

CS0424IT — ESERCITAZIONE S11L2
ANALISI MALWARE IDA PRO

Simone La Porta



20 agosto 2024

INDICE

| | | |
|-----|--|---|
| 1 | TRACCIA | 3 |
| 2 | SVOLGIMENTO | 4 |
| 2.1 | Indirizzo funzione DLLMain | 4 |
| 2.2 | Individuazione della funzione gethostbyname | 4 |
| 2.3 | Variabili locali della funzione alla locazione di memoria 0x10001656 | 6 |
| 2.4 | Parametri della funzione alla locazione di memoria 0x10001656 | 6 |
| 2.5 | Considerazioni sul comportamento del malware | 7 |

1 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`, 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 a macro livello sul malware (comportamento).

2 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, tra cui:

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

2.1 Indirizzo funzione *DLLMain*

Utilizzando IDA Pro, è stata identificata la funzione `DLLMain` attraverso il `disassembly panel`, che mostra la traduzione del codice macchina dell'eseguibile in codice Assembly. L'indirizzo esadecimale della funzione `DLLMain` è stato individuato ed è `0x1000D02E`.

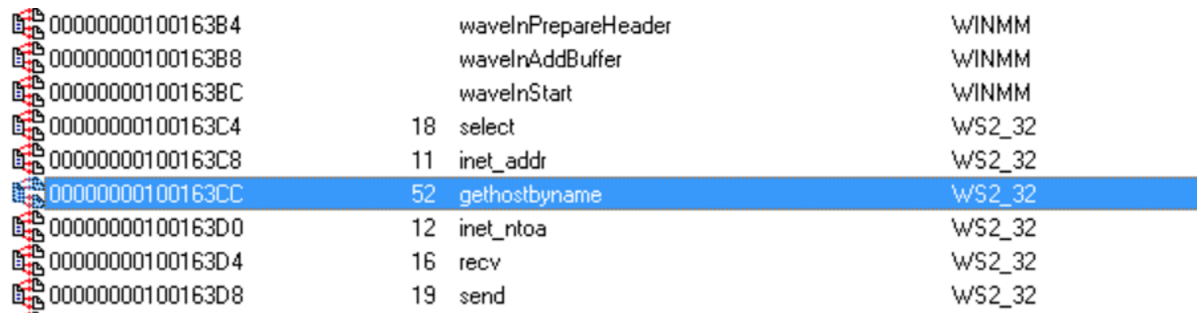
```
.text:1000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:1000D02E _DllMain@12      proc near                                ; CODE XREF: DllEntryPoint+4B↓p
.text:1000D02E                                     ; DATA XREF: sub_100110FF+2D↓o
.text:1000D02E hinstDLL      = dword ptr  4
.text:1000D02E fdwReason    = dword ptr  8
.text:1000D02E lpvReserved  = dword ptr 0Ch
.text:1000D02E
.text:1000D02E      mov     eax, [esp+fdwReason]
```

Figura 1: Indirizzo della funzione `DLLMain`

2.2 Individuazione della funzione *gethostbyname*

Dalla scheda `disassembly panel` in modalità testuale (ottenuta premendo la barra spaziatrice mentre si è nella modalità grafica), si è trovata la funzione `gethostbyname`, come mostrato nella figura sottostante. In particolare, nella finestra `imports` è stata identificata la funzione suddetta.

L'indirizzo dell'import della funzione `gethostbyname` è `0x100163CC`.



| | | |
|------------------|---------------------|--------|
| 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 |

Figura 2: Identificazione della funzione `gethostbyname`

La funzione `gethostbyname` è 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.

2.3 Variabili locali della funzione alla locazione di memoria 0x10001656

Cercando all'interno del disassembly panel, in modalità testuale, si sono trovate 23 variabili locali associate alla funzione alla locazione di memoria 0x10001656, come mostrato in figura:

```
.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 ; 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 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 -4BCh
.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
.text:10001656
.text:10001656 sub esp, 678h
```

Figura 3: Variabili locali individuate nel disassembly panel

Le variabili sono identificate da un offset negativo rispetto al registro EBP, che indica la loro natura di variabili locali, a differenza dei parametri che si trovano ad un offset positivo rispetto a EBP.

2.4 Parametri della funzione alla locazione di memoria 0x10001656

All'interno della stessa funzione, l'ultimo dato rappresentato è identificato come parametro della funzione, poiché presenta un offset positivo rispetto al registro EBP.
















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

Figura 4: Parametri individuati nel disassembly panel



2.5 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;

| Address | Ordinal | Name | Library |
|--|---------|-----------------------|----------|
|  0000000010016000 | | LookupPrivilegeValueA | ADVAPI32 |
|  0000000010016004 | | OpenProcessToken | ADVAPI32 |
|  0000000010016008 | | RegCloseKey | ADVAPI32 |
|  000000001001600C | | RegQueryValueExA | ADVAPI32 |
|  0000000010016010 | | RegOpenKeyExA | ADVAPI32 |
|  0000000010016014 | | CreateProcessAsUserA | ADVAPI32 |
|  0000000010016018 | | RegSetValueExA | ADVAPI32 |
|  000000001001601C | | RegDeleteValueA | ADVAPI32 |
|  0000000010016020 | | RegEnumKeyA | ADVAPI32 |
|  0000000010016024 | | RegOpenKeyA | ADVAPI32 |
|  0000000010016028 | | SetTokenInformation | ADVAPI32 |
|  000000001001602C | | DuplicateTokenEx | ADVAPI32 |
|  0000000010016030 | | RegEnumValueA | ADVAPI32 |
|  0000000010016034 | | AdjustTokenPrivileges | ADVAPI32 |
|  0000000010016038 | | RegCreateKeyA | ADVAPI32 |

- L'import delle librerie per maneggiare file;

| | | | |
|--|--|-------------|----------|
|  0000000010016104 | | CopyFileA | KERNEL32 |
|  0000000010016108 | | MoveFileExA | KERNEL32 |

- L'import della libreria socket;

| | | | |
|--|----|--------|--------|
|  00000000100163F8 | 23 | socket | WS2_32 |
|--|----|--------|--------|

- L'import delle librerie per effettuare connessioni, inviare e ricevere dati.

| | | | |
|--|----|---------------|--------|
|  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 |

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.