sub_1000D10D | .text | 000000001000D10D | 000000C6 | R | . | . | . | B | T | . |

Abbiamo identificato la funzione DLLMain; l'indirizzo di memoria ad essa associato è **1000D02E**

**2. Individuazione dell'indirizzo di import della funzione gethostbyname**

In questo passaggio, stiamo cercando la funzione gethostbyname all'interno delle funzioni importate dell'eseguibile. Questa funzione può essere trovata anche nella scheda "imports". Per trovarla, abbiamo deciso di utilizzare la funzionalità jump by name e fare una ricerca del nome della funzione cliccando su "Search".




Abbiamo individuato la funzione ricercata, presente all'indirizzo di memoria **1001063CC**.

**3. Quantificazione delle variabili lo In questa fase, vogliamo analizzare la funzione presente all'indirizzo di memoria 0x10001656.**

Per farlo, utilizzeremo la funzionalità Jump to address e inseriremo l'indirizzo che ci interessa nella barra di ricerca. cali e dei parametri della funzione alla locazione di memoria 0x10001656

```
.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 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
```

variabili locali

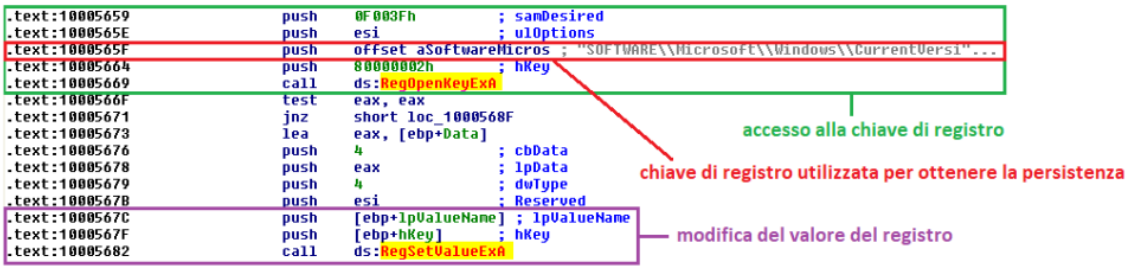
parametro

Dopo la nostra ricerca, abbiamo trovato la funzione di tipo subroutine sub_10001656, che ha 20 variabili locali e un parametro. È importante notare che il tool IDA distingue tra variabili e parametri utilizzando come riferimento l'offset rispetto al puntatore EBP. In particolare, le variabili si trovano

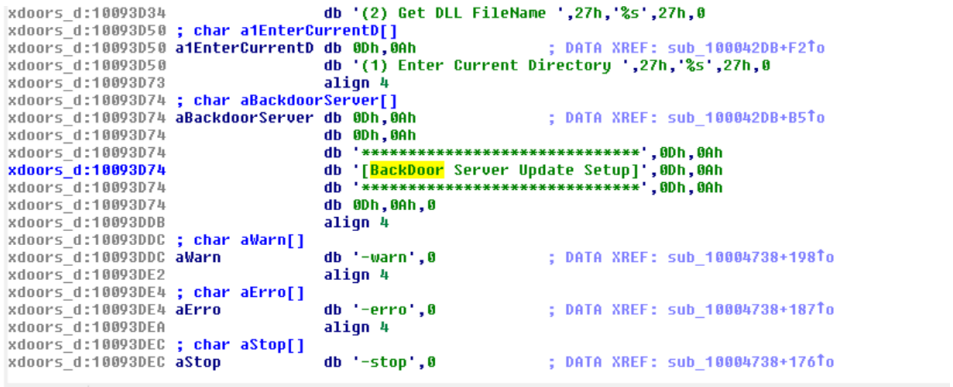
ad un offset negativo rispetto al registro EBP, mentre i parametri si trovano ad un offset positivo rispetto al registro EBP.

### 4. Considerazioni macro-livello circa il comportamento del malware

In questo passaggio, stiamo ipotizzando la finalità del malware e verificando se esso ha lo scopo di ottenere la persistenza all'interno del sistema della macchina vittima. Il malware utilizza la funzione RegOpenKeyEx per accedere alla chiave di registro e la funzione RegSetValueEx per modificare il valore del registro e aggiungere una nuova entry, al fine di ottenere la persistenza all'avvio del sistema operativo. Una delle chiavi di registro utilizzate dal malware per ottenere la persistenza è Software\\Microsoft\\Windows\\CurrentVersion\\Run.




In questo passaggio, abbiamo confermato la presenza di una richiesta da parte del malware di ottenere la persistenza all'interno della macchina vittima. Possiamo quindi ipotizzare che il malware sia una backdoor che necessita di essere in esecuzione continua sulla macchina target. Abbiamo cercato la parola chiave "backdoor" all'interno della sezione di IDA dedicata alle stringhe, utilizzando la funzionalità di ricerca "Search":




La ricerca della parola chiave "backdoor" ha avuto esito positivo, come confermato dal passaggio precedente. Possiamo quindi ipotizzare l'esistenza di una remote shell che il malware vuole mantenere in esecuzione. Possiamo procedere a una nuova ricerca tra le stringhe presenti all'interno dell'eseguibile, usando la funzionalità di ricerca "Search":

```
xdoors_d:10095B20 ; char aCommand_exeC[]
xdoors_d:10095B20 aCommand_exeC   db '\command.exe /c ',0 ; DATA XREF: sub_1000FF58:loc_100101D7↑o
xdoors_d:10095B31                 align 4
xdoors_d:10095B34 aCmd_exeC       db '\cmd.exe /c ',0     ; DATA XREF: sub_1000FF58+278↑o
xdoors_d:10095B41                 align 4
xdoors_d:10095B44 ; char aHiMasterDDDDDD[]
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
xdoors_d:10095B44                                         ; DATA XREF: sub_1000FF58+145↑o
xdoors_d:10095B44                 db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
xdoors_d:10095B44                 db 0Dh,0Ah
xdoors_d:10095B44                 db 'Machine UpTime  [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Secon'
xdoors_d:10095B44                 db 'ds]',0Dh,0Ah
xdoors_d:10095B44                 db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44                 db 'nds]',0Dh,0Ah
xdoors_d:10095B44                 db 0Dh,0Ah
xdoors_d:10095B44                 db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095B44                 db 0Dh,0Ah,0
```

Anche in questo caso la ricerca ha esito **positivo**.