

# Analisi statica basica

Lo scopo dell'esercizio è di analizzare un file denominato **Malware\_U3\_W2\_L5** all'interno della macchina virtuale dedicata all'analisi dei malware.

Una volta individuato il file ho proceduto ad analizzarlo con gli appositi strumenti ed ho identificato:

## 1. le seguenti librerie (con alcune delle funzioni da loro importate):

Malware_U3_W2_L5.exe						
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064E0	000064E4	000064E8	000064EC
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep
00006940	00006940	027C	SetStdHandle
0000692E	0000692E	0156	GetStringTypeW
0000691C	0000691C	0153	GetStringTypeA
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA
000068E6	000068E6	01E4	MultiByteToWideChar
00006670	00006670	00CA	GetCommandLineA
00006682	00006682	0174	GetVersion
00006690	00006690	007D	ExitProcess
0000669E	0000669E	029E	TerminateProcess

La libreria **KERNEL32.dll** è una libreria che serve per interagire col sistema operativo, permette infatti la manipolazione dei file e la gestione della memoria.

La seconda libreria utilizzata (screenshot sottostante) prende il nome di **WININET.dll** e viene utilizzata per l'implementazione di alcuni protocolli di rete come HTTP, FTP e NTP.

Malware_U3_W2_L5.exe						
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
00006664	N/A	000064F0	000064F4	000064F8	000064FC	00006500
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

## 2. Le sezioni di cui si compone il malware:

Com'è possibile vedere nello screenshot sottostante questo malware si compone di tre sezioni.

La sezione **.text** contiene le istruzioni che verranno inviate alla CPU per essere eseguite all'avvio del software.

La sezione **.rdata** contiene le informazioni riguardo le librerie e le funzioni importate ed esportate.

La sezione **.data** contiene le variabili globali del programma.

Malware_U3_W2_L5.exe									
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
000001E0	000001E8	000001EC	000001F0	000001F4	000001F8	000001FC	00000200	00000202	00000204
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

Per ottenere alcune informazioni aggiuntive ho effettuato uno step ulteriore che fa sempre parte dell'analisi statica basica, ovvero ho estrapolato l'hash del file in questione:

**c0b54534e188e1392f28d17faff3d454**

l'ho inserito su **VirusTotal** per un'analisi preliminare. Qui di seguito è possibile osservare i risultati

ottenuti:

39  
/ 71

Community Score

39 security vendors and no sandboxes flagged this file as malicious

Reanalyze

Similar

More

b71777edbf21167c96d20ff803cbcb25d24b94b3652db2f286dcd6efd3d8416a

Size  
40.00 KB

Last Analysis Date  
5 months ago

EXE

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 [automate checks](#).

Popular threat label trojan.r002c0pdm21

Threat categories trojan

Family labels r002c0pdm21

Security vendors' analysis

Do you want to automate checks?

Alibaba	Trojan:Win32/Generic.be125c32	Antiy-AVL	Trojan:Win32.BTS.Generic
Avast	Win32:Trojan-gen	AVG	Win32:Trojan-gen
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	Cybereason	Malicious.1fef74
Cylance	Unsafe	Cynet	Malicious (score: 100)
DeepInstinct	MALICIOUS	DrWeb	Trojan.MulDrop7.63090
Elastic	Malicious (high Confidence)	ESET-NOD32	Win32/Agent.WOO
Fortinet	W32/Agent.WOO!tr	GData	Win32.Trojan.Agent.DZ3C1W
Google	Detected	Gridinsoft (no cloud)	Ransom.Win32.Wacatac.oals1
Ikarus	Trojan.Win32.Agent	Lionic	Trojan.Win32.Generic.4lc
Malwarebytes	Generic.Trojan.Malicious.DDS	MAX	Malware (ai Score=97)
MaxSecure	Trojan.Malware.300983.susgen	McAfee	GenericRXAA-AAIC0B54534E188
McAfee-GW-Edition	Artemis!Trojan	Microsoft	Trojan:Win32/Ymacco.AAB7
NANO-Antivirus	Trojan.Win32.Agent.dveqkx	Rising	Trojan.Agent!B.B1E (TFE:5:W5kRuOpSwdF)
Sangfor Engine Zero	Trojan.Win32.Agent.Vrlo	Symantec	ML.Attribute.HighConfidence
TACHYON	Trojan/W32.Agent.40960.ESE	Tencent	Malware.Win32/Gencirc.115cd777
Trellix (FireEye)	Generic.mg.c0b54534e188e139	TrendMicro	TROJ_GEN.R002C0PDM21
TrendMicro-HouseCall	TROJ_GEN.R002C0PDM21	VBA32	Suspected Of Trojan.Downloader.gen
VirIT	Trojan.Win32.Agent5.CRS	Webroot	W32.Malware.Heur

Come si evince dall'immagine il file viene indicato come **trojan** da diverse fonti e quindi questo ci dà un'ulteriore conferma sulla natura del file stesso.

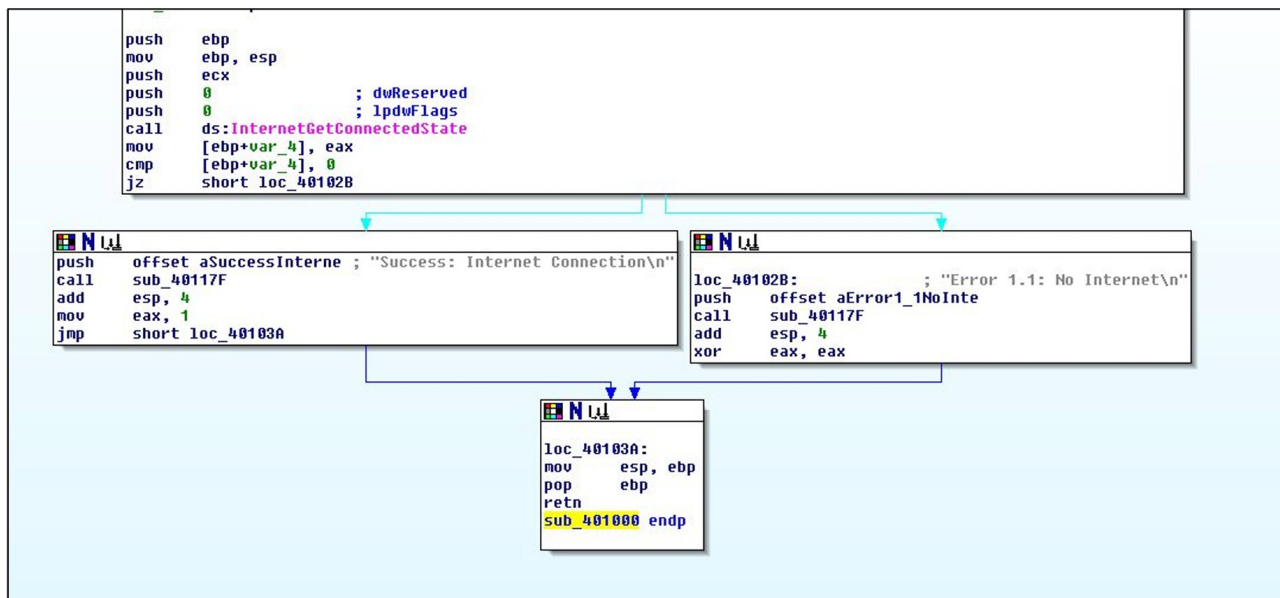
Ho proceduto anche all'estrapolazione delle stringhe contenute all'interno del file però si evince ben poco se non che cerca di instaurare una connessione ad internet.

Error 1.1: No Internet  
Success: Internet Connection

## Analisi codice assembly

Nella seconda parte dell'esercizio invece è necessario concentrarsi sul codice assembly fornitoci (vedi immagine sottostante).

Nello specifico quello che viene analizzato è assembly riferito all'architettura X86 (32 bit).



In particolare bisogna:

### 3. Identificare i costrutti noti

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

Il primo costrutto che troviamo è composto da queste due righe e consiste nella **creazione dello stack**. Si capisce dall'istruzione **push** utilizzata per aggiungere un elemento allo stack, e dall'utilizzo dei due registri **EBP / ESP** che vengono utilizzati rispettivamente per indicare la base e la cima dello stack.

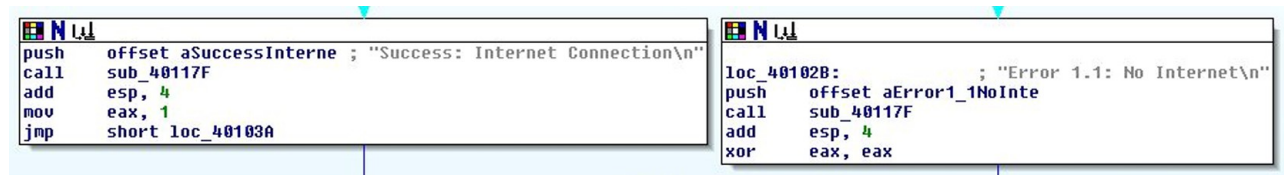
Poi abbiamo un secondo costrutto che rappresenta un **if**, in pratica viene fatto un compare tra due valori e successivamente viene utilizzato il **jz (jump zero)**, ovvero se lo **Zero Flag** è 1 allora verrà effettuato il jump alla cella di memoria indicata. Tradotto in codice si ha una condizione che coinvolge questi due valori e se questa condizione si verifica si andrà nel ramo del true altrimenti nel ramo dell'else. Questa diramazione è rappresentata dal jump.

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

Un altro costrutto che sono riuscito ad identificare è quello della **chiusura dello stack** che come per l'apertura utilizza gli stessi puntatori solamente invertiti e il comando **pop** per rimuovere l'ultimo elemento

```
loc_40103A:
mov     esp, ebp
pop     ebp
```

E' possibile anche individuare quale parte del codice è nel ramo del vero e quale nel falso:



Il blocco a sinistra sarà nel ramo del vero e quello a destra nel ramo del falso, questo perché il ramo del vero presenta un salto verso la chiusura dello stack mentre il ramo del falso presenta un salto iniziale nel caso il cmp sia falso. Il ramo del vero avrà un salto finale poiché il codice del ramo del falso presumibilmente sarà immediatamente conseguente e quindi non dovrà essere eseguito nel caso la condizione sia vera.

L'ultimo costrutto che sono riuscito ad identificare riguarda la chiamata alla funzione **InternetGetConnectdState** e l'inizializzazione dei suoi due parametri nelle righe precedenti (vedere immagine sottostante):

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

#### 4. Ipotizzare il comportamento della funzionalità implementata

Analizzando il codice assembly nella sua interezza si riesce a ipotizzare che il programma chiami la funzione **internetGetConnectedState** per ottenere lo stato della connessione, dopodiché farà un controllo sulla variabile (molto probabilmente fa riferimento allo stato della connessione) e inizierà e stamperà (presumibilmente) il messaggio di successo o errore a seconda che la condizione sia vera o falsa. Una volta fatto questo il programma si concluderà.

#### 5. BONUS fare tabella con significato delle singole righe di codice assembly

La parte in blu evidenzia il ramo del true e quella gialla il ramo del false:

Istruzione	Significato
Push ebp	Inizializza lo stack inserendo il primo valore
Mov ebp, esp	Copia il valore di esp in ebp
Push ecx	Il valore di ecx viene salvato sulla pila dello stack
Push 0 ; dwReserved	Aggiunge il valore sulla pila dello stack e rappresenta l'argomento dwReserved
Push 0 ; lpdwFlags	Come sopra però rappresenta l'argomento lpdwFlags
Call ds:InternetGetConnctedState	La funzione viene chiamata usando il segmento dati ds. I suoi parametri sono quelli delle due righe precedenti e sono entrambi inizializzati a 0

Mov [ebp+var_4], eax	Il contenuto di eax viene copiato nella posizione di memoria indicata dalla destinazione
Cmp [ebp+var_4], 0	Viene effettuato un confronto tra il valore precedentemente copiato e 0 e lo Zero Flag verrà impostato a seconda del risultato del confronto
Jz short loc_40102B	Con questa istruzione si effettua un salto condizionale all'indirizzo indicato dalla cella di memoria. In particolare la "z" indica che il salto viene effettuato se lo Zero Flag è settato a 1
Push offset aSuccessInterne ; "Success: Internet Connection\n"	Viene inserita la stringa indicata dai doppi apici all'interno della variabile aSuccessInterne
Call sub_40117F	Qui viene chiamata la subroutine ovvero la funzione assegnata a quell'etichetta
Add esp, 4	Viene aggiunto 4 allo stack pointer esp. Di solito quest'operazione viene effettuata per ripulire la pila dopo che una funzione è stata chiamata
Mov eax, 1	Il valore 1 viene aggiunto al registro eax
Jmp short loc_40103A	Viene fatto un salto incondizionato alla cella di memoria indicata
Push offset aError1_1NoInte ; "Error 1.1: No Internet\n"	Viene inserita la stringa indicata dai doppi apici all'interno della variabile aError1_1NoInte
Call sub_40117F	Qui viene chiamata la subroutine ovvero la funzione assegnata a quell'etichetta
Add esp, 4	Viene aggiunto 4 allo stack pointer esp. Di solito quest'operazione viene effettuata per ripulire la pila dopo che una funzione è stata chiamata
Xor eax, eax	Questa istruzione esegue lo xor tra eax e sé stesso. Questo fa sì che il registro venga azzerato
Mov esp, ebp	Viene copiato il valore di ebp in esp
Pop ebp	Qui viene estratto il valore dalla pila, si chiude lo stack
Retn	Corrisponde al return della funzione
Sub_401000 endp	Indica la fine di una funzione o subroutine

Davide Lecci