

---

Malware analysis

S10

Team

# Indice generale

01

Analisi Statica  
Basica

02

Analisi Dinamica  
Basica

03

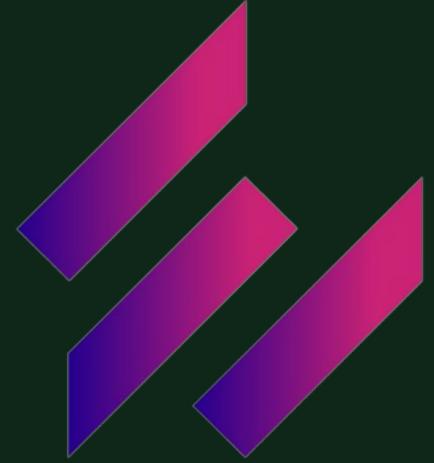
Introduzione ad  
Assembly

04

Assembly:  
Tecniche di  
Ingegneria inversa

05

Analisi Statica e  
Dinamica:  
approccio pratico



# GIORNO 1

L1

Analisi Statica Basica

# Traccia giorno 1

---

Con riferimento al file eseguibile contenuto nella cartella «Esercizio\_Pratico\_U3\_W2\_L1» presente sul Desktop della vostra macchina virtuale dedicata all'analisi dei malware, rispondere ai seguenti quesiti:

01

Indicare le librerie importate dal malware, fornendo una descrizione per ognuna di esse

02

Indicare le sezioni di cui si compone il malware, fornendo una descrizione per ognuna di essa

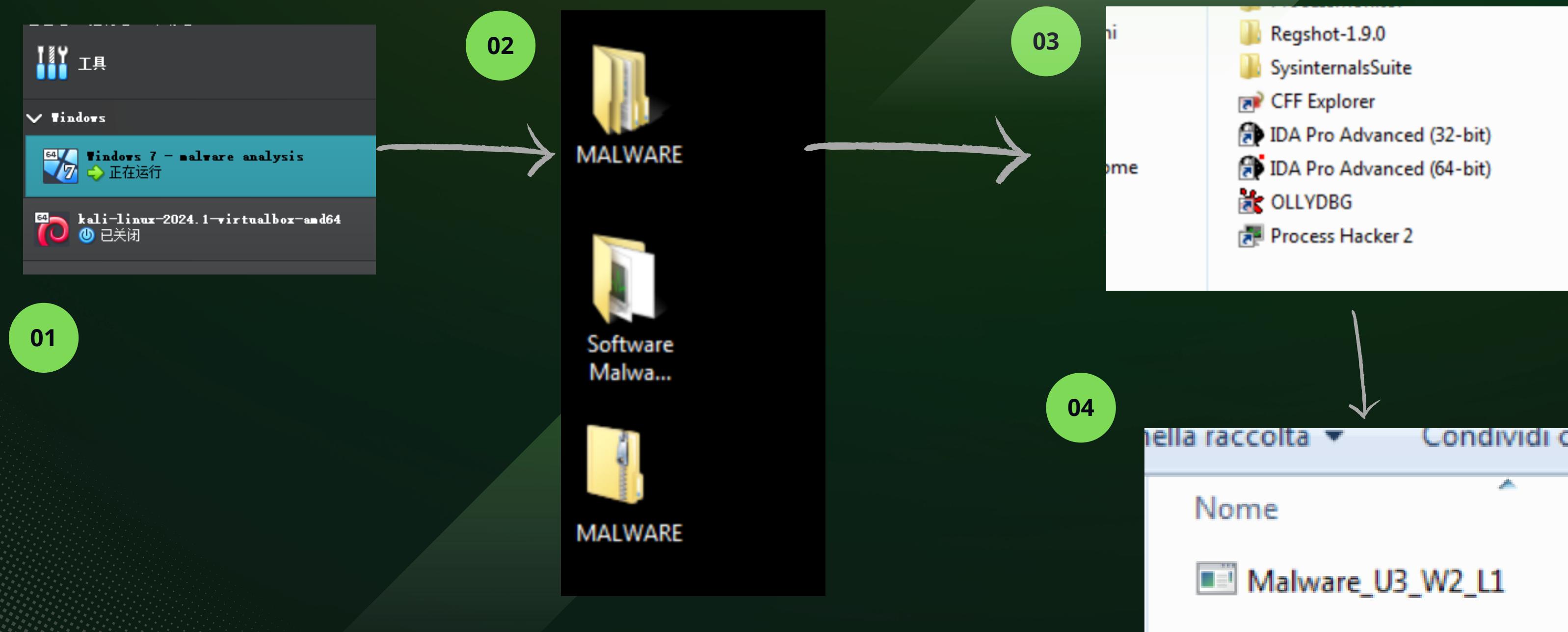
03

Aggiungere una considerazione finale sul malware in analisi in base alle informazioni raccolte

Per utilizzare Cff per analizzare il malware, è necessario seguire questi passaggi:

1. Scaricare e installare Windows 7.
2. Aprire la cartella "soft malware".
3. Avviare Cff Explorer.
4. Importare il file "Malware\_U3\_W2\_L1" in Cff Explorer.

Questi passaggi permetteranno di analizzare il malware utilizzando Cff Explorer su Windows



# CFF Explorer

CFF Explorer è uno strumento avanzato per l'analisi e la modifica di file eseguibili su Windows, parte della suite di strumenti chiamata Explorer Suite, sviluppata da NTCore. È particolarmente utile per programmati, analisti di malware e ricercatori di sicurezza informatica. Ecco alcune delle sue funzionalità principali:

1. Visualizzazione della struttura dei file PE: Permette di esplorare e modificare le intestazioni e le sezioni dei file Portable Executable (PE), come .exe e .dll.
2. Modifica degli import e degli export: Consente di visualizzare e modificare le tabelle degli import e degli export, essenziali per comprendere le dipendenze di un programma.
3. Risorse del file: Permette di visualizzare e modificare le risorse incorporate nel file, come icone, immagini, stringhe di testo e altri dati.
4. Editor HEX: Include un editor esadecimale per la modifica diretta dei dati binari del file.
5. Disassemblatore: Offre funzionalità di disassemblaggio per analizzare il codice macchina del file eseguibile.

CFF Explorer è utilizzato frequentemente nell'analisi di malware perché consente di esaminare la struttura interna dei file eseguibili sospetti, identificare potenziali comportamenti dannosi e apportare modifiche per ulteriori analisi o mitigazioni.

# LIBRERIE IMPORTATE

elenca tutte le funzioni e le librerie esterne che un file eseguibile o una libreria dinamica (.exe o .dll) necessita per funzionare correttamente.

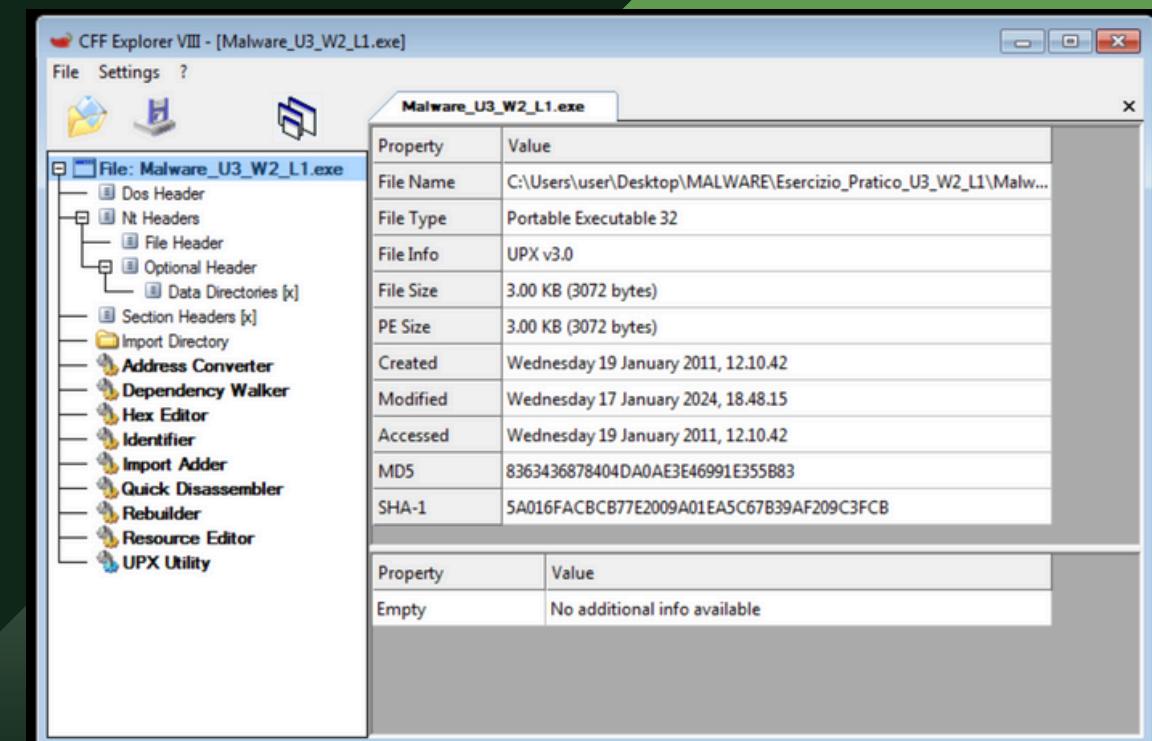
Utilizzando CFF Explorer, vediamo dalla sezione import directory che il malware U3\_W2\_L1 importa 4 librerie:

1. Kernel32.dll, che include le funzioni core del sistema operativo

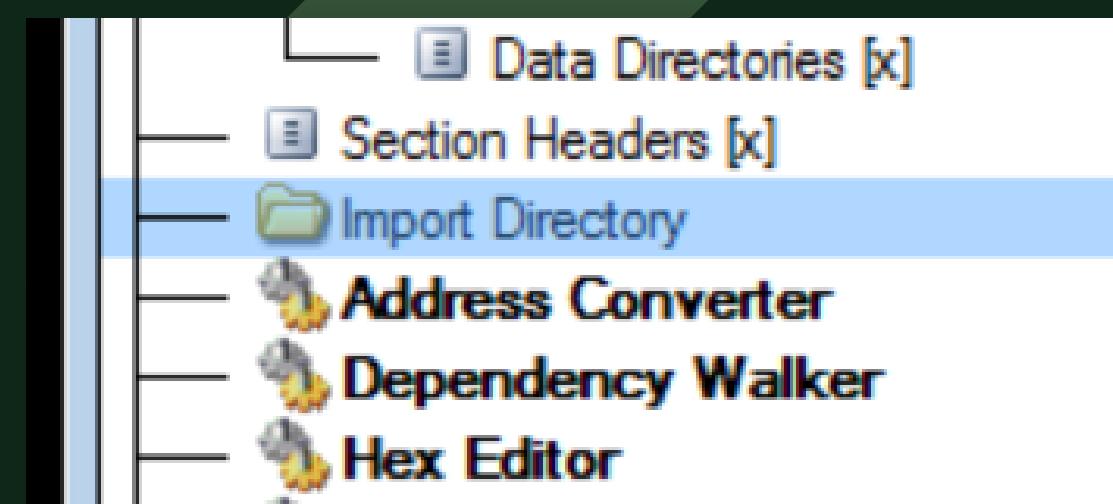
2. Advapi32.dll, che include le funzione per interagire con registri e servizi Windows

3. MSVCRT.dll, libreria scritta in C per la manipolazione scritte o allocazione memoria

4. Wininet.dll, include le funzione per implementare i servizi di rete come ftp, ntp, http



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.DLL	6	00000000	00000000	00000000	00006098	00006064
ADVAPI32.dll	1	00000000	00000000	00000000	000060A5	00006080
MSVCRT.dll	1	00000000	00000000	00000000	000060B2	00006088
WININET.dll	1	00000000	00000000	00000000	000060BD	00006090



# SECTION HEADER

testazione delle sezioni (section header) di un file eseguibile (PE, Portable Executable) su Windows è una parte cruciale che descrive le caratteristiche delle diverse sezioni del file. Ogni sezione può contenere codice, dati, risorse o altre informazioni necessarie per l'esecuzione del programma.

Da CFF Explorer, dalla sezione «section header» vediamo che l'eseguibile si compone di 3 sezioni. Purtroppo sembra che il malware abbia nascosto il vero nome delle sezioni e quindi non siamo in grado di capire che tipo di sezioni sono.

Malware_U3_W2_L1.exe									
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	00004000	00001000	00000000	00000400	00000000	00000000	0000	0000	E0000080
UPX1	00001000	00005000	00000600	00000400	00000000	00000000	0000	0000	E0000040
UPX2	00001000	00006000	00000200	00000A00	00000000	00000000	0000	0000	C0000040

# Considerazione finale

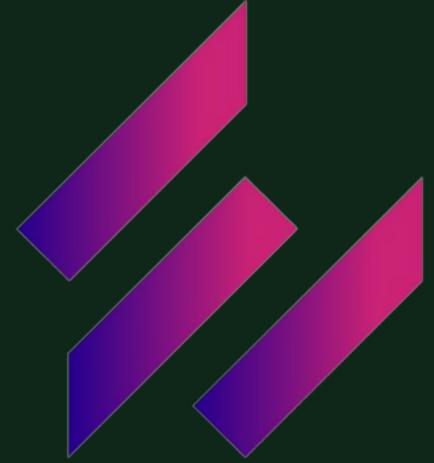
Si tratta di un malware avanzato che non ci consente di recuperare molte informazioni sul suo comportamento con l'analisi statica basica.

Ciò è supportato dal fatto che tra le funzioni importate troviamo «LoadLibrary e GetProcAddress», che ci fanno pensare ad un malware che importa le librerie a tempo di esecuzione (runtime) nascondendo di fatto le informazioni circa le librerie importate a monte.

Module Name	Imports	OFTs	TimeStamp
00000A98	N/A	00000A00	00000A04
szAnsi	(nFunctions)	Dword	Dword
KERNEL32.DLL	6	00000000	00000000
ADVAPI32.dll	1	00000000	00000000
MSVCRT.dll	1	00000000	00000000
WININET.dll	1	00000000	00000000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
N/A	000060C8	0000	LoadLibraryA
N/A	000060D6	0000	GetProcAddress
N/A	000060E6	0000	VirtualProtect
N/A	000060F6	0000	VirtualAlloc
N/A	00006104	0000	VirtualFree



# GIORNO 2

L2

Analisi Dinamica Basica

# Traccia giorno 2

---

Configurare la macchina virtuale per l'analisi dinamica (il malware sarà effettivamente eseguito).

Con riferimento al file eseguibile contenuto nella cartella «Esercizio\_Pratico\_U3\_W2\_L2» presente sul desktop della vostra macchina virtuale dedicata all'analisi dei malware, rispondere ai seguenti quesiti:

- 01 Identificare eventuali azioni del malware sul file system utilizzando Process Monitor (procmon)
- 02 Identificare eventuali azioni del malware su processi e thread utilizzando Process Monitor
- 03 Modifiche del registro dopo il malware (le differenze)
- 04 Provare a profilare il malware in base alla correlazione tra «operation» e Path.

# Configurazione macchina virtuale

In “rete” disabilitiamo tutte le interfacce di rete:



In “Sistema” disattiviamo il Floppy per poi disabilitare il controller USB in “Porte” > “USB”:

The image contains two side-by-side screenshots of a BIOS/UEFI setup interface. The left screenshot shows the 'System' tab with the following settings:

- 内存大小 (M): 4 MB
- 启动顺序 (B): 光驱, 硬盘, 软驱, 网络
- 芯片组 (C): PIIIx3
- TPM 模块: 空
- 指点设备 (D): PS/2 鼠标
- 扩展特性: 启用 I/O APIC, 硬件时钟使用国际标准时间 (UTC), 启用 EFI (只针对某些操作系统), 启用 Secure Boot 安全启动

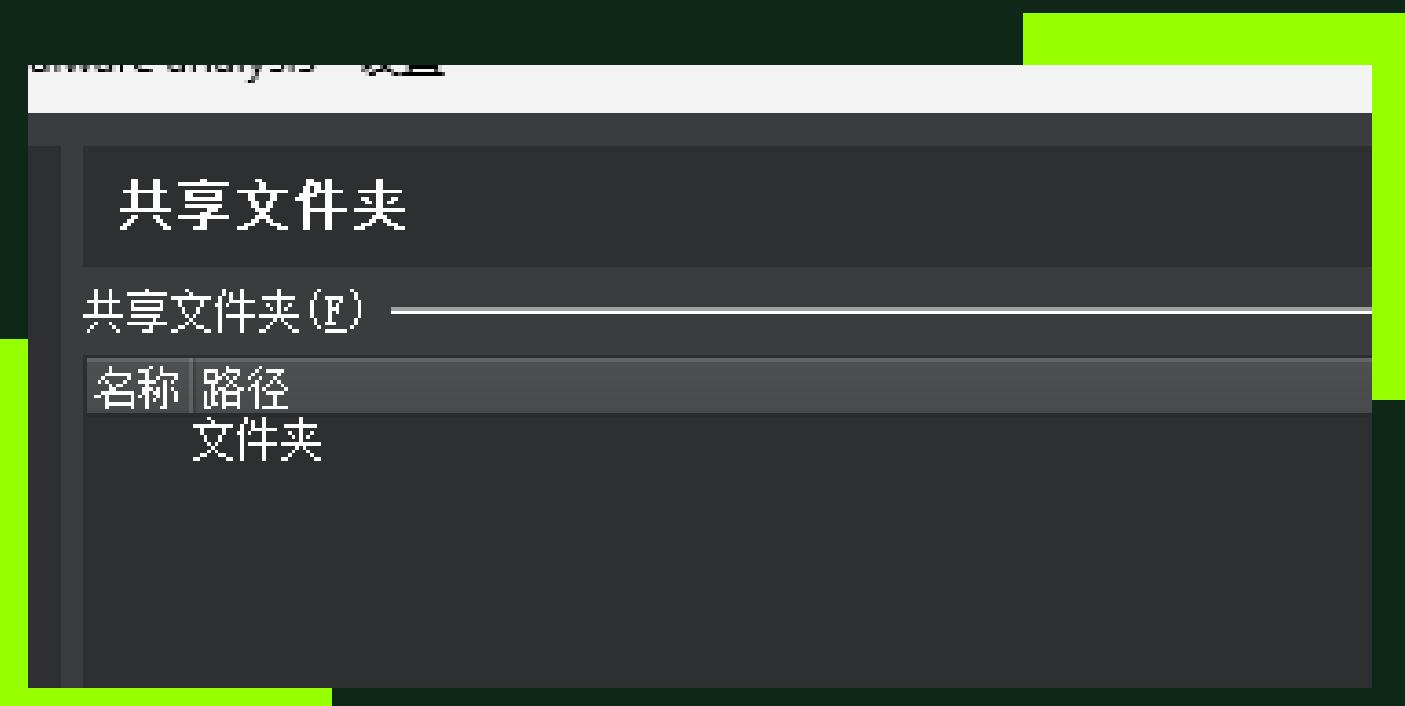
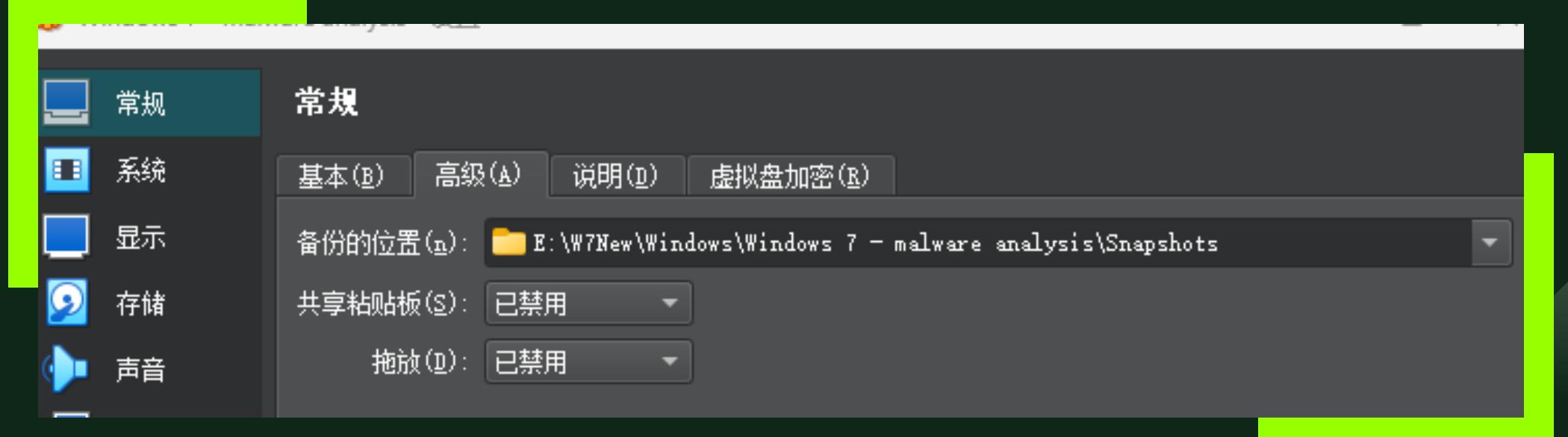
The 'Floppy' option under '启动顺序' is checked. The right screenshot shows the 'USB 设备' (USB Devices) configuration:

- 启用 USB 控制器 (Enable USB Controller): Unchecked
- USB 1.1 (OHCI) 控制器 (USB 1.1 (OHCI) Controller): Selected
- USB 2.0 (OHCI + EHCI) 控制器 (USB 2.0 (OHCI + EHCI) Controller): Unselected
- USB 3.0 (xHCI) 控制器 (USB 3.0 (xHCI) Controller): Unselected

A large green rectangular box highlights both screenshots.

# Configurazione macchina virtuale

In “Generale” > “Avanzate” disabilitare “appunti condivisi” e “trascina e rilascia”:



# Configurazione macchina virtuale

In



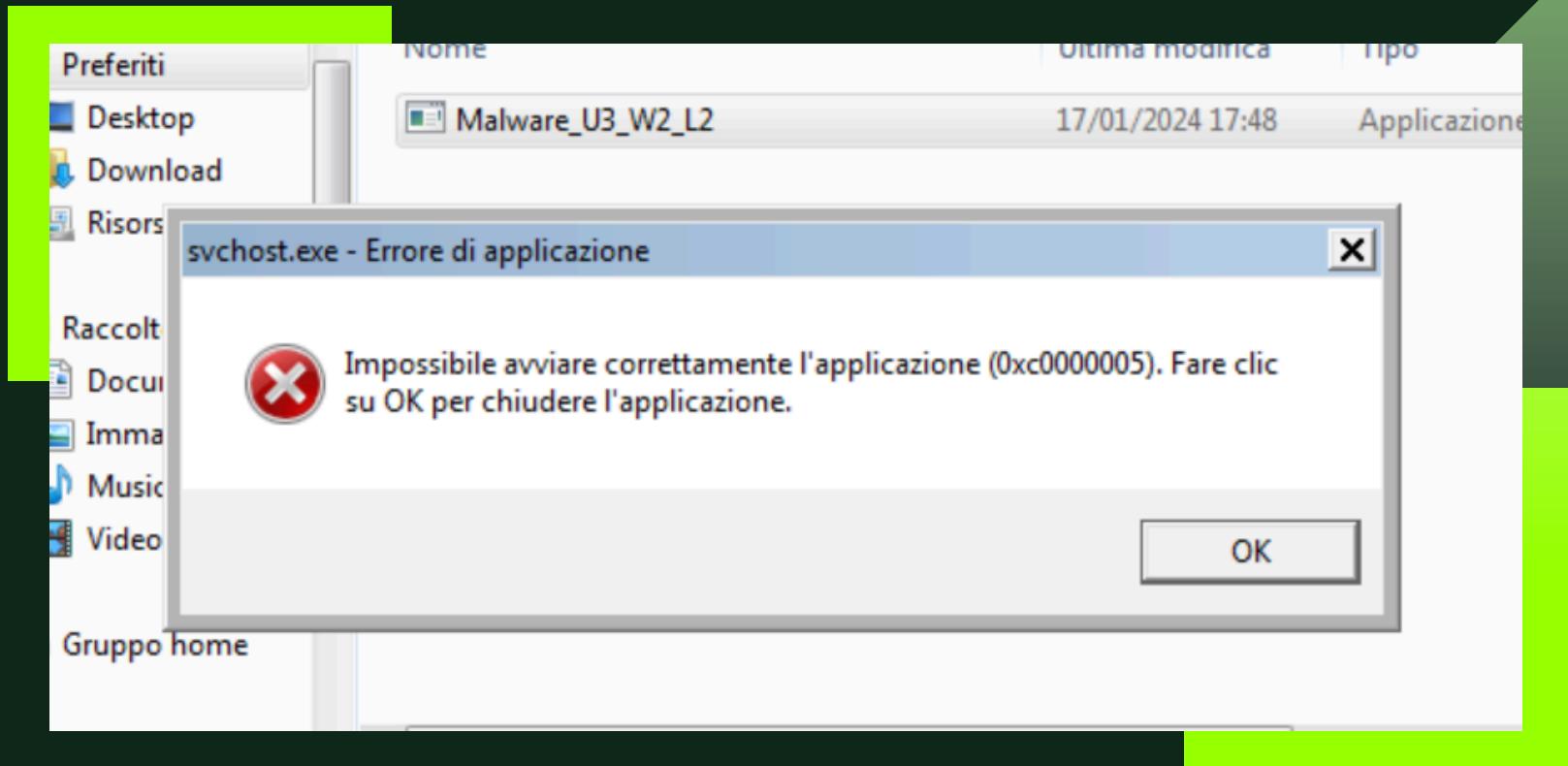
> “Istantanee” > “Crea”, creiamo un’istantanea della VM.

Questa operazione serve per avere un backup dello stato attuale della macchina in caso dopo l’esecuzione del malware questa venga compromessa.

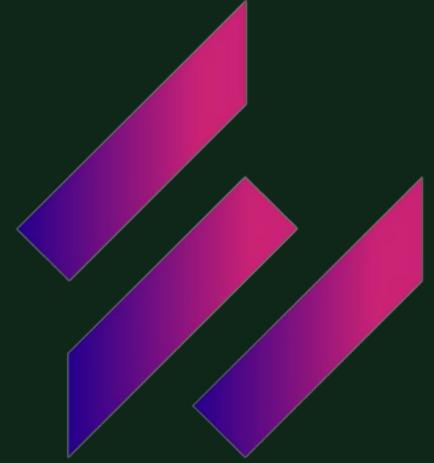


# Esecuzione del malware

Provando ad eseguire il Malware, il sistema ci restituisce il seguente errore:



Dopo svariate ricerche abbiamo appurato che il Malware è eseguibile su Windows XP e non su Windows 7. A questo punto abbiamo cercato un modo per copiare i file contenenti il malware da Windows 7 a Windows XP e lo abbiamo fatto creando un disco fisso VDI vuoto su VirtualBox. Abbiamo montato il disco vuoto su Windows 7 e abbiamo copiato su di esso i dati di nostro interesse, dopodiché abbiamo montato lo stesso disco (che a questo punto contiene i dati) su WindowsXP.



# GIORNO 3

L3

La memoria ed il linguaggio Assembly

# Traccia giorno 3

---

Dato il codice in Assembly per la CPU x86 allegato qui di seguito, identificare lo scopo di ogni istruzione, inserendo una descrizione per ogni riga di codice.

**01** 0x00001141 <+8>: mov EAX,0x20  
0x00001148 <+15>: mov EDX,0x38  
0x00001155 <+28>: add EAX,EDX  
0x00001157 <+30>: mov EBP, EAX  
0x0000115a <+33>: cmp EBP,0xa  
0x0000115e <+37>: jge 0x1176 <main+61>  
0x0000116a <+49>: mov eax,0x0  
0x0000116f <+54>: call 0x1030 <printf@plt>

# Linguaggio Assembly

La base del linguaggio Assembly sono le istruzioni. Nel linguaggio Assembly le istruzioni sono costituite da due parti:

- Un codice mnemonico, ovvero una parola che identifica l'istruzione da eseguire
- Uno o più operandi (per alcuni codici mnemonici, non è necessario l'operando come vedremo a breve), che identificano le variabili o la memoria oggetto dell'istruzione.

Un linguaggio assembly (detto anche linguaggio assemblativo o linguaggio assemblatore o semplicemente assembly) è un linguaggio di programmazione molto simile ai linguaggi macchina.

Si differenzia da questi ultimi principalmente per l'utilizzo di identificatori mnemonici, valori simbolici e altre caratteristiche che lo rendono più agevole da scrivere e leggere per gli esseri umani.

Erroneamente viene spesso chiamato assembler, ma quest'ultimo termine identifica solo l'applicativo che converte i programmi scritti in assembly nell'equivalente in linguaggio macchina.

# Conversione esadecimale -> decimale

Il sistema numerico esadecimale utilizza le cifre da 0 a 9 e le lettere da A a F (10-15). Per convertire un numero da base esadecimale a base decimale, è sufficiente moltiplicare la cifra o il carattere per 16 elevato alla posizione in cui si trova la cifra o il carattere proseguendo da destra verso sinistra.

Ecco un esempio pratico per chiarire il concetto:

3B ---->numero in base esadecimale

$$B = B \times 16^0 = 11 \times 16^0 = 11$$

$$3 = 3 \times 16^1 = 48$$

$11 + 48 = 59$  ----> numero corrispondente in base decimale

# Conversione decimale -> esadecimale

Il sistema numerico decimale utilizza le cifre da 0 a 9. Per convertire un numero da base decimale a base esadecimale, è sufficiente dividere il numero per 16, se il quoziente è diverso da zero va diviso nuovamente per 16 finché non sarà uguale a 0. A questo punto bisogna prendere il resto di ogni divisione in ordine opposto di come li abbiamo ottenuti.

Ecco un esempio pratico per chiarire il concetto:

59 ----->numero in base decimale

$59 : 16 = 3$  con resto 11

$3 : 16 = 0$  con resto 3

In riga scriviamo tutti i resti ottenuti, dall'ultimo al primo:

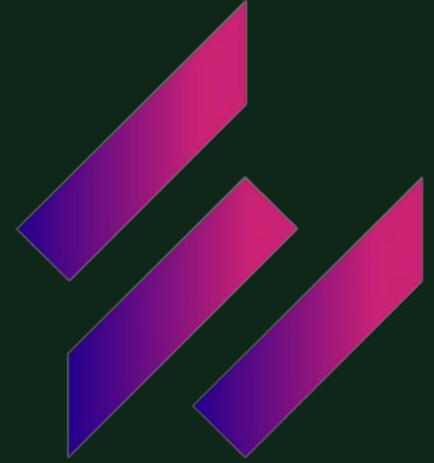
3 11

Sappiamo che in base esadecimale  $B = 11$  e quindi il risultato finale sarà

3B -----> numero in base esadecimale

# Descrizione codice

- 1.0x00001141 <+8>: mov EAX, 0x20 #Sposta il valore 0x20 (32 in decimale) nel registro EAX. Quindi EAX = 32
- 2.0x00001148 <+15>: mov EDX, 0x38 #Sposta il valore 0x38 (56 in decimale) nel registro EDX. Quindi EDX = 56
- 3.0x00001155 <+28>: add EAX, EDX #Aggiunge il valore in EDX a EAX. EAX = EAX + EDX = 32 + 56 = 88 (0x58)
- 4.0x00001157 <+30>: mov EBP, EAX #Sposta il contenuto di EAX in EBP. Quindi 88, che è il nuovo valore di EAX viene spostato in EBP. EBP = 88
- 5.0x0000115a <+33>: cmp EBP, 0xA #Confronta il valore in EBP con 0xA (10). EBP>0xA ---> 88 > 10
- 6.0x0000115e <+37>: jge 0x1176 <main+61> # jge, jump if greater or qual, Salta all'indirizzo 0x1176 se la relazione tra gli operatori valutati risulta essere maggiore o uguale. Questo prende il nome di salto condizionato. EBP > 0xA e quindi il salto viene effettuato
- 7.0x0000116a <+49>: mov EAX, 0x0 #Sposta il valore 0 in EAX. Quindi EAX = 0
- 8.0x0000116f <+54>: call 0x1030 <printf@plt> #Chiama la funzione printf allocata all'indirizzo 0x1030, e in questo caso non verrà eseguita perché è stato effettuato il salto direttamente all'indirizzo 0x1176



# GIORNO 4

ЛЧ

Linguaggio Assembly parte 2

# Traccia giorno 4

---

Traccia: La figura seguente mostra un estratto del codice di un malware. Identificare i costrutti noti Esercizio Linguaggio Assembly visti durante la lezione teorica.

```
* .text:00401000          push    ebp
* .text:00401001          mov     ebp, esp
* .text:00401003          push    ecx
* .text:00401004          push    0           ; dwReserved
* .text:00401006          push    0           ; lpdwFlags
* .text:00401008          call    ds:InternetGetConnectedState
* .text:0040100E          mov     [ebp+var_4], eax
* .text:00401011          cmp     [ebp+var_4], 0
* .text:00401015          jz      short loc_40102B
* .text:00401017          push    offset aSuccessInterne ; "Success: Internet Connection\n"
* .text:0040101C          call    sub_40105F
* .text:00401021          add    esp, 4
* .text:00401024          mov     eax, 1
* .text:00401029          jmp    short loc_40103A
.text:0040102B ; -----
.text:0040102B ; -----
```

# Identificare i costrutti

## 1. Identificare i costrutti noti (e s. while, for, if, switch, ecc.)

```
1.    00401000      push  ebp  
2.    00401001      mov   ebp,esp  
  
3.    00401003      push  ecx  
4.    00401004      push  0 ; dwReserved  
5.    00401006      push  0 ; lpdwFlags  
6.    00401008      call   ds:InternetGetConnectedState  
  
7.    0040100E      mov   [ebp+var_4], eax  
  
8.    00401011      cmp   [ebp+var_4],0  
9.    00401015      jz    short loc_40102B  
10.   00401017      push  offset aSuccessInterne ; "Succ....\n"  
11.   0040101C      call   sub_40105F  
12.   00401021      add   esp,4  
13.   00401024      mov   eax,1  
  
14.   00401029      jmp   short loc_40103A  
15.   0040102B  
16.   0040102B
```

Costrutto if



# Ipotizzare la funzionalità

## 2. Ipotizzare la funzionalità – esecuzione ad alto livello

Questa porzione di codice assembly è progettata per verificare lo stato della connessione a Internet su un sistema Windows. Utilizza la funzione InternetGetConnectedState della libreria WinINet per determinare se il sistema è attualmente connesso a Internet e agisce di conseguenza.

Il malware chiama la funzione internetgetconnectedstate e ne controlla con un «if» il valore di ritorno. Se il valore di ritorno (return) della funzione è diverso da 0, allora vuol dire che c'è una connessione attiva.

Pseudocodice C:

```
state = internetgetconnectedstate (par1,0,0);
If (state !=0) printf ("Active connection");
Else return 0;
```



# Bonus

## 3. BONUS: studiare e spiegare ogni singola riga di codice

01

Impostazione del Frame dello Stack

push ebp

mov ebp, esp

- Salva il valore corrente del puntatore di base (ebp) sullo stack.
- Imposta il puntatore di base (ebp) all'attuale puntatore dello stack (esp), creando un nuovo frame dello stack.

02

Salvataggio del Registro ecx

push ecx

- Salva il valore del registro ecx sullo stack per preservarlo, poiché sarà utilizzato nel corso della funzione.

03

Preparazione dei Parametri per InternetGetConnectedState

push 0 ; dwReserved

push 0 ; lpdwFlags

Imposta i parametri richiesti dalla funzione InternetGetConnectedState:

- lpdwFlags è un puntatore a una variabile che riceve la descrizione della connessione, ma qui non viene utilizzato (impostato a 0).
- dwReserved è riservato e deve essere 0.



# Bonus

## 3. BONUS: studiare e spiegare ogni singola riga di codice

04

La Chiamata alla Funzione InternetGetConnectedState

```
call ds:InternetGetConnectedState mov [ebp+var_4],
```

eax Chiama la funzione InternetGetConnectedState e memorizza il valore di ritorno nel registro eax.

Questo valore indica se il sistema ha una connessione a Internet attiva. Memorizza il risultato in una variabile locale (var\_4) nello stack.

05

Verifica dello Stato della Connessione Internet

```
cmp [ebp+var_4], 0
```

```
jz short loc_40102B
```

- Confronta il valore memorizzato in var\_4 con 0.
- Se var\_4 è 0 (nessuna connessione a Internet), salta all'etichetta loc\_40102B.

06

Stampa del Messaggio di Successo

```
push offset aSuccessInterne ; "Success\n"
```

```
call sub_4010fF
```

```
add esp, 4
```

- Se viene rilevata una connessione a Internet, impila l'indirizzo della stringa "Success\n" e chiama la funzione sub\_40105F (presumibilmente una funzione che stampa o gestisce la stringa).
- Dopo la chiamata, regola il puntatore dello stack (esp).



# Bonus

## 3. BONUS: studiare e spiegare ogni singola riga di codice

07

Impostazione del Valore di Ritorno e Uscita

```
mov eax, 1  
jmp short loc_40103A
```

- Imposta il valore di ritorno in eax a 1 (indicando successo) e salta all'etichetta loc\_40103A per uscire dalla funzione.

