



S11/L2

Made By : Noemi de Martino

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.

- Individuare **l'indirizzo** della funzione **DLLMain** (così com'è, in esadecimale)
 - Dalla scheda «imports» individuare la funzione «**gethostbyname** ». Qual è l'indirizzo dell'import? **Cosa fa la funzione?**
 - Quante sono le **variabili locali** della **funzione** alla locazione di memoria 0x10001656?
 - Quanti sono, invece, i **parametri** della funzione sopra?
 - Inserire altre considerazioni macro livello sul malware (comportamento)

1. FUNZIONE DDLMAIN

Attraverso l'utilizzo di **IDA Pro**, è stato individuata la funzione **DLLMain** utilizzando il **disassembly panel**, il quale fornisce la traduzione del codice macchina dell'eseguibile in codice Assembly.

```
.text:1000D02E ; ===== S U B R O U T I N E =====
.text:1000D02E
.text:1000D02E
.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
```

2. INDIVIDUAZIONE FUNZIONE <GETHOSTBYNAME>

Nel pannello di **disassembly** in modalità testuale è stata individuata la funzione **gethostbyname**, come illustrato nella figura:

```
.idata:100163C8 ; sub_10001074+1BF↑p ...  
.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:100163D0 ; char *__stdcall inet_ntoa(struct in_addr in)  
.idata:100163D0         extrn inet_ntoa:dword ; CODE XREF: sub_10001074:loc_10001311↑p  
.idata:100163D0         ; sub_10001365:loc_10001602↑p ...
```

In particolare, nella finestra "**imports**" è stata individuata la funzione, come illustrato nella figura.

	100163C8	11	inet_addr	WS2_32
	100163...	52	gethostbyname	WS2_32
	100163...	12	inet_ntoa	WS2_32

Nell'ambito dell'analisi di malware, la funzione **gethostbyname** viene spesso utilizzata per risolvere il nome di dominio di un server di comando e controllo.

Questa funzione converte il nome dell'host fornito in input in un indirizzo IP, consentendo al malware di stabilire una connessione verso il server per ricevere comandi o trasmettere dati rubati.

La funzione restituisce una struttura di tipo **hostent**, contenente l'indirizzo IP dell'host e altri eventuali indirizzi associati, facilitando così la comunicazione con il server maligno.

3. VARIBILI LOCALI DELLA FUNZIONE ALLA LOCAZIONE DI MEMORIA 0X10001656

Nel contesto dell'esplorazione del suggestivo universo del pannello di smontaggio, abbiamo individuato un totale di **23 variabili locali**, come chiaramente rappresentato nella figura allegata.

Osservando la lista fornita e facendo riferimento alla teoria sull'identificazione delle variabili, si possono trarre due considerazioni principali:

1. Le variabili sono posizionate a un offset negativo rispetto al registro EBP.
2. I parametri si trovano a un offset positivo rispetto allo stesso registro.

In questo contesto, per "**offset**" si intende la differenza rispetto al valore di riferimento, generalmente rappresentato dal registro EBP. Dalla sezione evidenziata in verde, si può dedurre che i primi **23** dati siano identificabili come **variabili**, poiché presentano un offset negativo rispetto al registro EBP.

.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 -388h
.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

4. PARAMENTRI DELLA FUNZIONE ALLA LOCAZIONE DI MEMORIA 0X10001656

I parametri della funzione si trovano a un offset positivo rispetto al registro EBP, quindi solo l'ultimo dato rappresentato si può identificare come **parametro**.

→

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

4. COMPORTAMENTO DEI 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:

10016008	RegCloseKey	ADVAPI32	
1001600C	RegQueryValueExA	ADVAPI32	
10016010	RegOpenKeyExA	ADVAPI32	
10016014	CreateProcessAsUserA	ADVAPI32	
10016018	RegSetValueExA	ADVAPI32	
1001601C	RegDeleteValueA		
10016020	RegEnumKeyA		
10016024	RegOpenKeyA		
10016028	SetTokenInformation		
1001602C	DuplicateTokenEx		
10016030	RegEnumValueA	ADVAPI32	
10016034	AdjustTokenPrivileges	ADVAPI32	
10016038	RegCreateKeyA	ADVAPI32	
1001603C	RegDeleteKeyA	ADVAPI32	
10016374	PostThreadMessageA		USER32
10016378	GetMessageA		USER32
1001637C	RedrawWindow		USER32
10016380	DrawTextA		USER32
10016384	GetSystemMetrics		USER32

4. COMPORTAMENTO DEI MALWARE

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

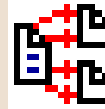



- L'import delle librerie per maneggiare i file

	100160FC	GlobalMemoryStatus	KERNEL32
	10016100	GetComputerNameA	KERNEL32
	10016104	CopyFileA	KERNEL32
	10016108	MoveFileExA	KERNEL32
	1001610C	GetModuleFileNameA	KERNEL32

4. COMPORTAMENTO DEI MALWARE

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

- L'import della libreria socket:

	100163F0	115	WSAStartup	WS2_32
	100163F4	3	closesocket	WS2_32
	100163F8	23	socket	WS2_32
	100163FC	111	WSAGetLastError	WS2_32

4. COMPORTAMENTO DEI MALWARE

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

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

100163C8	11	inet_addr	WS2_32
100163...	52	gethostbyname	WS2_32
100163...	12	inet_ntoa	WS2_32
100163...	16	recv	WS2_32
100163...	19	send	WS2_32
100163...	4	connect	WS2_32

4. COMPORTAMENTO DEI MALWARE

L'analisi degli import di tutte queste librerie suggerisce che il malware sia progettato per stabilire una connessione con l'esterno, consentendo a un utente malevolo di eseguire operazioni sul sistema compromesso. Questo tipo di comportamento è tipico di una backdoor, un tipo di malware che fornisce accesso remoto non autorizzato al sistema infetto.

Una backdoor consente a un attaccante di operare sulla macchina target senza dover passare attraverso i normali meccanismi di autenticazione, come l'inserimento di credenziali. Una volta stabilita la connessione, l'attaccante può eseguire una vasta gamma di operazioni dannose, che possono includere l'esfiltrazione di dati, l'installazione di ulteriori componenti malevoli, la manipolazione dei file di sistema, o persino il controllo completo del dispositivo compromesso.

L'utilizzo di queste librerie specifiche, necessarie per gestire la comunicazione di rete e interagire con il sistema operativo, rafforza l'ipotesi che il malware sia stato progettato per garantire un accesso persistente e segreto al sistema infetto, consentendo all'attaccante di mantenere il controllo nel tempo e di eludere le misure di sicurezza implementate sulla macchina bersaglio.