

## ANALISI STATICA AVANZATA CON IDA PRO

### Analisi del malware *Malware\_U3\_W3\_L2.dll*

#### Tasks:

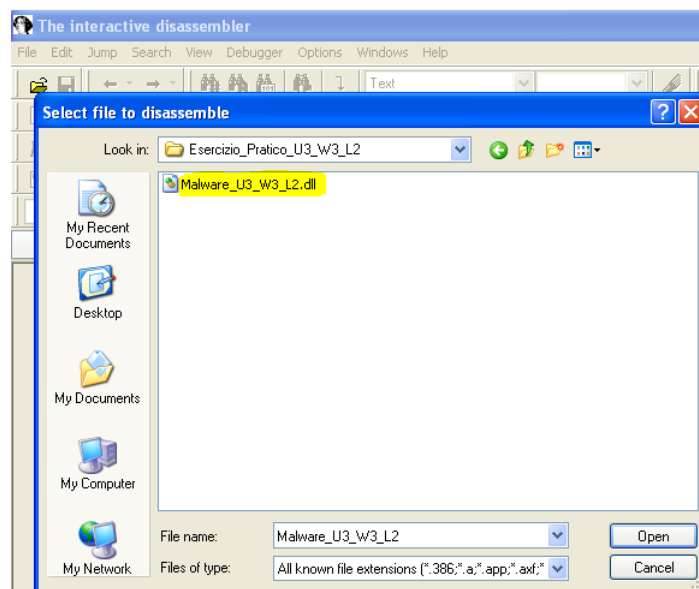
1. Individuazione dell'indirizzo della funzione **DLLMain**
2. Individuazione dell'indirizzo di import della funzione **gethostbyname**
3. Quantificazione delle **variabili locali** e dei **parametri** della funzione alla locazione di memoria 0x10001656
4. Considerazioni macro-livello circa il comportamento del malware

#### 1. Individuazione dell'indirizzo della funzione **DLLMain**

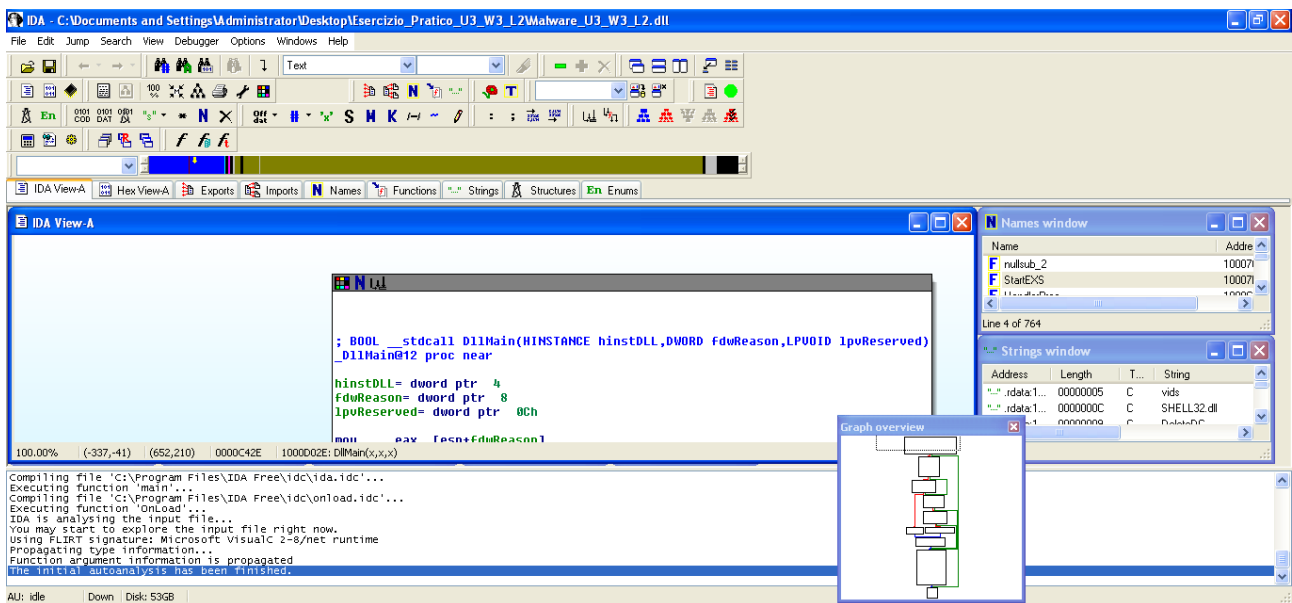
Le operazioni oggetto di focus odierno rientrano nel settore della malware analysis denominato **Analisi Statica Avanzata**: andremo ad esaminare un malware a partire dalle istruzioni in Assembly che lo compongono.

Il malware oggetto di analisi è **Malware\_U3\_W3\_L2.dll**, che rintracciamo nella VM Malware\_Analysis con sistema operativo Windows XP SP3. Per effettuare l'analisi ci serviremo del tool **IDA Pro** – un **disassembler** (= strumento che si occupa della traduzione completa del linguaggio macchina di un eseguibile in linguaggio Assembly) che riunisce al suo interno una serie di feature che lo rendono ampiamente consultabile.

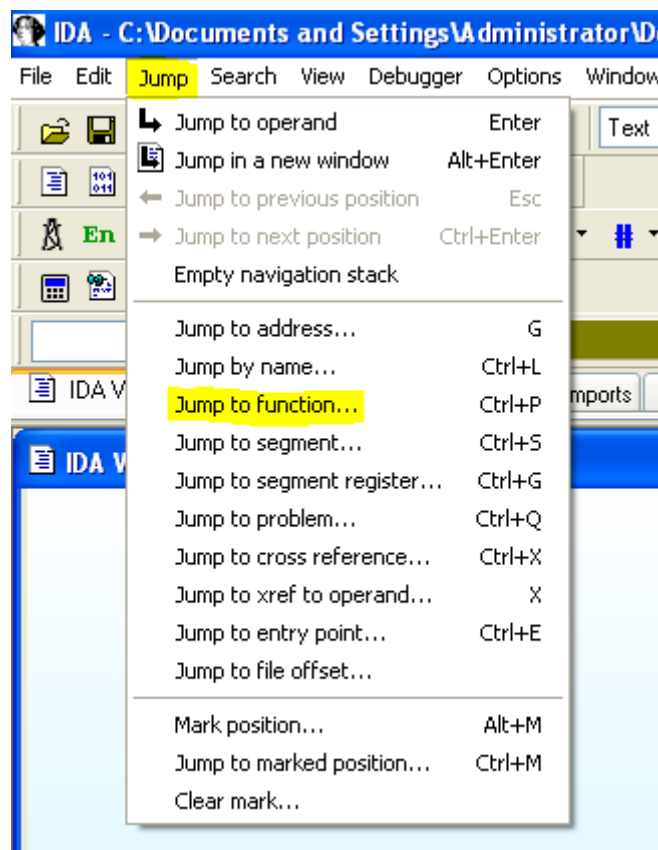
Avviamo dunque il tool menzionato e scegliamo di aprire il file di test appena menzionato:

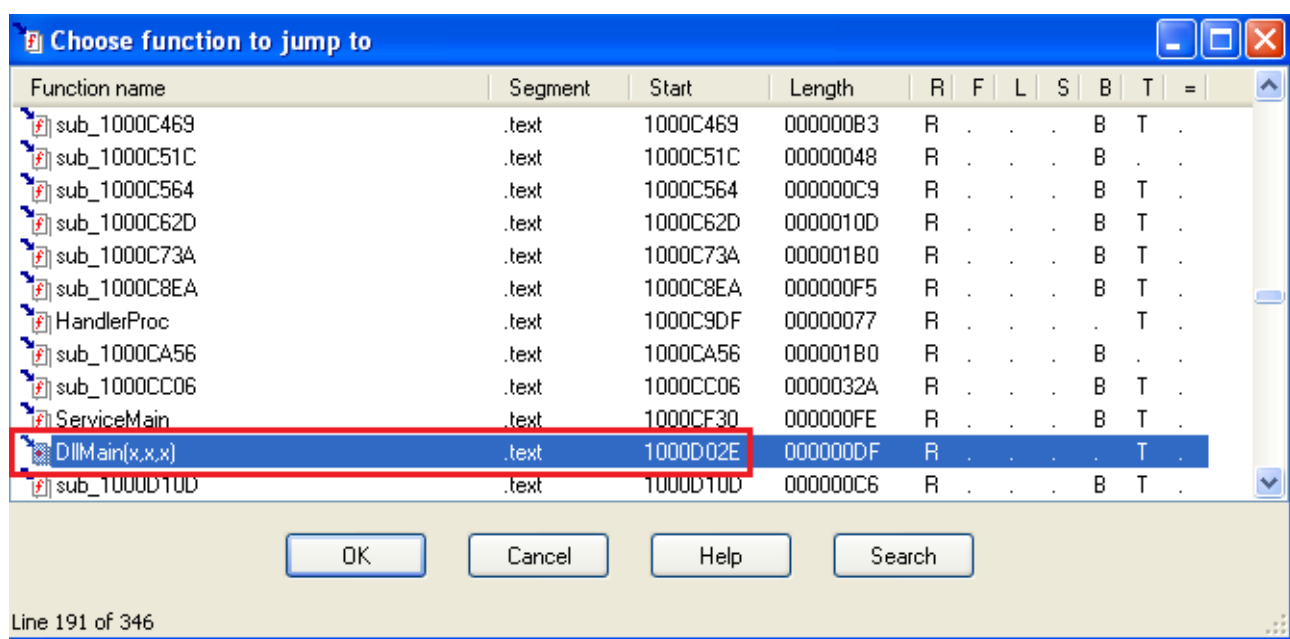
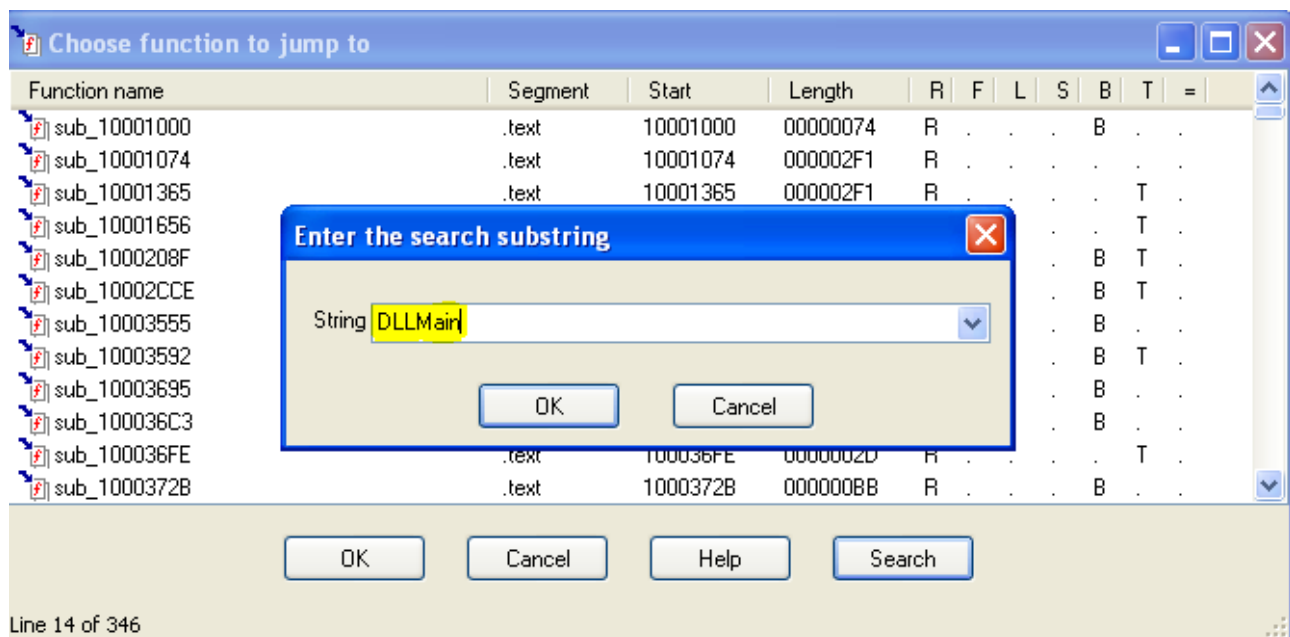


Il file appena aperto ci restituisce la seguente schermata, comprensiva di tutte le sezioni del tool a nostra disposizione per l'analisi del file in oggetto (exports, imports, functions, strings, etc.):



Adesso individuiamo l'indirizzo della funzione **DLLMain**. Per farlo, apriamo il menù **Jump** → **Jump to function...** e cerchiamo la funzione di nostro interesse tramite la funzionalità **Search**:

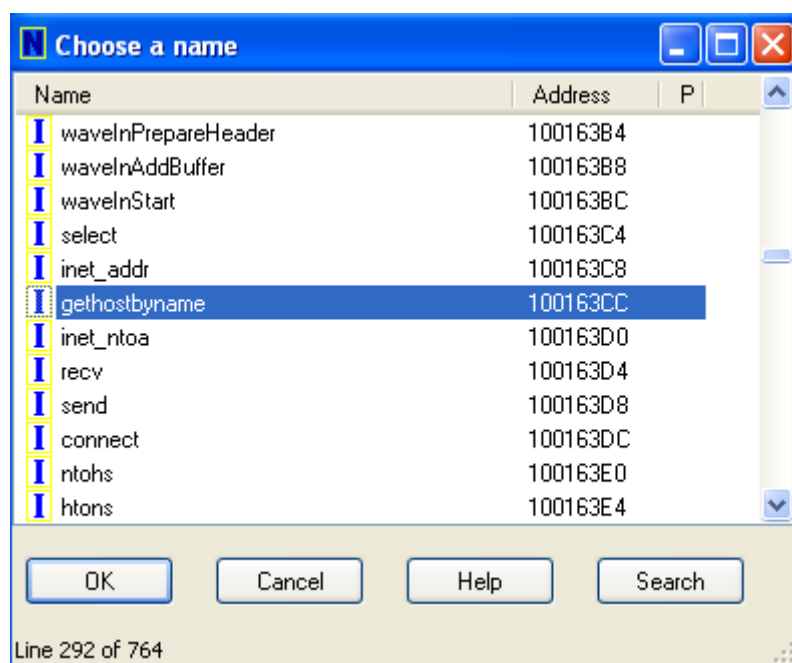
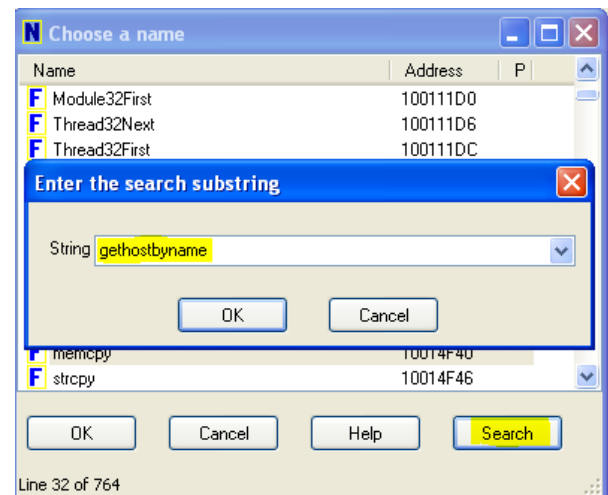
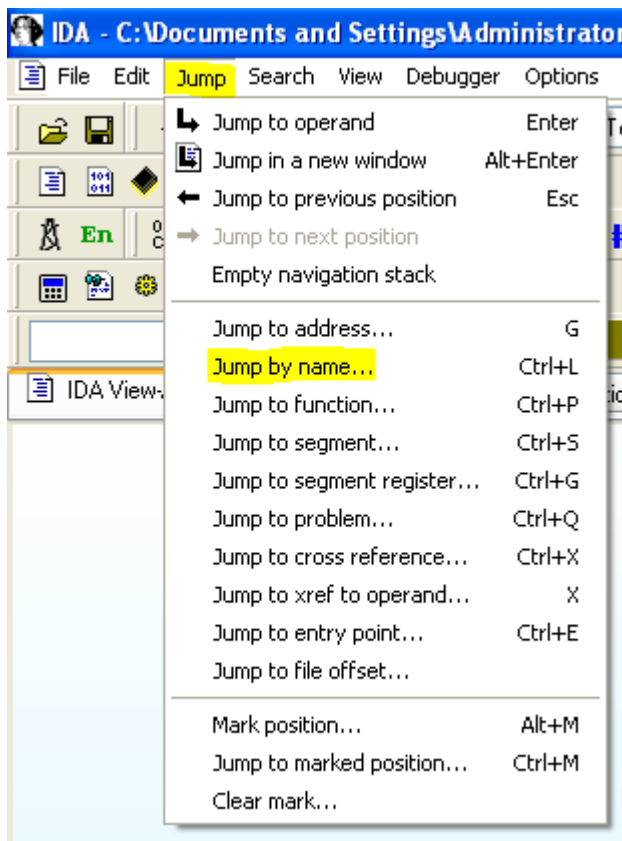




Abbiamo identificato la funzione DLLMain; l'indirizzo di memoria ad essa associato è **1000D02E**.

## 2. Individuazione dell'indirizzo di import della funzione **gethostbyname**

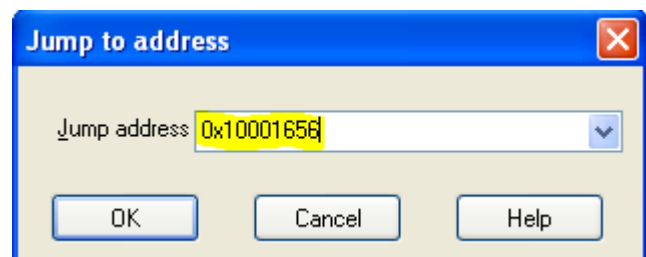
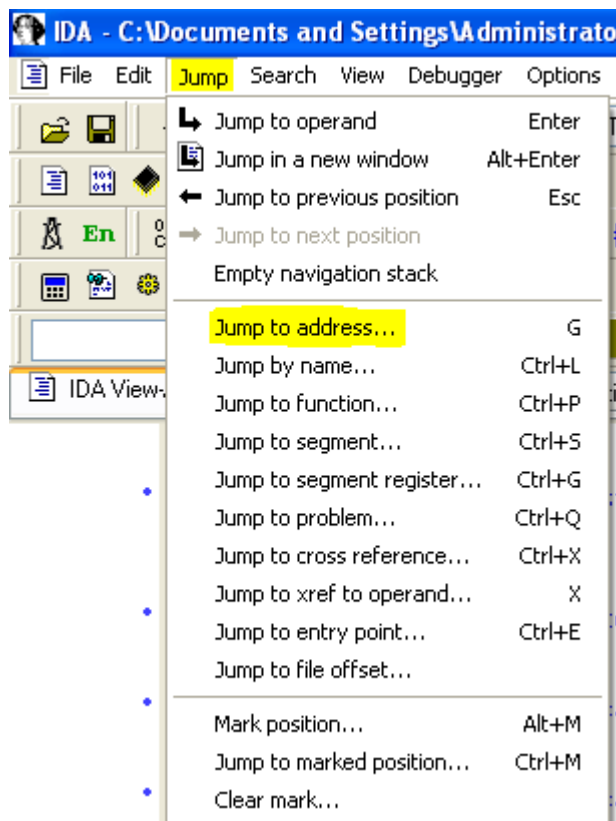
Adesso individuiamo la funzione **gethostbyname**, presente tra le funzioni importate dall'eseguibile (rintracciabile, quindi, anche all'interno del tab "imports"). Scegliamo di servirci della funzionalità *jump by name* e ricercare il nome della funzione cliccando su "Search":



Abbiamo individuato la funzione ricercata, presente all'indirizzo di memoria **1001063CC**.

### 3. Quantificazione delle **variabili locali** e dei **parametri** della funzione alla locazione di memoria 0x10001656

Adesso vogliamo analizzare la funzione presente alla locazione di memoria 0x10001656. Per farlo, utilizzeremo la funzionalità *Jump to address*, per poi inserire nella barra di ricerca l'indirizzo di nostro interesse:



```
.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 hModule = dword ptr -670h
.text:10001656 timeout = timeval ptr -66Ch
.text:10001656 name = sockaddr ptr -664h
.text:10001656 var_654 = word ptr -654h
.text:10001656 in = in_addr ptr -650h
.text:10001656 Parameter = byte ptr -644h
.text:10001656 CommandLine = byte ptr -63Fh
.text:10001656 Data = byte ptr -638h
.text:10001656 var_544 = dword ptr -544h
.text:10001656 var_50C = dword ptr -50Ch
.text:10001656 var_500 = dword ptr -500h
.text:10001656 var_4FC = dword ptr -4FCh
.text:10001656 readfds = fd_set ptr -4BCh
.text:10001656 phkResult = HKEY__ 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
```

variabili locali

parametro

Come si può notare, il risultato della ricerca ci restituisce la funzione di tipo subroutine **sub\_10001656**, di cui abbiamo evidenziato le caratteristiche di nostro interesse, ossia

- 20 variabili locali
- 1 parametro

A questo proposito, è importante sapere che il tool IDA differenzia variabili e parametri utilizzando come riferimento l'**offset** (= differenza rispetto ad un valore di riferimento) rispetto al **puntatore EBP**. In particolare,

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

#### 4. Considerazioni macro-livello circa il comportamento del malware

Adesso vediamo di ipotizzare la finalità del malware. Per prima cosa, vogliamo capire se il malware ha lo scopo di ottenere la **persistenza** dentro il sistema della macchina vittima. Per ottenerla, il malware aggiunge se stesso alle entry dei programmi che devono essere eseguiti all'avvio del PC, in modo tale da essere **eseguito in maniera automatica** e permanente senza alcun intervento da parte dell'utente. Per far ciò, il malware richiede l'accesso e la modifica ad una chiave di registro tramite due chiamate di funzione principali: RegOpenKeyEx per accedere alla key e RegSetValueEx per modificarla.

La funzione **RegOpenKeyEx** permette di aprire una chiave di registro al fine di modificarla; accetta tra i parametri la chiave da aprire. Con questa funzione il malware accede alla chiave di registro prima di modificarne il valore.

La funzione **RegSetValueEx** permette di aggiungere un nuovo valore all'interno del registro e di configurare i rispettivi dati. Accetta come parametri la chiave, la sottochiave ed il dato da inserire. Questa funzione viene utilizzata dal malware per **modificare il valore del registro** ed aggiungere una nuova entry in modo tale da ottenere la persistenza all'avvio del sistema operativo.

Una delle chiavi di registro che vengono utilizzate dai malware per ottenere persistenza su un sistema operativo Windows è **Software\\Microsoft\\Windows\\CurrentVersion\\Run**, come si può vedere nella figura sottostante:

.text:10005659	push	0F003Fh	; samDesired
.text:1000565E	push	esi	; u1Options
.text:1000565F	push	offset aSoftwareMicros	; "SOFTWARE\\Microsoft\\Windows\\CurrentVersion"...
.text:10005664	push	80000002h	; hKey
.text:10005669	call	ds:RegOpenKeyExA	
.text:1000566F	test	eax, eax	
.text:10005671	jnz	short loc_1000568F	
.text:10005673	lea	eax, [ebp+Data]	
.text:10005676	push	4	; cbData
.text:10005678	push	eax	; lpData
.text:10005679	push	4	; dwType
.text:1000567B	push	esi	; Reserved
.text:1000567C	push	[ebp+lpValueName]	; lpValueName
.text:1000567F	push	[ebp+hKey]	; hKey
.text:10005682	call	ds:RegSetValueExA	

accesso alla chiave di registro

chiave di registro utilizzata per ottenere la persistenza

modifica del valore del registro

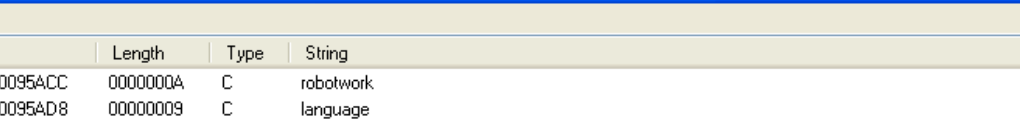
Come si può notare, la ricerca della richiesta da parte del malware di ottenere la persistenza ha avuto esito positivo. Con questa premessa, possiamo ipotizzare che il malware sia una **backdoor** che, come tale, abbia bisogno di essere perennemente in esecuzione sulla macchina target. Proviamo dunque a cercare questa parola chiave all'interno della sezione di IDA dedicata alle stringhe, utilizzando la funzionalità "Search":

The screenshot shows the 'Strings window' in Immunity Debugger. The window has a blue title bar and a menu bar with 'Edit' and 'Search'. Below the menu bar is a table with four columns: 'Address', 'Length', 'Type', and 'String'. The table lists various strings extracted from memory. The string '\r\n\r\n[BackDoor Server Update Setup]\r\n' is highlighted in blue. The status bar at the bottom indicates 'Line 483 of 746'.

Address	Length	Type	String
["..."] xdoors_d:10093D04	00000025	C	\r\n(3) Move %s To %s Successfully
["..."] xdoors_d:10093D2C	00000006	C	.ubak
["..."] xdoors_d:10093D34	0000001C	C	\r\n(2) Get DLL FileName %s'
["..."] xdoors_d:10093D50	00000023	C	\r\n(1) Enter Current Directory %s'
["..."] xdoors_d:10093D74	00000067	C	\r\n\r\n[BackDoor Server Update Setup]\r\n
["..."] xdoors_d:10093DDC	00000006	C	-warn
["..."] xdoors_d:10093DE4	00000006	C	-erro
["..."] xdoors_d:10093DEC	00000006	C	-stop
["..."] xdoors_d:10093DF4	0000000A	C	-shutdown
["..."] xdoors_d:10093E00	00000005	C	-+6
["..."] xdoors_d:10093E08	00000008	C	-reboot
["..."] xdoors_d:10093E10	00000008	C	Default

```
xdoors_d:10093D74 ; char aBackdoorServer[]
xdoors_d:10093D74 aBackdoorServer db 0Dh,0Ah ; DATA XREF: sub_100042DB+B5↑o
xdoors_d:10093D74 db 0Dh,0Ah
xdoors_d:10093D74 db '*****',0Dh,0Ah
xdoors_d:10093D74 db '[BackDoor Server Update Setup]',0Dh,0Ah
xdoors_d:10093D74 db '*****',0Dh,0Ah
xdoors_d:10093D74 db 0Dh,0Ah,0
xdoors_d:10093DDB align 4
```

La ricerca ha esito positivo. Possiamo supporre, dunque, l'esistenza di una **remote shell** che il malware ha lo scopo di mantenere in esecuzione. Proviamo ad effettuare una ricerca tra le stringhe presenti all'interno dell'eseguibile:



Address	Length	Type	String
"" xdoors_d:10095ACC	0000000A	C	robotwork
"" xdoors_d:10095AD8	00000009	C	language
"" xdoors_d:10095AE4	00000007	C	uptime
"" xdoors_d:10095AEC	00000005	C	idle
"" xdoors_d:10095AF4	0000000F	C	\r\n\r\n0x%02x\r\n\r\n
"" xdoors_d:10095B04	00000008	C	enmagic
"" xdoors_d:10095B10	00000005	C	exit
"" xdoors_d:10095B18	00000005	C	quit
"" xdoors_d:10095B20	00000011	C	\\command.exe /c
"" xdoors_d:10095B34	0000000D	C	\\cmd.exe /c
"" xdoors_d:10095B44	00000118	C	Hi,Master [%d/%d/%d %d:%d:%d]\r\nWelCome Back...Are You Enjoying Today?\r\n\r\nMachine UpTime [%:2d

Line 746 of 746

```

* xdoors_d:10095B20 ; char aCommand_exeC[]
xdoors_d:10095B20 aCommand_exeC db '\command.exe /c ',0 ; DATA XREF: sub_1000FF58:loc_100101D7↑o
* xdoors_d:10095B31 align 4
xdoors_d:10095B34 aCmd_exeC db '\cmd.exe /c ',0 ; DATA XREF: sub_1000FF58+278↑o
* xdoors_d:10095B41 align 4
xdoors_d:10095B44 ; char aHiMasterDDDDDD[]
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
xdoors_d:10095B44 ; DATA XREF: sub_1000FF58+145↑o
xdoors_d:10095B44 db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
xdoors_d:10095B44 db 0Dh,0Ah
xdoors_d:10095B44 db 'Machine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Secon'
xdoors_d:10095B44 db 'ds]',0Dh,0Ah
xdoors_d:10095B44 db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44 db 'nds]',0Dh,0Ah
xdoors_d:10095B44 db 0Dh,0Ah
xdoors_d:10095B44 db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095B44 db 0Dh,0Ah,0

```

Anche in questo caso la ricerca ha esito positivo.

Infine, vogliamo confrontare le conclusioni appena tratte con le verifiche effettuate dal database online VirusTotal, all'interno del quale recheremo l'hash MD5 del malware appena esaminato (1A9FD80174AAFECD9A52FD908CB82637).



Analyse suspicious files, domains, IPs and URLs to detect malware and other breaches, automatically share them with the security community.

FILE

URL

SEARCH



1A9FD80174AAFECD9A52FD908CB82637

By submitting data above, you are agreeing to our [Terms of Service](#) and [Privacy Policy](#), and to the sharing of your Sample submission with the security community. Please do not submit any personal information; VirusTotal is not responsible for the contents of your submission. [Learn more.](#)



50 / 67

Community Score

50 security vendors and no sandboxes flagged this file as malicious

eb1079bdd96bc9cc19c38b76342113a09666aad47518ff1a7536eebff8aadb4a

X-doorc

pedll

corrupt

amaddillo

overlay

130.94 KB

Size

2023-01-17 09:14:27 UTC

1 day ago

DLL

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 19

Security vendors' analysis

AhnLab-V3	Backdoor.Win32.Agent.R9408	Alibaba	Backdoor.Win32/Idcaf.9f3a5556
ALYac	Backdoor.XI/W	Antiy-AVL	Trojan.Backdoor.J/Win32.Agent
Arcabit	Backdoor.XI/W	Avast	Win32:Agent-OLH [Trj]
AVG	Win32:Agent-OLH [Trj]	Avira (no cloud)	BDS/Agent.twe.134160
BitDefender	Backdoor.XI/W	ClamAV	Win.Trojan.Idicaf-0937585-0
Cynet	Malicious (score: 100)	Cyren	W32/Backdoor.LTKC-2937
DrWeb	BackDoor.Siggen.47995	Elastic	Malicious (high Confidence)
Emsisoft	Backdoor.XI/W (B)	eScan	Backdoor.XI/W

Come si vede, il malware ha riportato uno score di 50/67 e molti Security vendors lo identificano come una backdoor, in accordo con le ipotesi formulate.

Fonte:

<https://www.virustotal.com/gui/file/eb1079bdd96bc9cc19c38b76342113a09666aad47518ff1a7536eebff8aadb4a/detection>