

# ANALISI STATICÀ AVANZATA CON IDA



# 1. Funzione DLLMain

Individuare l'indirizzo della funzione DLLMain:

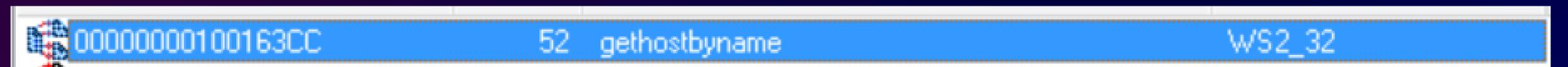
```
|.text:1000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:1000D02E _DllMain@12    proc near             ; CODE XREF: DllEntryPoint+4B↓p
.text:1000D02E                 ; DATA XREF: sub_100110FF+2D↓o
.text:1000D02E
.text:1000D02E     hinstDLL      = dword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

# 2. Funzione gethostbyname

Qual è l'indirizzo dell'import?

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

Cosa fa la funzione?



La funzione `gethostbyname` della libreria **WS2\_32** è una funzione utilizzata per ottenere informazioni sulle risorse di rete tramite il nome host. In pratica, converte un nome host in un indirizzo IP.

Quando si passa un nome host alla funzione `gethostbyname`, essa restituisce una struttura di tipo `hostent` che contiene informazioni sull'host, inclusi uno o più indirizzi IP associati a quel nome host. Questa funzione è spesso utilizzata nelle applicazioni di rete per risolvere nomi host in indirizzi IP prima di stabilire una connessione di rete.

# 3. Variabili locali della funzione alla locazione 0x10001656

Recandoci presso l'indirizzo 0x10001656 possiamo vedere, come dall'immagine a fianco che ci sono 23 variabili locali

```
var_675= byte ptr -675h
var_674= dword ptr -674h
hLibModule= dword ptr -670h
timeout= timeval ptr -66Ch
name= sockaddr ptr -664h
var_654= word ptr -654h
Dst= dword ptr -650h
Parameter= byte ptr -644h
var_640= byte ptr -640h
CommandLine= byte ptr -63Fh
Source= byte ptr -63Dh
Data= byte ptr -638h
var_637= byte ptr -637h
var_544= dword ptr -544h
var_50C= dword ptr -50Ch
var_500= dword ptr -500h
Buf2= byte ptr -4FCh
readfds= fd_set ptr -4BCh
phkResult= byte ptr -3B8h
var_3B0= dword ptr -3B0h
var_1A4= dword ptr -1A4h
var_194= dword ptr -194h
WSAData= WSAData ptr -190h
```

# 4. Quanti sono, invece, i parametri

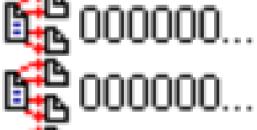
Recandoci presso l'indirizzo  
0x10001656 possiamo vedere,  
come dall'immagine a fianco che  
c'è un solo parametro

```
arg_0= dword ptr 4
```

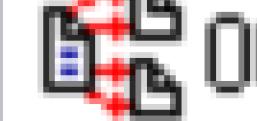
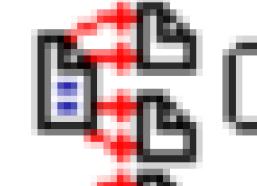
# 5. Considerazioni sul comportamento del malware

Analizzando il malware possiamo notare che effettua svariate operazioni tra le quali notiamo:

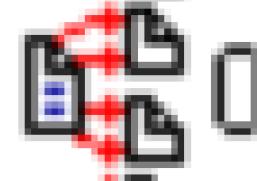
- L'import delle librerie per modificare le chiavi di registro;

 000000...	RegCloseKey	ADVAPI32
 000000...	RegQueryValueExA	ADVAPI32
 000000...	RegOpenKeyExA	ADVAPI32

- L'import delle librerie per maneggiare file;

 000000...	CopyFileA
 000000...	MoveFileExA
 000000...	WriteFile

- L'import della libreria socket;

 000000...	23 socket
---	-----------

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

000000...	52	gethostbyname
000000...	12	inet_ntoa
000000...	16	recv
000000...	19	send
000000...	4	connect

Basandosi su questi dati si può supporre che il malware stabilisca una connessione con l'esterno e permetta ad un utente malevolo di effettuare operazioni sul sistema. Tale comportamento può essere riconducibile a quello di una backdoor che permette ad un utente remoto di effettuare operazioni senza bisogno di effettuare un login lecito, comportamento simile a quello visto nelle settimane passate nelle esercitazioni con Meterpreter.

Ricavato l'hash MD5, effettuando una ricerca online su virus total possiamo verificare che il malware è catalogato proprio come backdoor.

