

MALWARE ANALYSIS - ANALISI DINAMICA AVANZATA CON OLLYDBG

Analisi del malware *U3_W3_L3.exe*

Tasks:

1. Individuazione del valore del parametro "**CommandLine**" che viene passato sullo stack nell'ambito della chiamata alla funzione *CreateProcess* all'indirizzo di memoria 0040106E
2. Inserimento di un **software breakpoint** all'indirizzo 004015A3 ed identificazione del valore del registro EDX
3. Esecuzione di uno **step-into** e conseguente individuazione del valore aggiornato del registro EDX; identificazione del tipo di istruzione eseguita
4. Inserimento di un **software breakpoint** all'indirizzo di memoria 004015AF ed identificazione del valore del registro ECX
5. Esecuzione di uno **step-into** conseguente individuazione del valore aggiornato del registro ECX; identificazione del tipo di istruzione eseguita
6. Identificazione del comportamento del malware

-
1. Individuazione del valore del parametro "**CommandLine**" che viene passato sullo stack nell'ambito della chiamata alla funzione *CreateProcess* all'indirizzo di memoria 0040106E

L'attività odierna consiste nell'**analisi dinamica avanzata** del malware *U3_W3_L3.exe*.

L'analisi dinamica presuppone l'esecuzione del malware per studiare gli impatti che ha sul sistema in tempo reale, ossia mentre è in esecuzione. L'analisi dinamica basica ci aiuta a capire i flussi di rete che genera un malware, la creazione di nuovi processi e thread oppure la modifica di processi esistenti e come esso impatta il registro di windows, ad esempio ottenendo la persistenza.

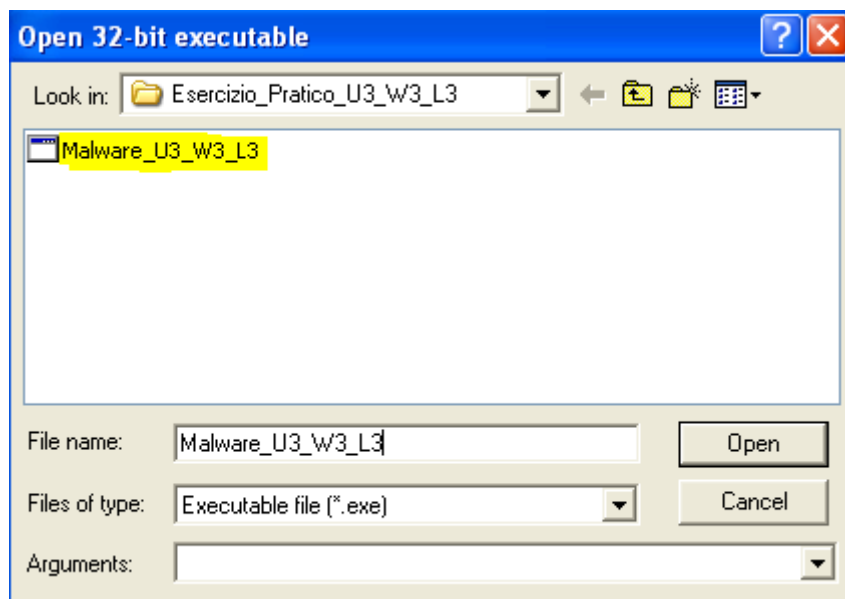
Per capire lo stato della memoria, dei registri, delle variabili e dei parametri durante l'esecuzione del malware, tuttavia, abbiamo bisogno dell'analisi dinamica avanzata.

Per procedere con l'analisi, ci serviremo del debugger **OllyDBG**.

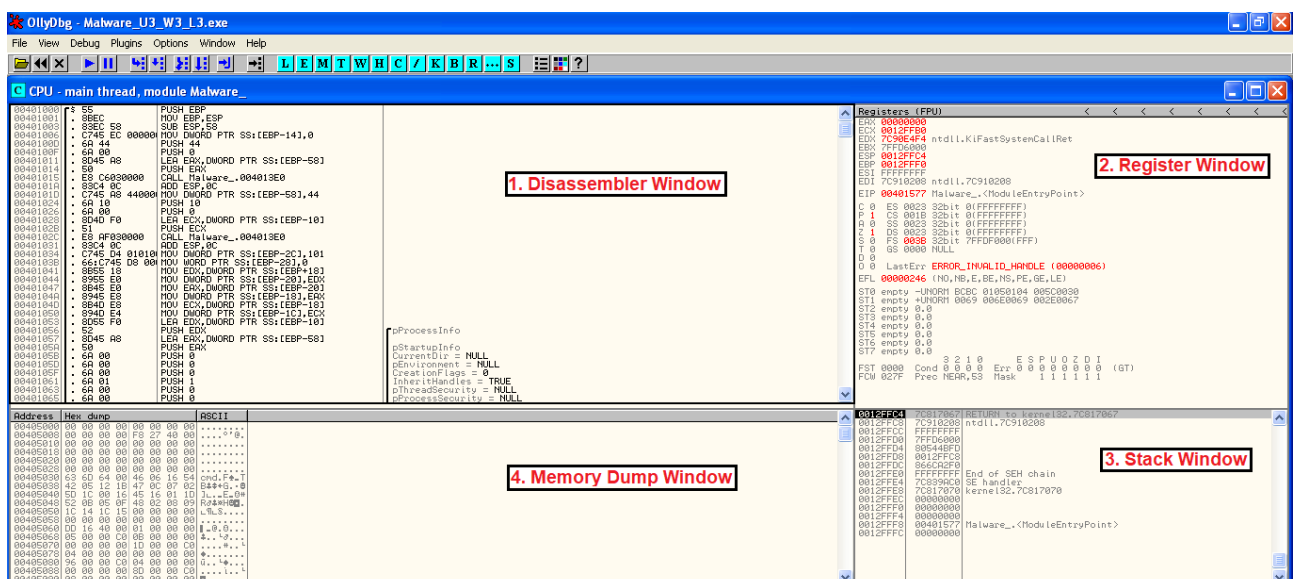
Un debugger è un software che permette di analizzare l'esecuzione di un eseguibile, evidenziando nel dettaglio le modifiche che esso subisce in tempo reale, ad esempio

- Come cambiano gli **indirizzi di memoria**
- Come cambia il contenuto dei **registri della CPU**
- Come si modificano i **parametri di una funzione**
- Come si modificano le **variabili di una funzione**

Per prima cosa, dunque, apriamo l'eseguibile di nostro interesse all'interno del programma:



All'apertura del file, la schermata principale è la seguente:



```

Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E4F4 ntdll.KiFastSystemCallRet
EBX 7FFD6000
ESP 0012FFC4
EBP 0012FFD0
ESI FFFFFFFF
EDI 7C910208 ntdll.7C910208
EIP 00401577 Malware_.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_INVALID_HANDLE (00000006)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BCBC 01050104 005C0030
ST1 empty +UNORM 0069 006E0069 002E0067
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1

```

Come si può notare, possiamo distinguere 4 sezioni:

1. **Disassembler Window** (simile alla schermata testuale di IDA Pro) – contiene le istruzioni eseguite dalla CPU con aggiunta di commenti inseriti da OllyDBG in base al suo DB
2. **Register Window** – riporta lo stato dei registri e del loro valore al momento del breakpoint. Appena caricato il malware, il registro EIP (puntatore all'istruzione da eseguire) punta all'entry point (= punto di entrata per il codice, ossia la prima istruzione che viene eseguita dalla CPU). Si tratta di un pannello fondamentale per capire lo stato attuale del programma. Quando un registro cambia valore viene evidenziato in rosso
3. **Stack Window** – schermata dello stack. Mostra lo stato attuale dello stack in memoria per il programma/funzione che è attualmente in esecuzione. OllyDBG aggiunge dei commenti piuttosto significativi in questa finestra per supportare la comprensione dell'eseguibile
4. **Memory Dump Window** – schermata della memoria. In questa sezione è incluso il contenuto degli indirizzi di memoria del programma in esecuzione

Adesso siamo pronti per procedere con la nostra analisi. Per prima cosa, ci spostiamo all'indirizzo di memoria **0040106E** ed esaminiamo il parametro "CommandLine" passato alla funzione *CreateProcess*. Il valore del parametro è "cmd", ossia il prompt dei comandi di Windows:

```

CPU - main thread, module Malware_
00401031 83C4 0C ADD ESP,0C
00401034 C745 D4 010101 MOV DWORD PTR SS:[EBP-2C],101
00401038 66:C745 D8 0000 MOV WORD PTR SS:[EBP-20],0
00401041 8B55 18 MOV EDX,DWORD PTR SS:[EBP+18]
00401044 8B55 E0 MOV EDI,DWORD PTR SS:[EBP-20],EDI
00401047 8B45 E0 MOV EAX,DWORD PTR SS:[EBP-20]
0040104A 094C E0 MOV DWORD PTR SS:[EDP+10],EAX
0040104D 8B4D E8 MOV ECX,DWORD PTR SS:[EBP-18]
00401050 894D E4 MOV DWORD PTR SS:[EBP-1C],ECX
00401053 8D55 F0 LEA EDI,DWORD PTR SS:[EBP-10]
00401056 52 PUSH EDI
00401057 8D45 A8 LEA EAX,DWORD PTR SS:[EBP-58]
0040105A 50 PUSH EAX
0040105B 6A 00 PUSH 0
0040105D 6A 00 PUSH 0
0040105F 6A 00 PUSH 0
00401061 6A 01 PUSH 1
00401063 6A 00 PUSH 0
00401065 6A 00 PUSH 0
00401067 68 30504000 PUSH Malware_.00405000
0040106C 6A 00 PUSH 0
0040106E FF15 04404000 CALL DWORD PTR DS:[&KERNEL32.CreateProcessA]
00401074 8945 EC MOV DWORD PTR SS:[EBP-14],EAX
00401077 6A FF PUSH -1
00401079 8B4D F0 MOV ECX,DWORD PTR SS:[EBP-10]
0040107C 51 PUSH ECX
0040107D FF15 04404000 CALL DWORD PTR DS:[&KERNEL32.WaitForSingleObject]
00401083 33C0 XOR EAX,EAX
00401085 8BE5 MOV ESP,EBP
00401087 5D POP EBP
00401088 C3 RETN
00401089 55 PUSH EBP
0040109A 8BEC MOV EBP,ESP
0040109C 81EC 00010000 SUB ESP,100
0040109E 57 PUSH EDI

```

```

pProcessInfo
pStartupInfo
CurrentDir = NULL
pEnvironment = NULL
CreationFlags = 0
InheritHandles = TRUE
pThreadSecurity = NULL
pProcessSecurity = NULL
CommandLine = "cmd"
ModuleFileName = NULL
CreateProcessA

Timeout = INFINITE
hObject
WaitForSingleObject

```

2. Inserimento di un **software breakpoint** all'indirizzo **004015A3** ed identificazione del valore del registro EDX

I **breakpoint** vengono utilizzati per mettere in pausa l'esecuzione di un programma in un determinato punto ed analizzare i vari registri, le memorie, le variabili ed altro ancora.

Generalmente, all'avvio di un programma tramite debugger, il primo breakpoint viene impostato prima dell'entry point di un eseguibile. L'analista ha successivamente controllo del programma e può scegliere dove impostare i breakpoint per fermare l'esecuzione dell'eseguibile e controllare i vari parametri.

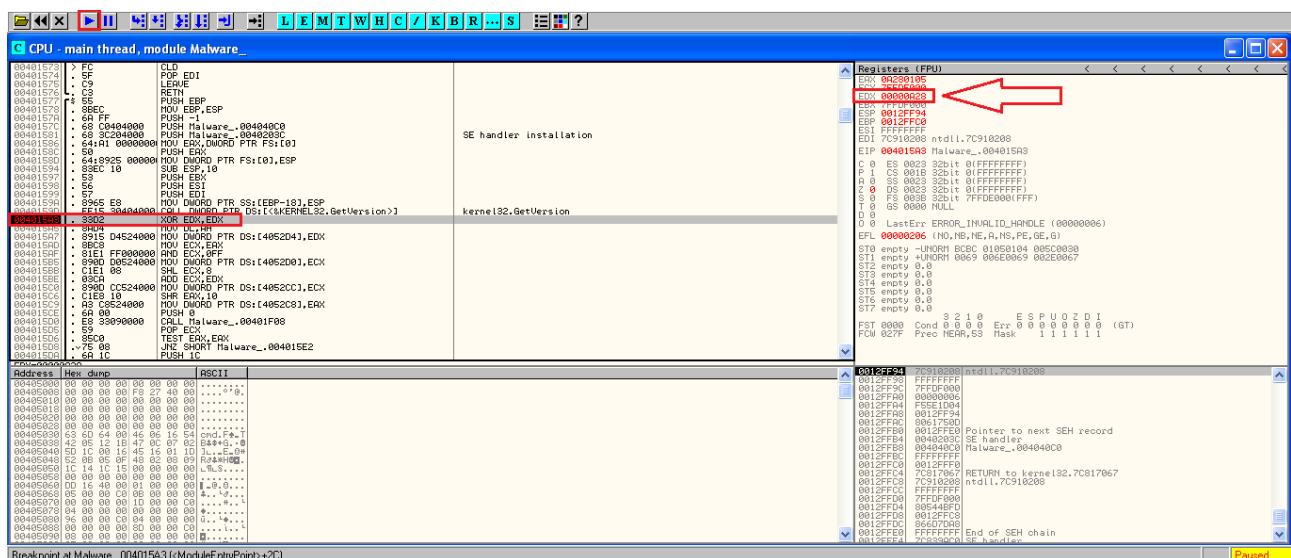
Ci sono diversi tipi di breakpoint:

Software breakpoint: permettono di fermare il programma quando una data istruzione è eseguita. Si possono configurare ad esempio su una chiamata di funzione per studiarne i dettagli, oppure all'inizio di un ciclo per capire di che si tratta.

Hardware breakpoint: permettono di fermare il programma ad un dato indirizzo di memoria, indipendentemente dal contenuto dell'istruzione che risiede a quell'indirizzo. Sono molto utili contro i malware che modificano il loro codice durante l'esecuzione

Breakpoint condizionali: fermano l'esecuzione del programma solo a patto che una determinata condizione sia vera. Ad esempio, si potrebbe impostare un breakpoint a condizione che una data funzione venga chiamata. Nei malware che generano traffico web sono spesso utilizzati i breakpoint condizionali per bloccare l'esecuzione dell'eseguibile quando si incontrano le chiamate alle funzioni della libreria WinSock o WinINet per studiare da vicino il loro comportamento, l'URL al quale il malware sta cercando di connettersi, oppure capire che file il malware sta scaricando da internet, oppure su quale porta si mette in ascolto per accettare connessioni esterne.

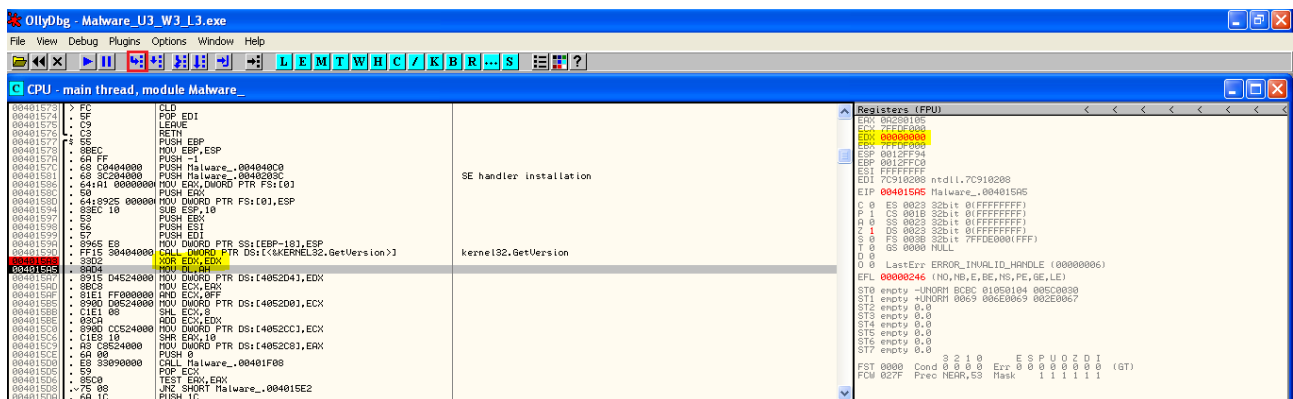
Adesso, dunque, spostiamoci all'indirizzo di memoria **004015A3** per impostare un software breakpoint; successivamente clicchiamo sul tasto **"play"**.



Come si nota, il valore del registro EDX in questo momento è **00000A28**.

3. Esecuzione di uno **step-into** e conseguente individuazione del valore aggiornato del registro EDX; identificazione del tipo di istruzione eseguita

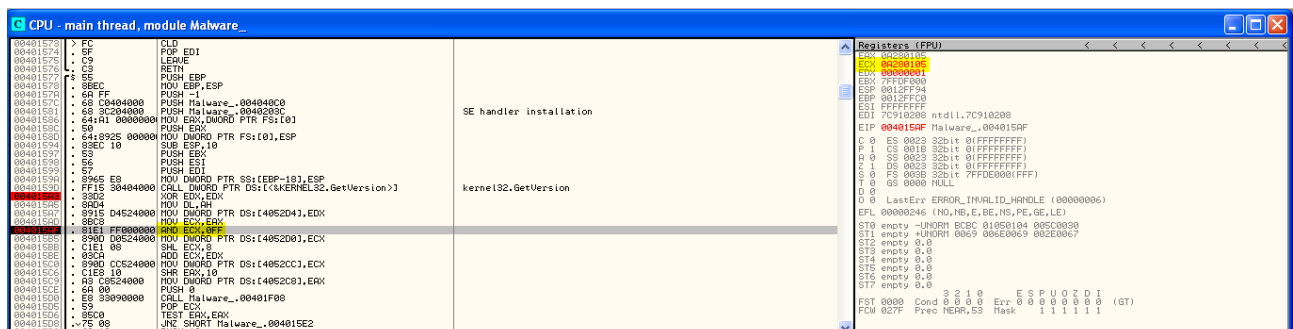
Adesso vogliamo eseguire uno **step-into**: si tratta di una tecnica di debugging che a fronte di una chiamata di funzione ci permette di **entrare nel codice della funzione**, ossia dove essa è implementata. È utile soprattutto quando vogliamo analizzare il contenuto e l'implementazione di una data funzione custom (ossia non di una funzione standard di libreria).



La tecnica di step-into ci ha permesso di entrare dentro l'istruzione **XOR EDX, EDX** che inizializza a zero la variabile EDX. Ne consegue che, dopo l'esecuzione dell'istruzione, il registro EDX abbia valore zero.

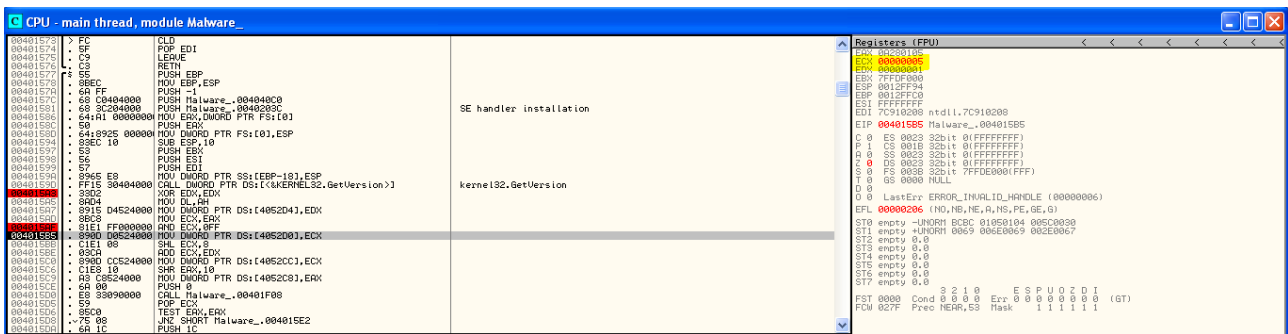
4. Inserimento di un **software breakpoint** all'indirizzo di memoria 004015AF ed identificazione del valore del registro ECX

Configuriamo adesso il secondo software breakpoint all'indirizzo di memoria **004015AF** e clicchiamo su play; il valore del registro ECX è **0A280105**.



5. Esecuzione di uno **step-into** conseguente individuazione del valore aggiornato del registro ECX; identificazione del tipo di istruzione eseguita

Analogamente al caso precedente, eseguiamo anche qui uno step-into ed osserviamo il comportamento del registro ECX:

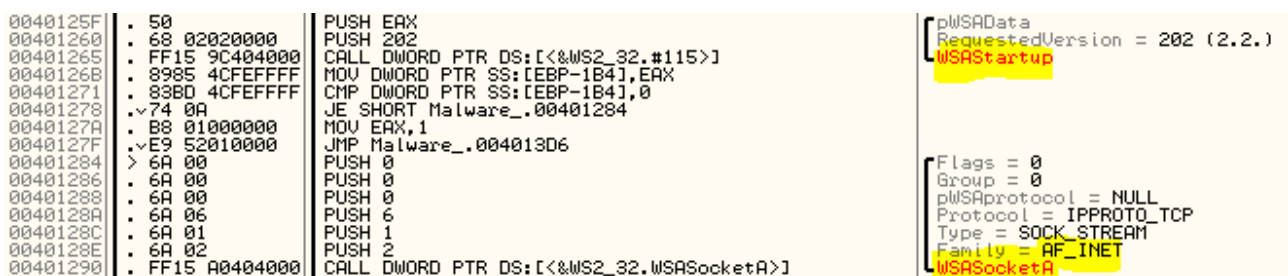


The screenshot shows a debugger window titled "CPU - main thread, module Malware_". The assembly window displays instructions from address 0040125F to 004012D0. The registers window shows the value of ECX as 00000005. The instruction at 0040125F is "AND ECX, 00000005".

Come possiamo vedere, è stata eseguita l'istruzione **AND ECX, OFF** che esegue l'AND logico tra i bit di ECX e la forma binaria del numero esadecimale OFF. Nello specifico, il formato binario del valore di ECX (che ricordiamo essere 0A280105 in formato esadecimale) di partenza è 000010100010100000000000100000101, mentre il valore di OFF in formato binario è 00000000000000000000000011111111. Eseguendo l'AND logico tra i bit uno ad uno il risultato finale è 0000000000000000000000000000101, che in formato esadecimale si traduce in 00000005. Ne consegue che **il nuovo valore di ECX è 00000005**.

6. Identificazione del comportamento del malware

In base a quanto analizzato finora, abbiamo potuto vedere come il malware oggetto di analisi si serva della chiamata alla funzione *CreateProcess* per accedere al prompt dei comandi di Windows (**cmd**): l'obiettivo è quello di creare un processo relativo all'esecuzione di tale funzionalità. Questo comportamento è tipico delle **backdoor**. Andiamo dunque alla ricerca di ulteriori indizi:



The screenshot shows a debugger window titled "CPU - main thread, module Malware_". The assembly window displays instructions from address 0040125F to 004012D0. The registers window shows the value of ECX as 00000005. The instruction at 0040125F is "AND ECX, 00000005".

Le backdoor sono facilmente riconoscibili durante l'analisi statica e dinamica in quanto possiedono delle caratteristiche piuttosto uniche. Una backdoor implementa le seguenti funzionalità:

Funzionalità di networking

Come si può notare nell'immagine, il malware effettua chiamate a due funzioni della libreria **Winsock**: **WSAStartup** (prima di qualsiasi chiamata di funzione, un malware che vuole utilizzare risorse di tipo network deve necessariamente chiamare la funzione WSAStartup) e **WSASocketA**.

Le backdoor sfruttano le APIs di windows per la gestione della rete e di networking (APIs Winsock) per mettersi in ascolto su una determinata porta del PC sul quale sono eseguite e fornire servizi amministrativi/programmi a chiunque riesca a connettersi ad essa.

Il nucleo principale di una backdoor è l'utilizzo della libreria Winsock di Windows per la creazione e gestione dei socket. In questa fase troviamo le funzioni lato server tipo **bind()** per associare il socket ad una coppia indirizzo IP + porta e **listen()** utilizzata principalmente per mettersi in ascolto ed intercettare connessioni in entrata.

Funzionalità di creazione di processi

Una volta creato il socket, la backdoor deve garantire dei servizi/processi all'utente che si connette. Questo viene fatto interagendo con il file system utilizzando le classiche librerie e funzioni, ad esempio *CreateProcess* è una delle funzioni più utilizzate. Il parametro passato alla funzione CreateProcess() specifica il processo da creare. Nel caso del prompt dei comandi sarà cmd.exe. Il command prompt viene poi successivamente sfruttato dai malintenzionati per eseguire qualsiasi operazione con privilegi amministrativi sul sistema.

Una volta creato il processo, l'esecuzione passa interamente al nuovo processo creato e la backdoor ha ultimato il suo lavoro. Essa resta in esecuzione in background solo per mantenere aperta la connessione tra la macchina locale e la macchina remota.

Nel caso del malware analizzato, possiamo anche ipotizzare che il malware svolga il ruolo di **client** che si connette ad un server controllato dall'attaccante (ad esempio per scaricare materiale da un sito o dominio), in quanto è presente la chiamata alla funzione "**connect**" che è una delle più usate lato client:

004012E7	. 51	PUSH ECX	[Socket
004012E8	. FF15 A8404000	CALL DWORD PTR DS:[<&WS2_32.#3>]	closesocket
004012EE	. FF15 AC404000	CALL DWORD PTR DS:[<&WS2_32.#116>]	WSACleanup
004012F4	. 68 30750000	PUSH 7530	Timeout = 30000. ms
004012F9	. FF15 08404000	CALL DWORD PTR DS:[<&KERNEL32.Sleep>]	Sleep
004012FF	. ^E9 48FFFFFF	JMP Malware_.0040124C	
00401304	> 8B95 44FEFFFF	MOV EDX,DWORD PTR SS:[EBP-1BC]	
0040130A	. 8B42 0C	MOV EAX,DWORD PTR DS:[EDX+C]	
0040130D	. 8B08	MOV ECX,DWORD PTR DS:[EAX]	
0040130F	. 8B11	MOV EDX,DWORD PTR DS:[ECX]	
00401311	. 8995 38FEFFFF	MOV DWORD PTR SS:[EBP-1C8],EDX	
00401317	. 68 0F270000	PUSH 270F	
0040131C	. FF15 B0404000	CALL DWORD PTR DS:[<&WS2_32.#9>]	[NetShort = 270F
00401322	. 66:8985 36FEFF	MOV WORD PTR SS:[EBP-1CA],AX	ntohs
00401329	. 66:C785 34FEFF	MOV WORD PTR SS:[EBP-1CC],2	
00401332	. 6A 10	PUSH 10	
00401334	. 8085 34FEFFFF	LEA EAX,DWORD PTR SS:[EBP-1CC]	[AddrLen = 10 (16.)
0040133A	. 50	PUSH EAX	pSockAddr
0040133B	. 8B8D FCFCFFFF	MOV ECX,DWORD PTR SS:[EBP-304]	[Socket
00401341	. 51	PUSH ECX	connect
00401342	. FF15 B4404000	CALL DWORD PTR DS:[<&WS2_32.#4>]	
00401348	. 8985 4CFEFFFF	MOV DWORD PTR SS:[EBP-1B4],EAX	
0040134E	. 83BD 4CFEFFFF	CMP DWORD PTR SS:[EBP-1B4],-1	
00401355	. ^75 23	JNZ SHORT Malware_.0040137A	
00401357	. 8095 FCFCFFFF	MOV EDX,DWORD PTR SS:[EBP-304]	
0040135D	. 52	PUSH EDX	
0040135E	. FF15 A8404000	CALL DWORD PTR DS:[<&WS2_32.#3>]	[Socket
00401364	. FF15 AC404000	CALL DWORD PTR DS:[<&WS2_32.#116>]	closesocket
0040136A	. 68 30750000	PUSH 7530	WSACleanup
0040136F	. FF15 08404000	CALL DWORD PTR DS:[<&KERNEL32.Sleep>]	Timeout = 30000. ms
00401375	. ^E9 D2FEFFFF	JMP Malware_.0040124C	Sleep
0040137A	> 8B85 FCFCFFFF	MOV EAX,DWORD PTR SS:[EBP-304]	