
Giacomo Manca - Epicode

Malware analysis e assembly

01 Dicembre, 2023

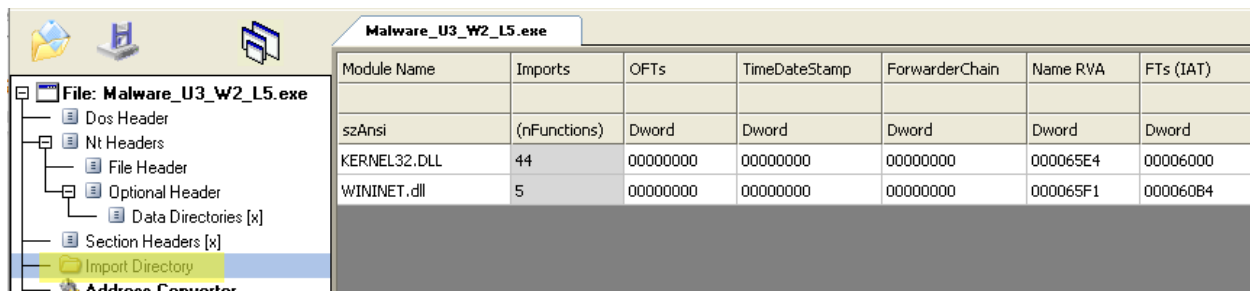
Introduzione

Lo scopo di questo progetto è analizzare il file malware “**Malware_U3_W2_L5**”. Andremo ad effettuare un’analisi statica per esaminare i contenuti del file senza eseguirlo. Andremo poi ad esaminare una porzione di codice assembly (architettura x86 32 bit).

Obiettivi

1. Identificare le librerie vengono importate dal file eseguibile.
2. Identificare le sezioni di cui si compone il file eseguibile del malware.
3. Analizzare il codice assembly e individuare i vari costrutti (creazione dello stack, eventuali cicli).
4. Ipotesizzare il comportamento della funzionalità implementata.

1. Librerie



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.DLL	44	00000000	00000000	00000000	000065E4	00006000
WININET.dll	5	00000000	00000000	00000000	000065F1	000060B4

OFTs	FTs (IAT)	Hint	Name
N/A	000060C0	00006900	00006902
Dword	Dword	Word	szAnsi
N/A	000068C6	0000	InternetOpenUrlA
N/A	000068D8	0000	InternetCloseHandle
N/A	000068EE	0000	InternetReadFile
N/A	00006900	0000	InternetGetConnectedState
N/A	0000691C	0000	InternetOpenA

Utilizzando il programma **CFF Explorer** analizziamo la struttura del file eseguibile.

Dalla sezione **Import Directory** possiamo osservare come il malware abbia importato le seguenti librerie:

- **KERNEL32.dll**: fornisce funzioni di base per l'interazione tra il software dell'applicazione e il kernel del sistema operativo.
- **WININET.dll**: gestisce le operazioni di rete e la comunicazione su Internet.

2. Sezioni del malware

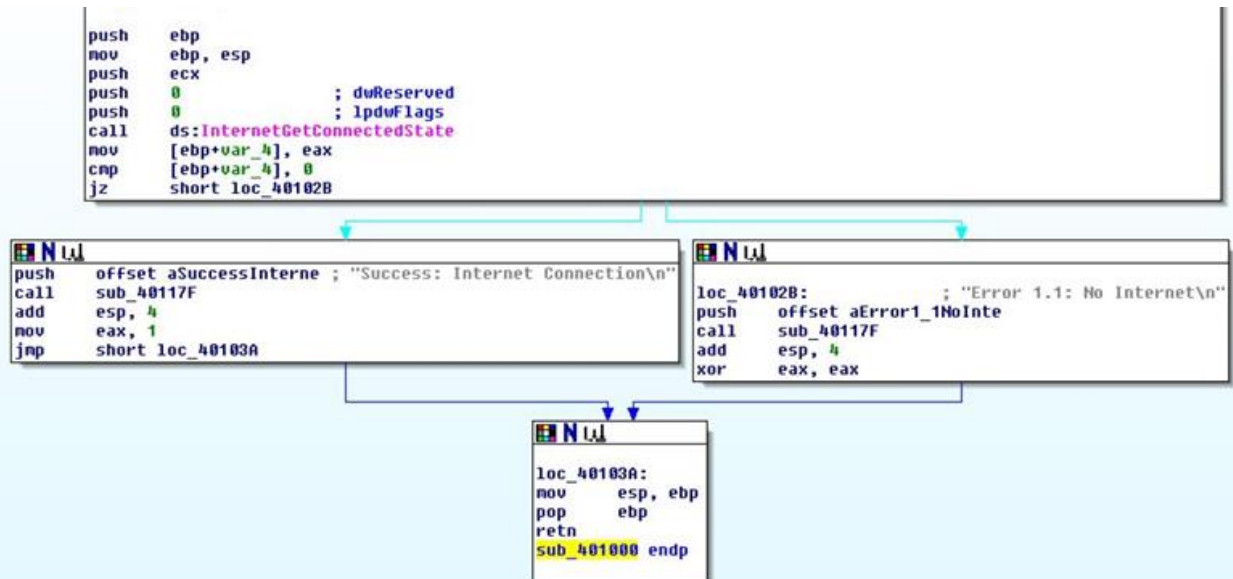
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	00008000	00001000	00000000	00000400	00000000	00000000	0000	0000	E0000080
UPX1	00004000	00009000	00003A00	00000400	00000000	00000000	0000	0000	E0000040
UPX2	00001000	0000D000	00000200	00003E00	00000000	00000000	0000	0000	C0000040

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

Tramite la UPX utility possiamo “spacchettare” il contenuto delle sezioni per poterne leggere il nome e i contenuti. Queste sono le seguenti:

- **.text**: contiene il codice eseguibile del programma che verrà interpretato ed eseguito dal processore quando il programma viene avviato.
- **.rdata**: contiene dati di sola lettura (*read only*) che non vengono modificati durante l'esecuzione.
- **.data**: contiene dati modificabili come variabili globali e altre informazioni che il programma utilizza durante l'esecuzione.

3. Analisi del codice assembly



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

In questa parte c'è la creazione dello stack con l'istruzione **push** che aggiunge un elemento nello stack. I registri **ebp** e **esp** rappresentano la base e la cima dello stack.

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

Qui abbiamo la chiamata di funzione **InternetGetConnectedState**, già individuata precedentemente nell'analisi delle librerie importate dal malware.

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

A seguire abbiamo il **ciclo if** nel quale vengono messi a confronto dei valori con l'istruzione **cmp** (*compare*): in questo caso l'istruzione **jz** (*jump zero*) effettua un salto alla cella di memoria indicata se la **zero flag** è impostata. Questa è un'istruzione di **salto condizionato**.

```
lock_40103A:
mov     esp,ebp
pop     ebp
retn
sub_401000 endp
```

In questa ultima parte c'è la **chiusura dello stack**: con l'istruzione **pop** viene rimosso dalla cima dello stack il valore e memorizzato in un registro o un'altra locazione di memoria. Questa fase viene anche chiamata "pulizia dello stack", ovvero il ripristino del valore originale del puntatore modificato durante la funzione.



Successivamente al ciclo if abbiamo i due blocchi di codice, che corrispondono rispettivamente alla condizione **true** (sinistra) e **false** (destra).

4. Ipotizzare il comportamento della funzionalità implementata

Dall'analisi del codice, in particolare del ciclo if, possiamo ipotizzare che il programma voglia verificare lo stato della connessione tramite la funzione **InternetGetConnectedState**, e una volta effettuata la verifica andrà a stampare a schermo il messaggio di connessione avvenuta o di errore.

5. Approfondimenti

```
C:\Documents and Settings\Epicode_user\Desktop\Malanalysis\md5deep-4.3>md5deep "
C:\Documents and Settings\Epicode_user\Desktop\Malanalysis\Esercizio_Pratico_U3_
W2_L5\Malware_U3_W2_L5.exe"
c0b54534e188e1392f28d17faff3d454 C:\Documents and Settings\Epicode_user\Desktop
\Malanalysis\Esercizio_Pratico_U3_W2_L5\Malware_U3_W2_L5.exe
C:\Documents and Settings\Epicode_user\Desktop\Malanalysis\md5deep-4.3>
```

Per ottenere ulteriori informazioni sul file senza eseguirlo ne ho estrapolato l'hash, per poi fare una ricerca sul **Virustotal**. Il risultato della ricerca è una conferma della natura malevola del file.



Community Score

39 security vendors and no sandboxes flagged this file as malicious

b71777edbf21167c96d20ff803cbcb25d24b94b3652db2f286dcd6efd3d8416a

Lab06-02.exe

peexe

checks-network-adapters

runtime-modules

armadillo

direct-cpu-clock-access

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 7

[Join the VT Community](#) and enjoy additional community insights and crowdsourced detections, plus an API key to [auto](#)

Popular threat label trojan.r002c0pdm21

Threat categories trojan