REPORT | MALWARE ANALYSIS

S10 - L5

Pablo Ballesteros

Indice

Introduzione

In questo documento analizzeremo un file eseguibile sospetto di nome *Malware_U3_W2_L5* nella nostra macchina virtuale Windows XP. È sempre una buona idea ogni volta che devi manipolare un file potenzialmente dannoso farlo all'interno di una macchina virtuale, come in questo caso Windows XP.

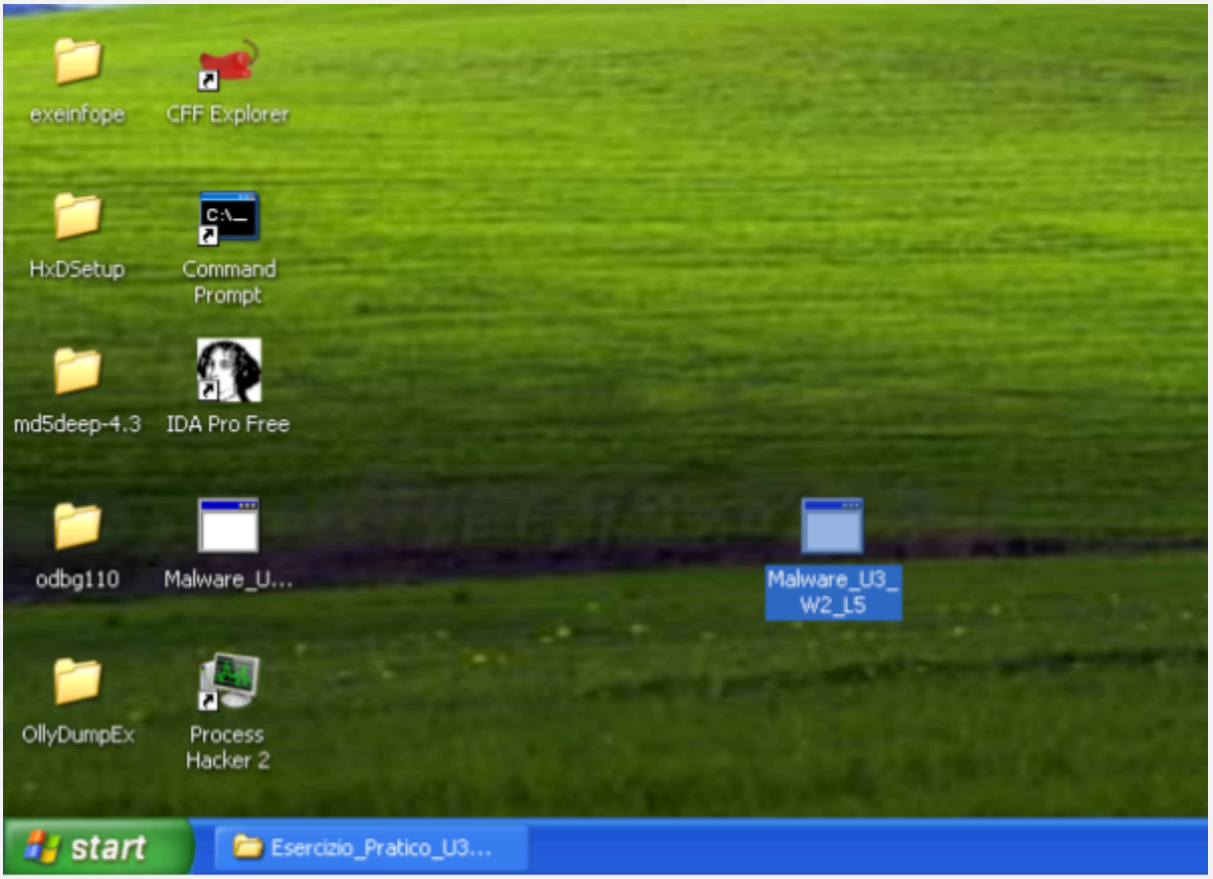


Figura 1
File da analizzare, esposto sul desktop di Windows XP.

Analisi statica basica con CFF

CFF è un insieme di strumenti freeware che consente l'analisi dei file PE a livelli avanzati. In questo caso è utile identificare le librerie importate e le sezioni che compongono il file analizzato.

La macchina virtuale Windows XP viene fornita con CFF preinstallato, quindi è sufficiente fare clic con il pulsante destro del mouse sul file e aprirlo con CFF come visto in figura 3.

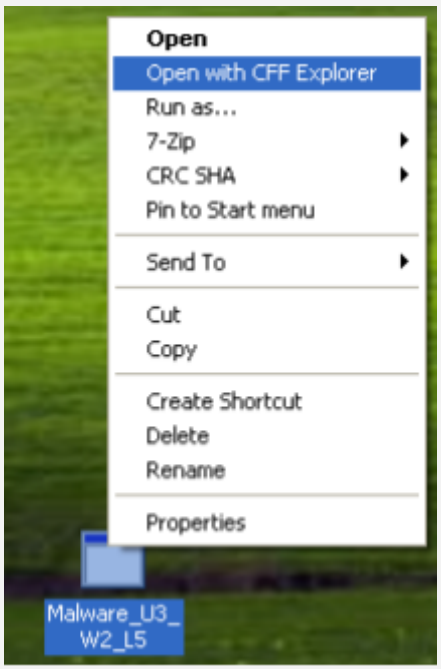


Figura 3

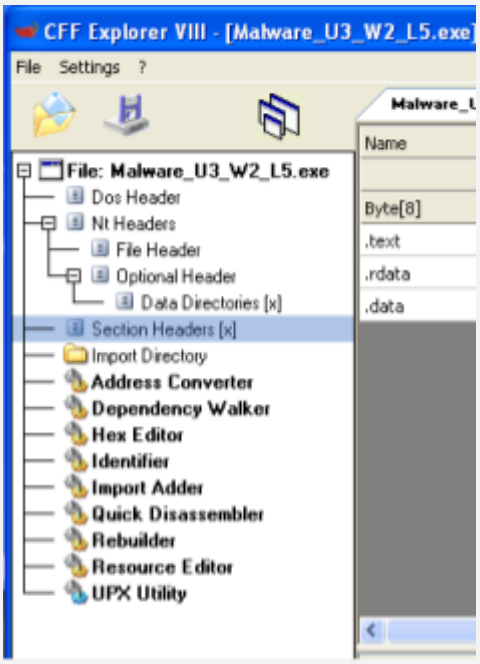


Figura 4

Una volta aperta, sul lato sinistro dell'interfaccia di CFF, possiamo vedere le 2 sezioni che ci interessano: *Section Headers* e *Import Directory*. Per prima cosa diamo un'occhiata alla directory di importazione.

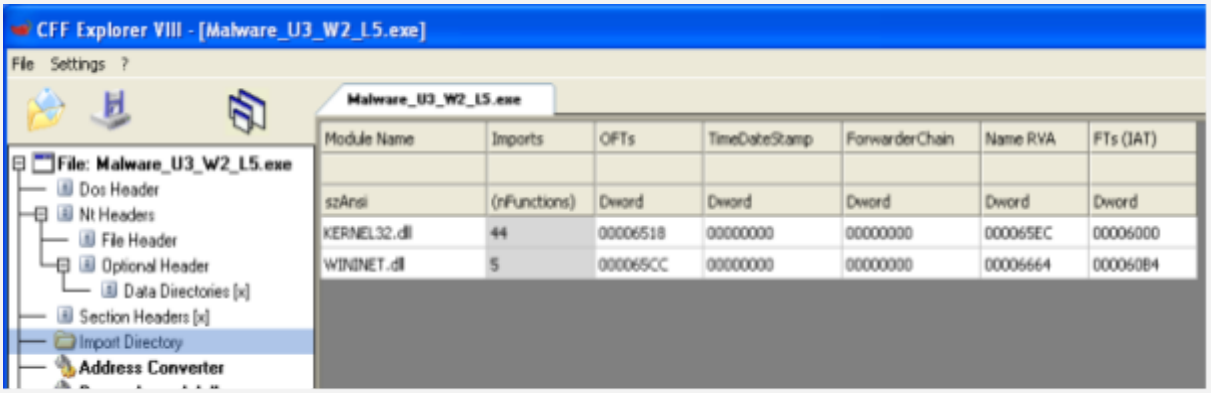


Figura 5. Import Directory

Come possiamo vedere nella figura 4, il file importa solo 2 librerie. WININET.dll con 5 funzioni e KERNEL32.dll con 44, cosa che é un po' sospetta.

Analisi delle sezioni | ASCII

Adesso facciamo riferimento alla figura 5 per identificare le sezioni del file, di cui sono 3 con molte informazioni.

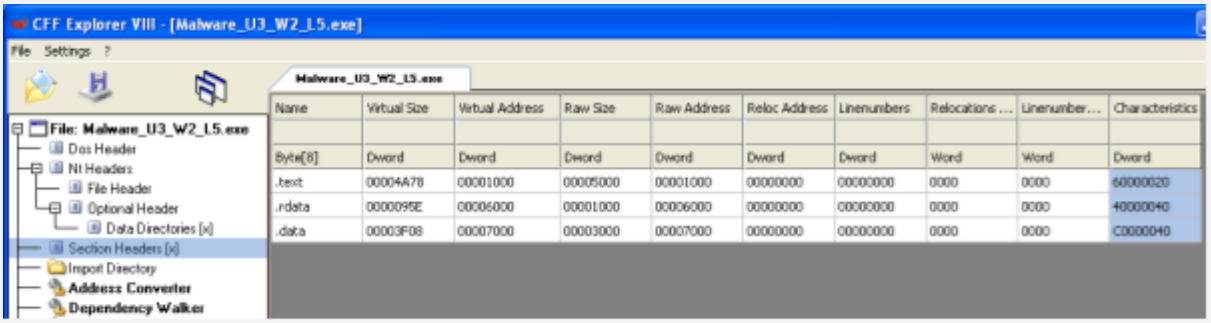


Figura 6. Section Headers

Una volta seletta una sezione, possiamo andare ad analizzare suoi contenuti in ASCII, sperando di trovare parole sospette.

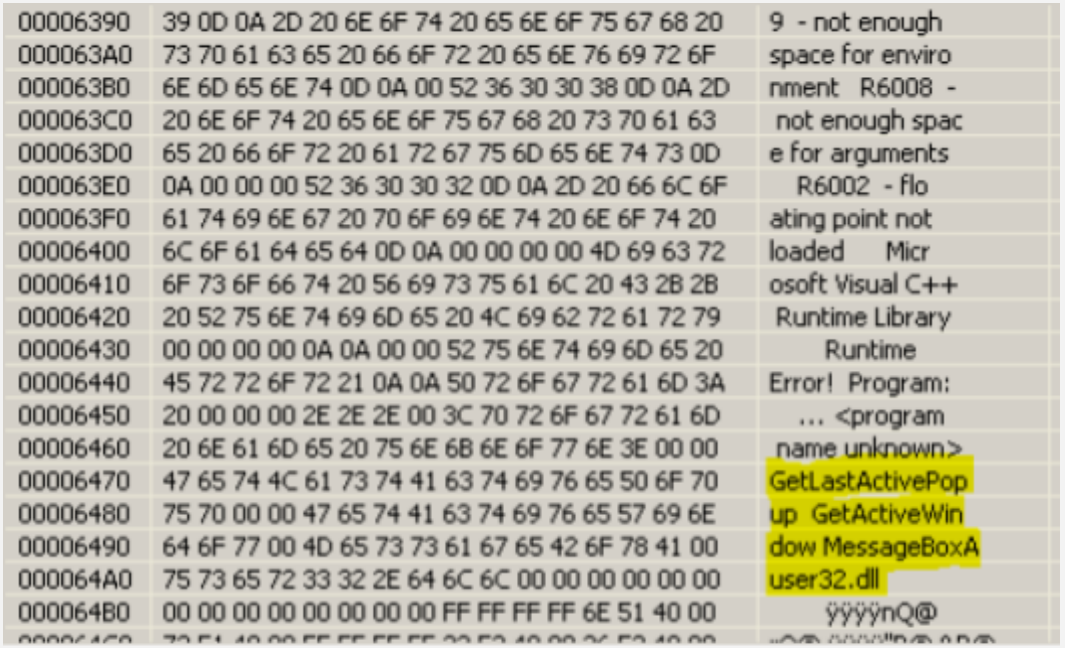
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00007030	45	72	72	6F	72	20	31	2E	31	3A	20	4E	6F	20	49	6E	Error.1.1:.No.In
00007040	74	65	72	6E	65	74	0A	00	53	75	63	63	65	73	73	3A	ternet..Success:
00007050	20	49	6E	74	65	72	6E	65	74	20	43	6F	6E	6E	65	63	.Internet.Connec
00007060	74	69	6F	6E	0A	00	00	00	45	72	72	6F	72	20	32	2E	tion....Error.2.
00007070	33	3A	20	46	61	69	6C	20	74	6F	20	67	65	74	20	63	3:.Fail.to.get.c
00007080	6F	6D	6D	61	6E	64	0A	00	45	72	72	6F	72	20	32	2E	ommand..Error.2.
00007090	32	3A	20	46	61	69	6C	20	74	6F	20	52	65	61	64	46	2:.Fail.to.ReadF
000070A0	69	6C	65	0A	00	00	00	00	45	72	72	6F	72	20	32	2E	ile.....Error.2.
000070B0	31	3A	20	46	61	69	6C	20	74	6F	20	4F	70	65	6E	55	1:.Fail.to.OpenU
000070C0	72	6C	0A	00	68	74	74	70	3A	2F	2F	77	77	77	2E	70	rl..http://www.p
000070D0	72	61	63	74	69	63	61	6C	6D	61	6C	77	61	72	65	61	racticalmalwarea
000070E0	6E	61	6C	79	73	69	73	2E	63	6F	6D	2F	63	63	2E	68	nalysis.com/cc.h
000070F0	74	6D	00	00	49	6E	74	65	72	6E	65	74	20	45	78	70	tm..Internet.Exp
00007100	6C	6F	72	65	72	20	37	2E	35	2F	70	6D	61	00	00	00	lorer.7.5/pma...
00007110	53	75	63	63	65	73	73	3A	20	50	61	72	73	65	64	20	Success:.Parsed.
00007120	63	6F	6D	6D	61	6E	64	20	69	73	20	25	63	0A	00	00	command.is.%c...

Figura 7. Contenuto Ascii di .data

Questa è una parte del contenuto ASCII della sezione .text. Questa acquisizione rivela informazioni molto importanti, che ci aiutano a dedurre diverse cose sul comportamento del malware. Questi sono gli errori che verrebbero visualizzati nel caso in cui non fosse possibile eseguire alcune parti della funzione del programma. Trascrivendo gli errori otteniamo quanto segue:

- Error.1.1: No Internet Success: Internet Connection
- Error 2.3: Fail to get command
- Error 2.2 : Fail to ReadFile
- Error 2.1: Fail to OpenUrl http://www.practical.malwareanalysis.com/cc.htm

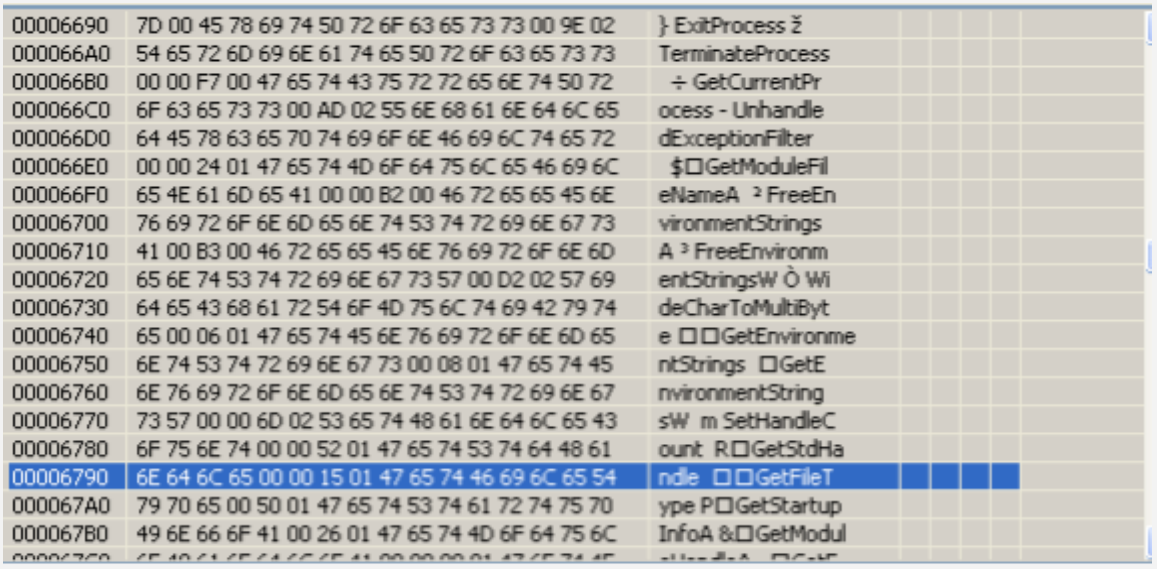
Basandosi sui messaggi di errore è facile dedurre che il file stia cercando di connettersi a Internet con un URL dal nome sospetto.



00006390	39 0D 0A 2D 20 6E 6F 74 20 65 6E 6F 75 67 68 20	9 - not enough
000063A0	73 70 61 63 65 20 66 6F 72 20 65 6E 76 69 72 6F	space for enviro
000063B0	6E 6D 65 6E 74 0D 0A 00 52 36 30 30 38 0D 0A 2D	nment R6008 -
000063C0	20 6E 6F 74 20 65 6E 6F 75 67 68 20 73 70 61 63	not enough spac
000063D0	65 20 66 6F 72 20 61 72 67 75 6D 65 6E 74 73 0D	e for arguments
000063E0	0A 00 00 00 52 36 30 30 32 0D 0A 2D 20 66 6C 6F	R6002 - flo
000063F0	61 74 69 6E 67 20 70 6F 69 6E 74 20 6E 6F 74 20	ating point not
00006400	6C 6F 61 64 65 64 0D 0A 00 00 00 00 4D 69 63 72	loaded Micr
00006410	6F 73 6F 66 74 20 56 69 73 75 61 6C 20 43 2B 2B	osoft Visual C++
00006420	20 52 75 6E 74 69 6D 65 20 4C 69 62 72 61 72 79	Runtime Library
00006430	00 00 00 00 0A 0A 00 00 52 75 6E 74 69 6D 65 20	Runtime
00006440	45 72 72 6F 72 21 0A 0A 50 72 6F 67 72 61 6D 3A	Error! Program:
00006450	20 00 00 00 2E 2E 2E 00 3C 70 72 6F 67 72 61 6D	... <program
00006460	20 6E 61 6D 65 20 75 6E 6B 6E 6F 77 6E 3E 00 00	name unknown>
00006470	47 65 74 4C 61 73 74 41 63 74 69 76 65 50 6F 70	GetLastActivePop
00006480	75 70 00 00 47 65 74 41 63 74 69 76 65 57 69 6E	up GetActiveWin
00006490	64 6F 77 00 4D 65 73 73 61 67 65 42 6F 78 41 00	dow MessageBoxA
000064A0	75 73 65 72 33 32 2E 64 6C 6C 00 00 00 00 00 00	user32.dll
000064B0	00 00 00 00 00 00 00 00 FF FF FF FF 6E 51 40 00	ÿÿÿÿnQ@

Figura 8. Ascii .rdata

Analizziamo ora il contenuto Ascii di .rdata, la sezione con più informazioni. Ci sono molte funzioni in questa sezione, le prime che possiamo vedere sono GetLastActivePopUp e GetActiveWindow. La prima funzione determina quale finestra popup di proprietà della finestra specificata era attiva più recentemente e la seconda funzione recupera l'handle della finestra attiva allegata alla coda di messaggi del thread chiamante.



00006690	7D 00 45 78 69 74 50 72 6F 63 65 73 73 00 9E 02	} ExitProcess ž
000066A0	54 65 72 6D 69 6E 61 74 65 50 72 6F 63 65 73 73	TerminateProcess
000066B0	00 00 F7 00 47 65 74 43 75 72 72 65 6E 74 50 72	÷ GetCurrentPr
000066C0	6F 63 65 73 73 00 AD 02 55 6E 68 61 6E 64 6C 65	ocess - Unhandle
000066D0	64 45 78 63 65 70 74 69 6F 6E 46 69 6C 74 65 72	dExceptionFilter
000066E0	00 00 24 01 47 65 74 4D 6F 64 75 6C 65 46 69 6C	$□GetModuleFil
000066F0	65 4E 61 6D 65 41 00 00 B2 00 46 72 65 65 45 6E	eNameA ² FreeEn
00006700	76 69 72 6F 6E 6D 65 6E 74 53 74 72 69 6E 67 73	vironmentStrings
00006710	41 00 B3 00 46 72 65 65 45 6E 76 69 72 6F 6E 6D	A ³ FreeEnvironm
00006720	65 6E 74 53 74 72 69 6E 67 73 57 00 D2 02 57 69	entStringsW Ò Wi
00006730	64 65 43 68 61 72 54 6F 4D 75 6C 74 69 42 79 74	deCharToMultiByt
00006740	65 00 06 01 47 65 74 45 6E 76 69 72 6F 6E 6D 65	e □□GetEnvironme
00006750	6E 74 53 74 72 69 6E 67 73 00 08 01 47 65 74 45	ntStrings □GetE
00006760	6E 76 69 72 6F 6E 6D 65 6E 74 53 74 72 69 6E 67	nvironmentString
00006770	73 57 00 00 6D 02 53 65 74 48 61 6E 64 6C 65 43	sW m SetHandleC
00006780	6F 75 6E 74 00 00 52 01 47 65 74 53 74 64 48 61	ount R□GetStdHa
00006790	6E 64 6C 65 00 00 15 01 47 65 74 46 69 6C 65 54	ndle □□GetFileT
000067A0	79 70 65 00 50 01 47 65 74 53 74 61 72 74 75 70	ype P□GetStartup
000067B0	49 6E 66 6F 41 00 26 01 47 65 74 4D 6F 64 75 6C	InfoA &□GetModul

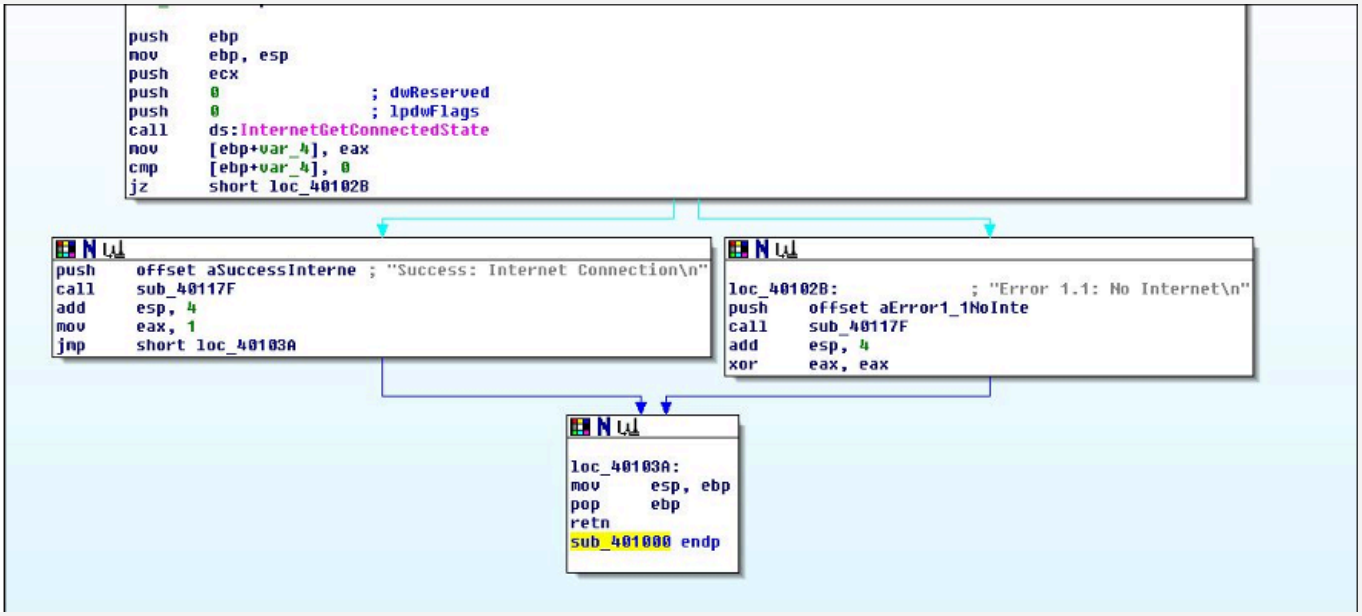
Figura 9. Ascii .rdata

Come visto in figura 9 Il file è pieno di funzioni Get, tutte utilizzate per raccogliere informazioni di sistema o di rete eseguendo il programma. Le informazioni recuperate variano dai tipi di file alle informazioni di avvio.

Esistono anche funzioni che possono compromettere l'integrità del sistema, come GetCPinfo. La funzione "GetCPInfo" può fornire informazioni preziose sulla lingua e sulle capacità di codifica dei caratteri di un sistema, che potrebbero essere sfruttate dal malware per eludere il rilevamento, manipolare la codifica del testo, localizzare i payload e sfruttare le vulnerabilità specifiche della lingua.

Analisi statica avanzata | Assembly

Leggere il codice Ascii ci dà un'idea generale del comportamento del malware, ma se vogliamo avere un'idea migliore dobbiamo vedere il codice Assembly che ci dice direttamente ogni azione che il programma eseguirà.



La Figura 10 è un diagramma di una parte di codice assembly che determina il comportamento in base al risultato della funzione InternetGetConnectedState. La funzione InternetGetConnectedState è una delle tante funzioni "get" eseguite dal programma. Questa funzione controlla se il computer della vittima dispone di una connessione Internet.

In sintesi, il codice viene eseguito come segue:

```
push    ebp
mov     ebp, esp
push    ecx
```

- Per prima cosa crea uno stack tramite le istruzioni **push ebp** e **mov ebp, esp**.

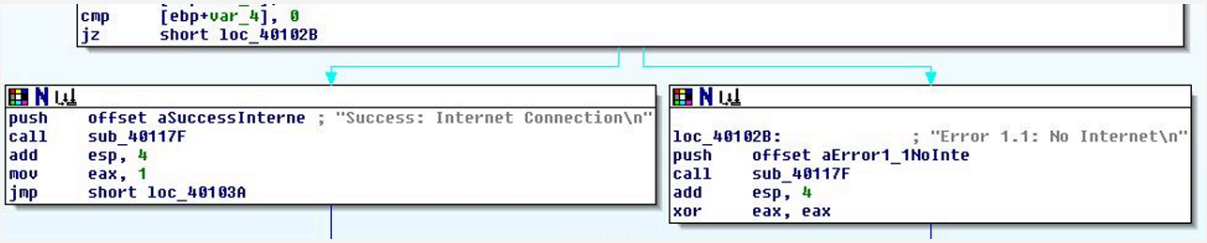
- Quindi con le 2 righe di codice **push 0** prepara gli argomenti che verranno utilizzati nella funzione richiamata con **call** nella riga successiva.

```
push    0                       ; dwReserved
push    0                       ; lpdwFlags
call    ds:InternetGetConnectedState
```

- Una volta eseguita questa funzione, nel registro **ebp+var_4** viene salvato il valore **1** o **0** (eax), per indicare positivo o negativo che viene confrontato con **0**. Se entrambi i valori sono **0**, ciò indica che non è stata trovata una connessione stabilita tra i computer e Internet, quindi salta per eseguire le istruzioni all'indirizzo **40102B**

```
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

- Come si vede nello schema, a sinistra c'è il codice che verrà eseguito se il salto non è stato eseguito (cioè non ha trovato una connessione internet). Questo codice chiama quindi la funzione sub_40117F (probabilmente printf) per stampare un messaggio "Success: Internet Connection" e poi passa ad un'altra riga di codice che vedremo tra poco.
- A destra invece ci sono le istruzioni che comporterebbe il salto a 40102B. Queste istruzioni forniscono il messaggio di errore "Error 1.1: No Internet"



- In quest'ultima sezione questa parte di codice è terminata, indipendentemente dal percorso intrapreso. Esegue semplicemente la funzione di “stack cleanup" in cui lo spazio di memoria utilizzato viene liberato per evitare problemi di gestione della memoria.

Conclusioni

In conclusione, questo malware sembra raccogliere molte informazioni dal sistema attacato, per poi inviarle all'URL http://www.practical.malwareanalysis.com/cc.htm, dove sicuramente si trova un server in ascolto controllato dal l'aggressore.