

S11L2

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)

SVOLGIMENTO

Il codice assembly di un malware si può recuperare avendo a disposizione l'eseguibile e un disassembler che traduce le istruzioni da linguaggio macchina a linguaggio assembly. Senza disassembler non sarebbe possibile ricavare il linguaggio assembly da un file eseguibile, e di conseguenza non sarebbe possibile procedere con l'analisi statica avanzata.

IDA Pro è un potente disassembler che supporta molti formati di file eseguibili. Questo strumento riesce a mettere a disposizione degli analisti una serie di caratteristiche intuitive per semplificare le attività. Infatti, oltre alla traduzione completa del linguaggio macchina di un eseguibile in linguaggio assembly, IDA identifica:

- Funzioni / chiamate di funzione
- Analisi dello stack
- Variabili locali e parametri

1. Funzione DLLMain

Utilizzando IDA Pro è stata identificata la funzione DLLMain attraverso il <disassembly panel>, che ci mostra la traduzione del codice macchina dell'eseguibile in codice Assembly.

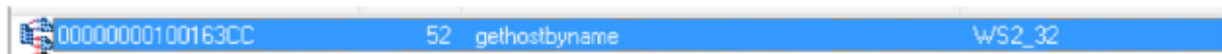
```
.text:1000002E ; ===== S U B R O U T I N E =====
.text:1000002E
.text:1000002E
.text:1000002E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:1000002E _DllMain@12      proc near                                ; CODE XREF: DllEntryPoint+48↓p
.text:1000002E                                           ; DATA XREF: sub_100110FF+2D↓o
.text:1000002E
.text:1000002E hinstDLL      = dword ptr  4
.text:1000002E fdwReason     = dword ptr  8
.text:1000002E lpvReserved  = dword ptr 0Ch
```

2. Individuazione funzione <gethostbyname>

Dal disassembly panel in modalità testuale (ottenuta premendo la barra spaziatrice mentre si è nella modalità grafica), si è trovata la funzione <gethostbyname>, come si può vedere nella figura sottostante.

```
.idata:100163C8 ; unsigned __int32 __stdcall inet_addr(const char *cp)
.idata:100163C8          extrn inet_addr:dword      ; CODE XREF: sub_10001074+11E1p
.idata:100163C8          ; sub_10001074+1BF1p ...
.idata:100163CC ; struct hostent *__stdcall gethostbyname(const char *name)
.idata:100163CC          extrn gethostbyname:dword
.idata:100163CC          ; CODE XREF: sub_10001074:loc_100011AF1p
.idata:100163CC          ; sub_10001074+1D31p ...
.idata:100163D0 ; char *__stdcall inet_ntoa(struct in_addr in)
.idata:100163D0          extrn inet_ntoa:dword      ; CODE XREF: sub_10001074:loc_100013111p
.idata:100163D0          ; sub_10001365:loc_100016021p ...
```

In particolare, nella finestra “imports” è stata identificata la funzione suddetta, come mostrato nella prima colonna della figura seguente.



La funzione <gethostbyname> è una funzione utilizzata per ottenere informazioni sulle risorse di rete tramite il nome host: converte quindi un nome host in un indirizzo IP.

Dando in input alla funzione <gethostbyname> un nome host, essa restituisce una struttura di tipo 'hostnet' che contiene informazioni sull'host, in particolare sul suo indirizzo IP ed eventuali indirizzi associati.

Tale funzione viene utilizzata spesso nelle applicazioni di rete prima di stabilire una connessione di rete.

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

Cercando all'interno del disassembly panel, in modalità testuale, si sono trovate 23 variabili locali, come mostrato in figura:

```
.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
.text:10001656 readfds      = fd_set ptr -48Ch
.text:10001656 phkResult    = byte 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
```

Osservando bene la lista mostrata, e riprendendo la teoria sull'identificazione delle variabili notiamo due cose:

- Le variabili sono ad un offset negativo rispetto al registro EBP.
- I parametri si trovano ad un offset positivo rispetto ad EBP.

dove con offset si intende la differenza rispetto ad un valore di riferimento, che in generale è il registro EBP.

In questo caso, si nota dal testo in verde che i primi 23 dati sono identificabili come VARIABILI, in quanto presentano un offset negativo.

4. Parametri della funzione alla locazione di memoria 0x10001656












Per contro, l'ultimo dato rappresentato è identificato come PARAMETRO, poiché presenta un offset positivo.

```
.text:10001656 WSAData          = WSAData ptr -190h  
.text:10001656 arg_0           = dword ptr  4
```

5. Considerazioni sul comportamento del malware

Osservando il codice del malware si possono ricavare alcune informazioni sul suo comportamento:

- l'import di alcune librerie per modificare le chiavi di registro:

	1001637C	RedrawWindow	USER32
	10016008	RegCloseKey	ADVAPI32
	10016038	RegCreateKeyA	ADVAPI32
	1001603C	RegDeleteKeyA	ADVAPI32
	1001601C	RegDeleteValueA	ADVAPI32
	10016020	RegEnumKeyA	ADVAPI32
	10016030	RegEnumValueA	ADVAPI32
	10016024	RegOpenKeyA	ADVAPI32
	10016010	RegOpenKeyExA	ADVAPI32
	1001600C	RegQueryValueExA	ADVAPI32
	10016018	RegSetValueExA	ADVAPI32

- l'import delle librerie per maneggiare file:

	000000...	CopyFileA
	000000...	MoveFileExA



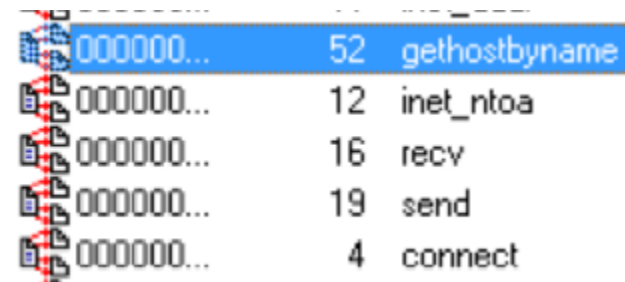
00000000... WriteFile

- l'import della libreria socket:



00000000... 23 socket

- l'import delle librerie per effettuare connessioni, inviare e ricevere dati:



00000000...	52	gethostbyname
00000000...	12	inet_ntoa
00000000...	16	recv
00000000...	19	send
00000000...	4	connect

Dall'import di tutte queste librerie si può dedurre che il malware stabilisca una connessione con l'esterno e permetta ad un utente malevolo di effettuare operazioni sul sistema attaccato. Questo comportamento lascia pensare ad una backdoor che permette ad un attaccante di effettuare diverse operazioni sulla macchina attaccata senza la necessità di inserire credenziali.