

S11-L2

(Bonaldi Cristian)



Malware analysis - Analisi statica avanzata con IDA

Traccia:

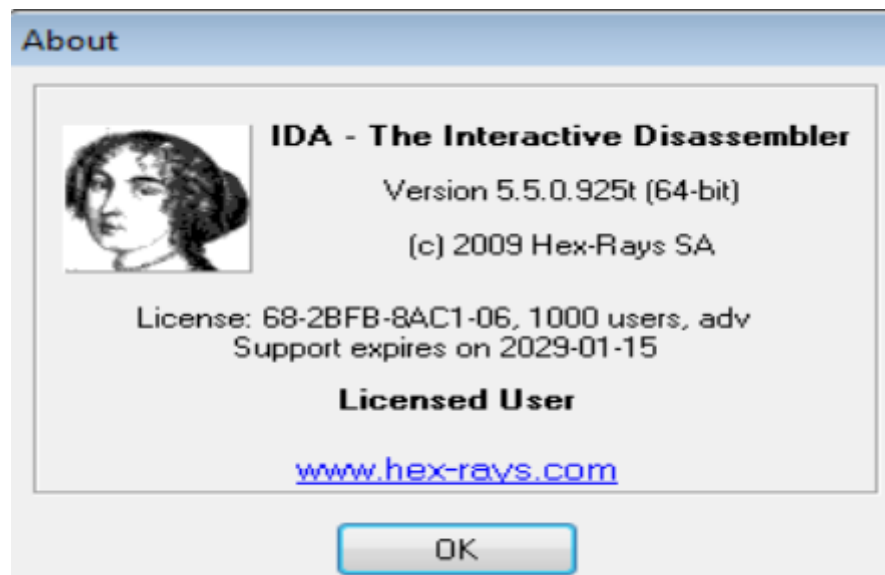
Lo scopo dell'esercizio di oggi è di acquisire esperienza con IDA, un tool fondamentale per l'analisi statica. A tal proposito, con riferimento al malware chiamato «Malware_U3_W3_L2» presente all'interno della cartella «Esercizio_Pratico_U3_W3_L2» sul Desktop della macchina virtuale dedicata all'analisi 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).

Esercizio e sviluppo:

Lo scopo dell'esercizio pratico di oggi consiste nell'analisi statica avanzata di un malware, mediante l'utilizzo del tool IDA Pro, uno

strumento molto utile per il reverse engineering che ci permette di disassemblare il codice binario e analizzare il comportamento di programmi e malware senza eseguirli.



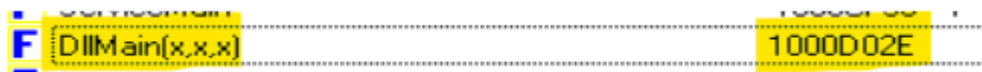
Vengono richieste diverse operazioni di analisi e procediamo dunque nello svolgimento:

1. Individuare l'indirizzo della funzione DLLMain.

Per fare questo tipo di operazione, prima di tutto avviamo IDA e importiamo il file **Malware_U3_W3_L2**.

All'interno della sezione *Names Windows* di IDA ci viene fornita una lista completa di tutti i nomi definiti nel codice. I nomi vengono classificati con un'etichetta di fianco che specifica se fa riferimento a funzioni, variabili, dati ecc.

In questo caso `DllMain` è una funzione definita nel codice (F) e l'indirizzo di memoria è `0x1000D02E`.



Con un doppio click sulla funzione IDA ci porterà a navigare alla posizione specifica in memoria dove viene definita la funzione `DllMain`. Questa è una subroutine che viene chiamata dalla funzione `DllEntryPoint`.

```
.text:1000D02E ; ===== S U B R O U T I N E =====
.text:1000D02E
.text:1000D02E
.text:1000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL
.text:1000D02E _DllMain@12      proc near
.text:1000D02E
.text:1000D02E
.text:1000D02E

push    edi
push    esi
push    ebx
call     _DllMain@12

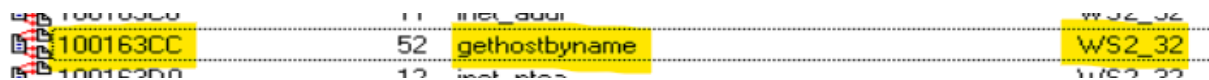
; CODE XREF: DllEntryPoint+24↑j
; DllEntryPoint:loc_100151AF↑j
; lpvReserved
; fdwReason
; hinstDLL
; DllMain(x,x,x)
```

2. Funzione «gethostbyname» e cosa fa

La funzione gethostbyname fa parte delle funzioni importate dalla libreria WS2_32.dll, trovata all'interno della sezione *Imports* di IDA.

Viene utilizzata per convertire nomi di dominio in indirizzi ip.

Un malware ad esempio può utilizzarla per scoprire l'indirizzo IP di un server remoto a partire da un nome di dominio.



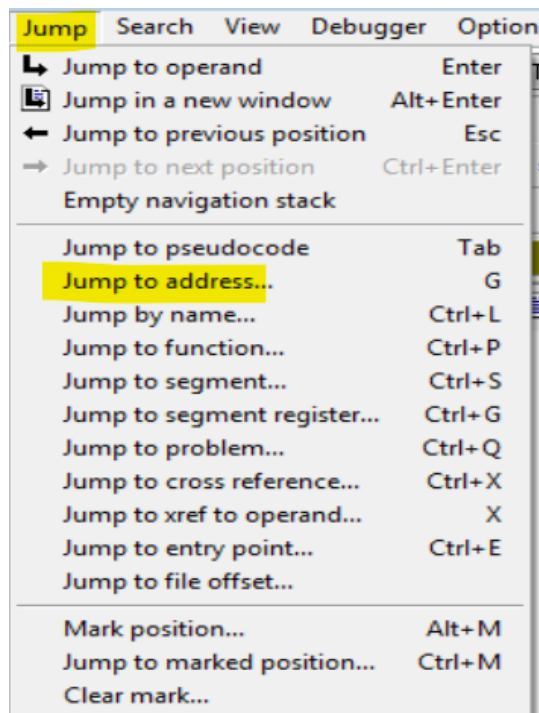
00103000	11	include	WS2_32
001063CC	52	gethostbyname	WS2_32
001063D0	12	inet_addr	WS2_32

L'indirizzo dell'import è 0x100163CC.

```
ta:100163CC ; struct hostent * _stdcall gethostbyname(const char *name)
ta:100163CC      extrn gethostbyname:dword
ta:100163CC      ; CODE XREF: sub_10001074:loc_100011AF↑p
```

3. Variabili locali della funzione alla locazione di memoria 0x10001656

Per trovare le variabili locali della funzione all'indirizzo 0x10001656, utilizziamo lo strumento *Jump to address* nella sezione *Jump* di IDA. Questo ci permette di effettuare un salto all'indirizzo di memoria specificato sopra.



Inseriamo l'indirizzo all'interno della barra di ricerca e procediamo ad identificare le variabili della funzione, che vengono riportate

```
.text:10001656
.text:10001656 ; ===== S U B R O U T I N E ==
.text:10001656
|.text:10001656
.text:10001656 ; DWORD __stdcall sub_10001656(LPVOID)
.text:10001656 sub_10001656      proc near
.text:10001656
.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
```

in figura (in totale 23). Il parametro invece, risulta essere 1, ovvero:

```
.text:10001656 arg_0 = dword ptr 4
```

5. Comportamento malware

Dall'analisi effettuata durante lo studio del file malevolo, possiamo dedurre chiaramente il comportamento del malware.

In particolare, l'analisi delle librerie importate dal malware evidenzia l'uso di funzioni chiave come `recv` e `send` dalla libreria **WS2_32.dll** (Windows Sockets API), che sono utilizzate per stabilire una connessione con un server remoto.

La funzione `recv` consente al malware di ricevere dati dal server, mentre `send` per inviare dati. Quindi il malware si mette in ascolto, instaurando una comunicazione con il server remoto, agendo così da **backdoor**.

	100163...	16	recv	WS2_32
	100163...	19	send	WS2_32

In più va ad effettuare una serie di operazioni nelle chiavi di registro per garantire persistenza all'interno della macchina, anche dopo il riavvio. Questa è una tecnica comune sfruttata dai malware per evitare che vengano rimossi facilmente o rilevati durante un normale riavvio di sistema.

OpenProcessToken	ADVAPI32
RegCloseKey	ADVAPI32
RegQueryValueExA	ADVAPI32
RegOpenKeyExA	ADVAPI32
CreateProcessAsUserA	ADVAPI32
RegSetValueExA	ADVAPI32
RegDeleteValueA	ADVAPI32
RegEnumKeyA	ADVAPI32
RegOpenKeyA	ADVAPI32
SetTokenInformation	ADVAPI32
DuplicateTokenEx	ADVAPI32
RegEnumValueA	ADVAPI32
AdjustTokenPrivileges	ADVAPI32
RegCreateKeyA	ADVAPI32
RegDeleteKeyA	ADVAPI32
CloseServiceHandle	ADVAPI32
