

PROGETTO S10/L5

ANALISI STATICA E DINAMICA: UN APPROCCIO PRATICO

Traccia.....	1
Operazioni preliminari.....	2
- Configurazione schede di rete:.....	2
- Dispositivi USB:.....	3
- Cartelle condivise:.....	3
- Creazione di istantanee:.....	4
Malware analysis.....	5
Ma cos'è un malware?.....	6
CFF EXPLORER.....	7
Librerie importate.....	10
Sezioni.....	11
Costrutti noti.....	12
Assembly.....	13
Costrutto 1 (riquadro rosso).....	14
Costrutto 2 (riquadro rosa).....	14
Costrutto 3 (riquadro blu).....	15

Traccia

Con riferimento al file **Malware_U3_W2_L5** presente all'interno della cartella «**Esercizio_Pratico_U3_W2_L5**» sul desktop della macchina virtuale dedicata per l'analisi dei malware, rispondere ai seguenti quesiti:

- Quali librerie vengono importate dal file eseguibile?
-

- Quali sono le sezioni di cui si compone il file eseguibile del malware?

Con riferimento alla *figura 1*, risponde ai seguenti quesiti:

- Identificare i costrutti noti (creazione dello stack, eventuali cicli, costrutti)
- Ipotezzare il comportamento della funzionalità implementata.

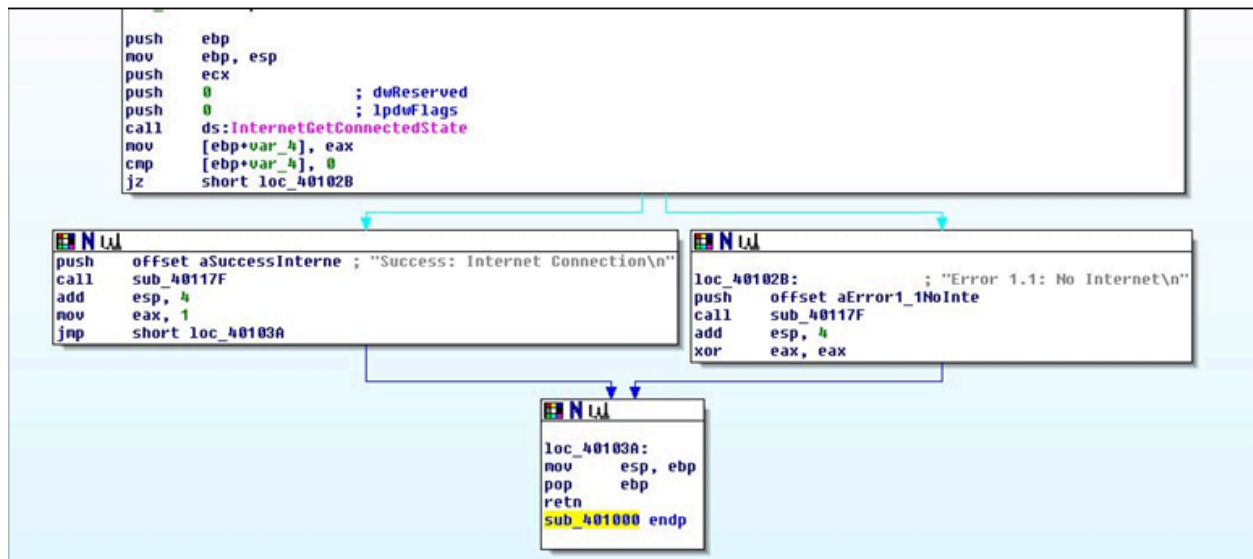


figura 1

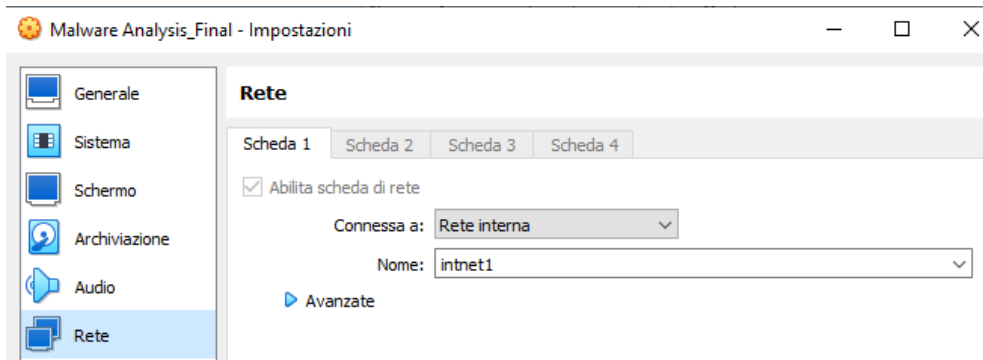
Operazioni preliminari

Prima di qualsiasi operazione, quando si tratta di malware analysis, è fondamentale configurare propriamente un ambiente di test.

Tra le buone pratiche da adottare per configurare un ambiente sicuro troviamo:

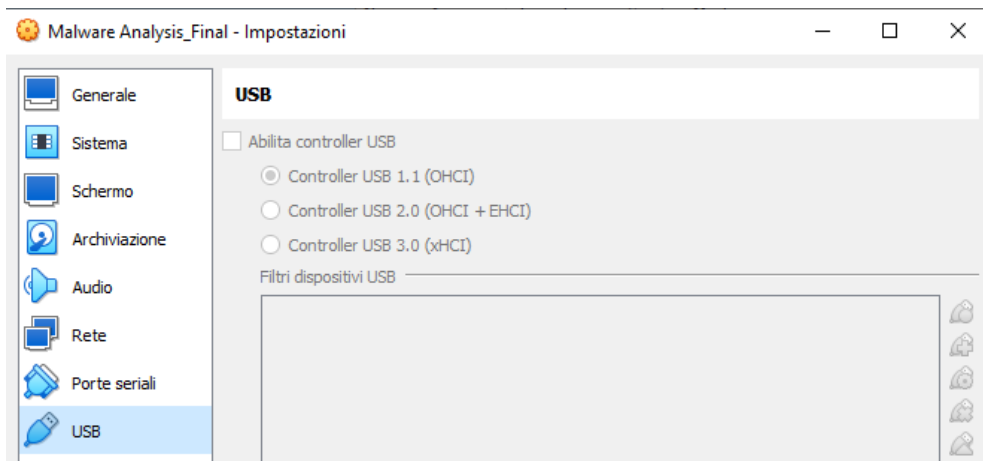
- **Configurazione schede di rete:**

la macchina sulla quale effettueremo il test non deve avere accesso diretto ad internet e preferibilmente nemmeno accesso ad altre macchine sulla rete, per questo si consiglia sempre di eliminare le interfacce di rete durante l'analisi statica e di abilitare un'interfaccia di rete interna per l'analisi dinamica. Questa impostazione è necessaria per monitorare il traffico che genera potenzialmente il malware.



- Dispositivi USB:

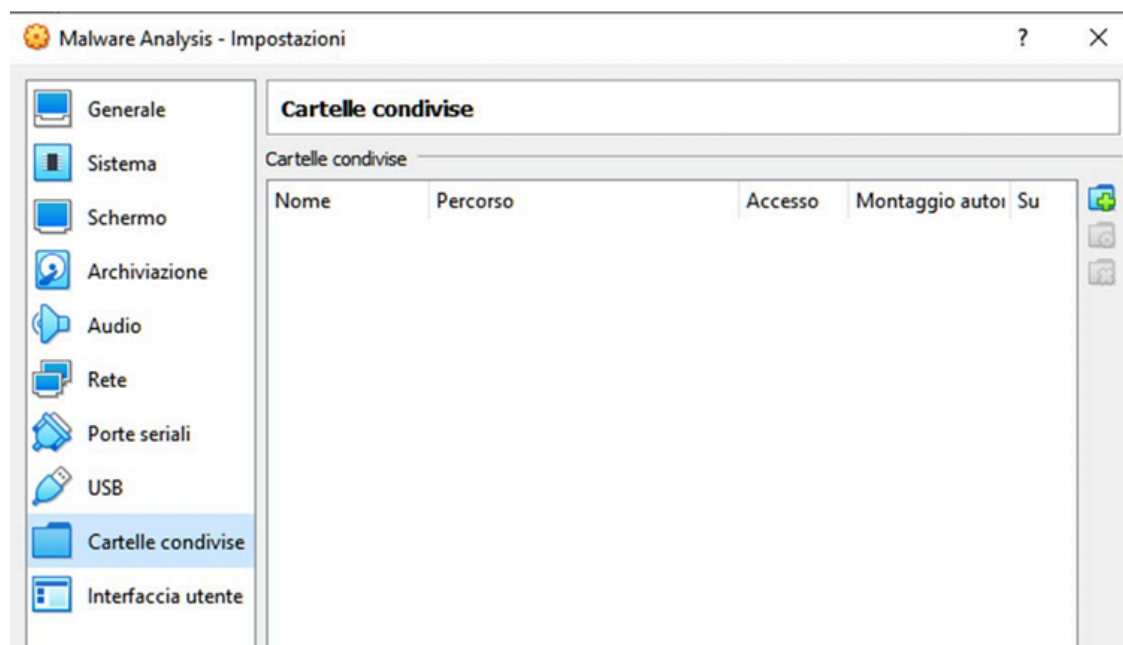
è buona pratica non abilitare o disabilitare il controller USB. Infatti il malware potrebbe utilizzare il dispositivo USB per propagarsi poi sulla macchina fisica.



- Cartelle condivise:

stesso discorso vale per le cartelle condivise tra la macchina ed il laboratorio virtuale. Potrebbero essere utilizzate dal malware per propagarsi al di fuori del laboratorio

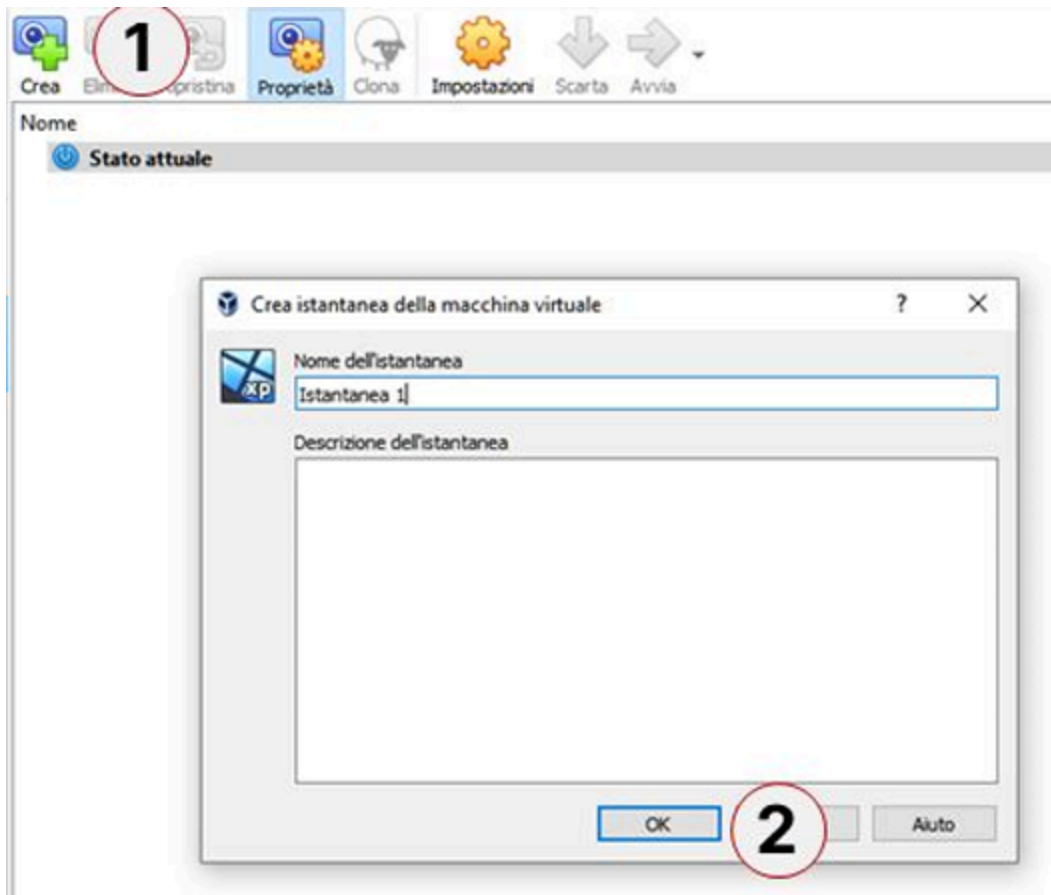
causando danni alla macchina e alle macchine sulla rete domestica. Sconsigliato condividere cartelle tra host e guest.



- Creazione di istantanee:

analizzando malware spesso capita di arrecare danno all'ambiente di test o di comprometterlo definitivamente. Una buona pratica è creare delle istantanee della macchina virtuale nel suo stato iniziale, prima di iniziare tutte le analisi, in modo tale da ripristinarlo qualora ce ne fosse bisogno.

Per creare un'istananea cliccare su <<crea>> e poi su OK dopo aver inserito un nome ed una descrizione facoltativa.



Malware analysis

L'analisi del malware o «malware analysis» è l'insieme di competenze e tecniche che permettono ad un analista della sicurezza informatica di indagare accuratamente un malware per studiare e capire esattamente il suo comportamento al fine di rimuoverlo dal sistema.

Durante lo studio dell'analisi dei malware incontreremo due tecniche principali di analisi:

- **Analisi statica**
- **Analisi dinamica**

Mentre **l'analisi dinamica** prevede **l'esecuzione del malware** in ambiente controllato, **l'analisi statica fornisce tecniche e strumenti per analizzare** il comportamento di un software malevolo **senza la necessità di eseguirlo**.

Le due tecniche sono tra di loro complementari, per un'analisi efficace i risultati delle analisi statiche devono essere poi confermate dai risultati delle analisi dinamiche.

Ma cos'è un malware?

Il termine malware nasce dalla combinazione delle parole **"malicious"** (malevolo) e **"software"** e si riferisce a qualsiasi tipo di software progettato per danneggiare, compromettere o alterare il funzionamento di un sistema informatico, di un dispositivo o di una rete, senza il consenso o la conoscenza da parte dell'utente. Un malware può assumere diverse forme e può essere progettato per svolgere una vasta gamma di attività dannose.

I tipi più comuni di malware sono:

- **Virus**

Si diffonde passando da computer a computer, senza azione diretta o autorizzazione da parte dei sistemi infetti. Si copiano in sezioni particolari all'interno del file system e i più sofisticati cercano di nascondersi dalle analisi dei vari sistemi di sicurezza (come antivirus o anti malware).

- **Trojan**

Un tipo di malware che si nasconde all'interno di un file apparentemente innocuo, come un documento office oppure un PDS. Si attiva quando la vittima apre il file. Tra i troja più comunemente utilizzati troviamo le backdoor, che sono generalmente utilizzate per fornire agli attaccanti delle shell sui sistemi infetti.

- **Rootkit**

Un malware progettato per nascondersi dagli utenti e dagli antivirus per prendere il controllo completo del sistema operativo. Un rootkit permette di mantenere privilegi elevati su una macchina senza essere notati.

- **Bootkit**

Sono dei rootkit che aggirano le protezioni del sistema operativo in quanto entrano in funzione prima dell'avvio completo del sistema operativo, in particolar modo prima dell'attivazione dei moduli di sicurezza di un sistema operativo.

- **Adware**

Sono dei programmi fastidiosi che mostrano pubblicità agli utenti di un pc.

- **Spyware**

Programmi che si usano per raccogliere informazioni sulle attività degli utenti di un sistema, ad esempio: il tipo di sistema operativo installato sulla macchina, i siti visitati, le password. Queste informazioni vengono inviate successivamente ad un server sotto il controllo dell'attaccante.

- **Dialer**

Un programma che cerca di chiamare numeri telefonici a pagamento per guadagnare soldi

- **Keylogger**

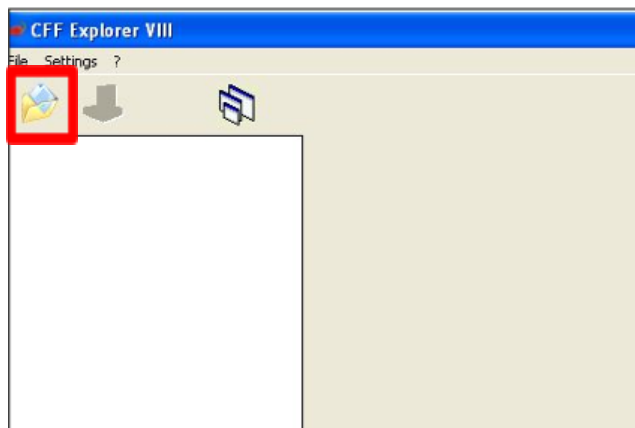
Programma che registra ogni tasto premuto sulla macchina della vittima. I keylogger registrano: i tasti premuti sulla tastiera, il nome delle finestre aperte dall'utente. Salvano poi queste informazioni in un file di log che spediscono ad un server controllato dall'attaccante.

CFF EXPLORER

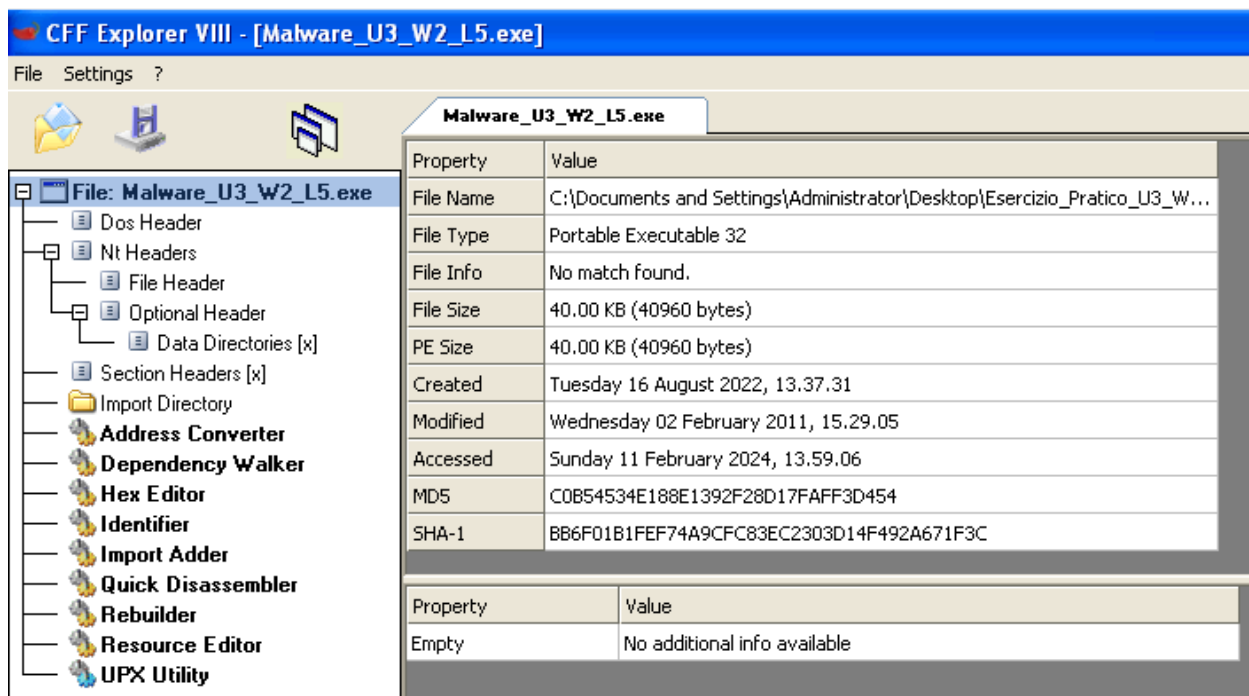
Per controllare le funzioni importate ed esportate da un malware, possiamo utilizzare il tool CFF Explorer, installato sulle macchine virtuali dedicate all'analisi dei malware.

CFF Explorer offre una vasta gamma di funzionalità, tra cui la visualizzazione delle sezioni del file, l'esplorazione delle directory di importazione ed esportazione, l'analisi delle risorse, il supporto per lo scripting, e molto altro. È uno strumento utile per gli sviluppatori e gli analisti di sicurezza.

Una volta aperto il programma dobbiamo scegliere un file eseguibile da caricare per analizzarlo.



Dopo aver caricato l'eseguibile scelto ci apparirà una schermata come la seguente, dove vengono riportate informazioni generali:



Per controllare le librerie e le funzioni importate ci spostiamo su **"import directory"** nel menù a sinistra.

Il pannello che spunterà alla destra ci darà informazioni sulle librerie importate

dall'eseguibile, mentre per ognuna delle librerie, il pannello inferiore ci mostrerà la lista delle funzioni richieste all'interno della libreria selezionata.

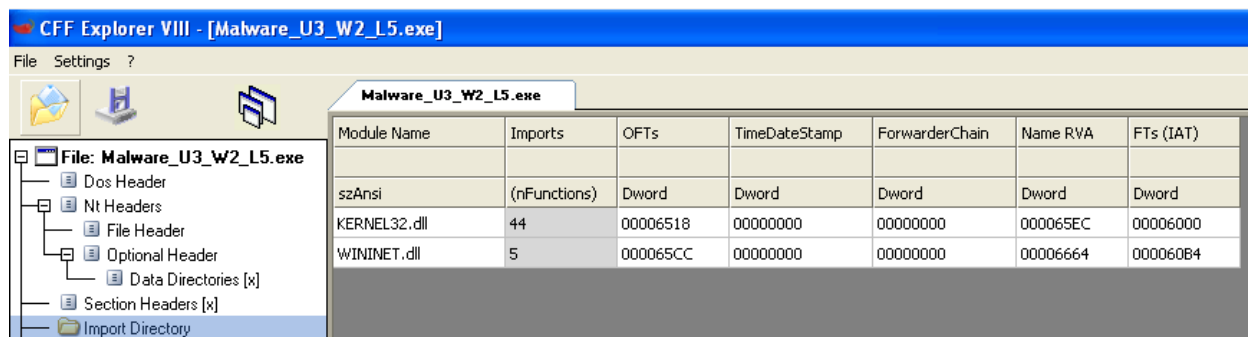


figura 2

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00002124	00002124	001B	CloseHandle
00002132	00002132	02B0	UnmapViewOfFile
00002144	00002144	01B5	IsBadReadPtr
00002154	00002154	01D6	MapViewOfFile
00002164	00002164	0035	CreateFileMappingA
0000217A	0000217A	0034	CreateFileA
00002188	00002188	0090	FindClose
00002194	00002194	0090	FindNextFileA
000021A4	000021A4	0094	FindFirstFileA
000021B6	000021B6	0028	CopyFileA

figura 3

Librerie importate

Possiamo rapidamente definire le librerie (anche chiamate moduli) come insieme di funzioni. Quando un programma ha bisogno di una funzione “chiama” una libreria al cui interno è definita la funzione necessaria.

Le librerie e le funzioni possono essere importate in tre modi diversi dagli eseguibili:

-
- **Importate staticamente:** ovvero l'eseguibile non fa altro che copiare tutto il contenuto della libreria all'interno del proprio codice. Da un punto di vista pratico, questo approccio incrementa la dimensione di un file. Dal punto di vista dell'analista, risulta più complicato invece distinguere in fase di analisi statica avanzata il codice della libreria dal codice dell'eseguibile.
 - **Importate a tempo di esecuzione (runtime):** in questa casistica l'eseguibile richiama la libreria solamente quando necessita di una particolare funzione. Questo comportamento è ampiamente utilizzato dai malware, che «chiamano» una determinata funzione solo all'occorrenza per risultare quanto meno invasivi e rilevabili possibile. Per chiamare la libreria all'occorrenza si utilizzano delle funzioni messe a disposizione dal sistema operativo come «LoadLibrary» e «GetProcAddress».
 - **Importate dinamicamente:** questa è la casistica più interessante per gli analisti di sicurezza ed anche la più comune. Le librerie importate dinamicamente vengono caricate dal sistema operativo quando l'eseguibile è avviato. Quando necessario, la funzione viene chiamata ed eseguita all'interno della libreria.

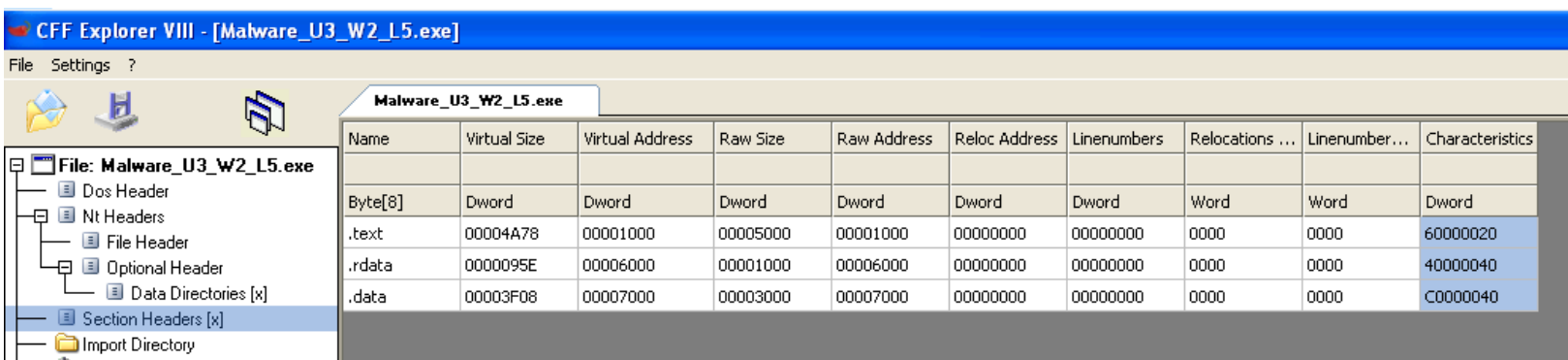
Nel caso preso in esame, il malware da noi analizzato, abbiamo a che fare con due librerie diverse:

- **Kernel32.dll:** libreria piuttosto comune che contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria.
- **Wininet.dll:** libreria che contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP.

Sezioni

Nell'analisi statica di un malware la "sezione" fa riferimento a una parte specifica del file eseguibile o del codice sorgente del malware che viene esaminato. Le sezioni possono contenere diverse informazioni, come il codice eseguibile, i dati, le risorse, le tabelle di importazione ed esportazione e altro ancora.

Ogni sezione ha un preciso scopo, e conoscerle è una preziosa informazione per le analisi. Per controllare quelle che compongono il malware che stiamo analizzando ci spostiamo nel menù “**Section Headers**” sul pannello a sinistra di CFF Explorer.



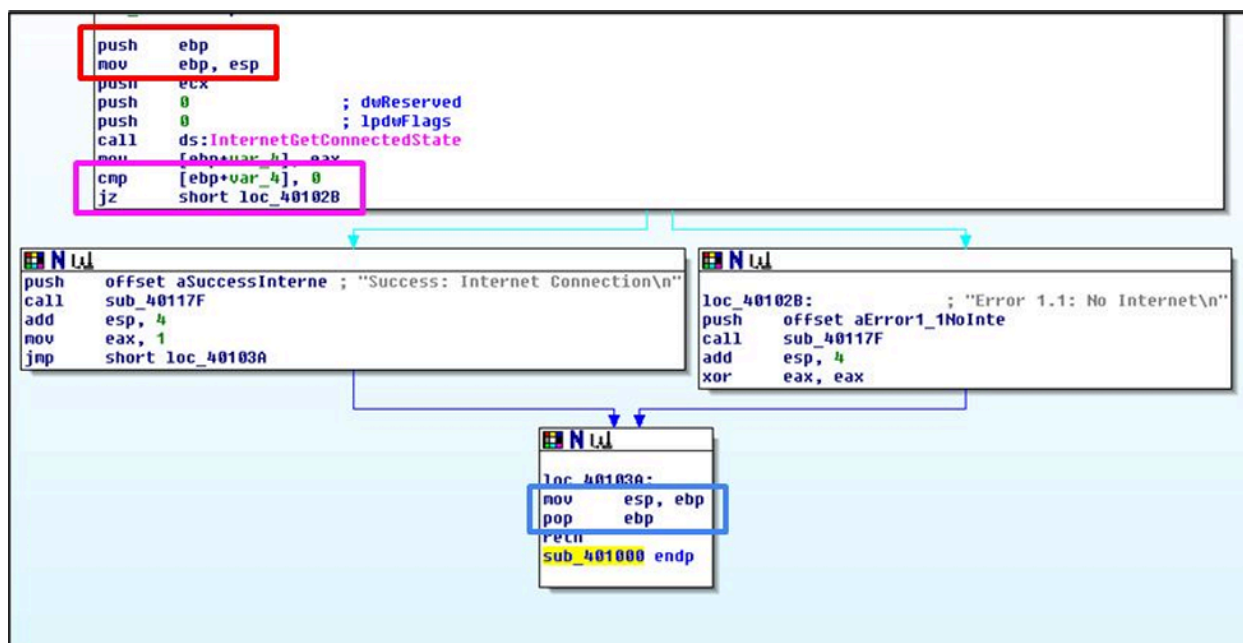
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000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

Le sezioni presenti sono:

- **.text** che contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
- **.rdata** che include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile, informazione che come abbiamo visto possiamo ricavare con CFF Explorer.
- **.data** che contiene tipicamente i dati/le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma. Ricordiamo che una variabile si dice globale quando non è definita all'interno di un contesto di una funzione, ma bensì è globalmente dichiarata ed è di conseguenza accessibile da qualsiasi funzione dell'eseguibile.

Costrutti noti

Riportiamo la *figura 1* evidenziando i costrutti a noi noti:



Quelli che vediamo nella figura sopra sono costrutti in codice **assembly**.

Assembly

L'analisi statica avanzata presuppone la conoscenza di un particolare tipo di linguaggio, chiamato linguaggio Assembly. Il linguaggio Assembly è univoco per una data architettura di un PC, ma cambia da architettura ad architettura.

L'analista di sicurezza durante l'analisi statica utilizzerà dei tool chiamati «Disassembler» che sono programmati per tradurre le istruzioni binarie eseguite dalla CPU in istruzioni più leggibili dall'uomo, che è proprio il linguaggio Assembly.

La conoscenza del linguaggio Assembly servirà per "leggere" le istruzioni eseguite dalla CPU in formato leggibile dall'uomo.

Come detto in precedenza Assembly è un linguaggio che dipende dall'architettura del calcolatore, tra le più note troviamo x86, 64, ARM, MIPS e PowerPC.

Quello che ci interessa sapere è che i processori possono essere a 32 o 64 bit, cioè

l'ammontare massimo di informazioni che la CPU riesce a gestire per ogni singola operazione.

Per lo scopo del nostro progetto vedremo l'Assembly per i set di istruzioni x86, ovvero per i processori a 32 bit.

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

- **un codice mnemonico**, ovvero una parola che identifica l'istruzione da eseguire
- **uno o più operandi**, che identificano le variabili o la memoria oggetto dell'istruzione.

Si possono utilizzare tre tipi diversi di operandi:

- **un valore**, come ad esempio un numero. Attenzione, generalmente i valori immediati non sono scritti in formato decimale ma bensì in formato esadecimale nella forma **0xYY** dove **YY** rappresenta la versione esadecimale del numero decimale
- **uno dei registri** messi a disposizione della CPU
- **un indirizzo di memoria** che contiene un valore di interesse

Un **registro** è un tipo di memoria a rapido accesso (contenute nelle dimensioni ma che hanno una velocità di accesso maggiore rispetto ad altre memorie, come quelle secondarie) che consente di salvare temporaneamente una variabile che deve essere utilizzata dalla CPU.

Costrutto 1 (riquadro rosso)

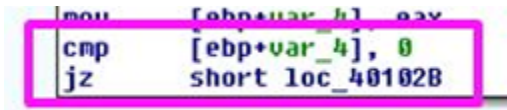


```
push    ebp
mov     ebp, esp
push    ecx
```

Le istruzioni "**push ebp**" e "**mov ebp, esp**" nel linguaggio assembly x86 sono utilizzate all'inizio di una funzione per impostare un "contesto" per quella funzione. Questo contesto include la creazione di uno stack frame per la funzione, che contiene informazioni come variabili locali e parametri della funzione. In particolare, "**push ebp**" salva il valore corrente del registro base nello stack, mentre "**mov ebp, esp**" imposta il registro base allo stesso

valore del puntatore dello stack (ESP). Questo permette di accedere agevolmente alle variabili locali e ai parametri della funzione utilizzando il registro base come punto di riferimento all'interno dello stack.

Costrutto 2 (riquadro rosa)



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

Questa frazione di codice assembly x86 effettua un confronto (cmp) tra il valore memorizzato all'indirizzo di memoria [ebp+var_4] e il valore 0. Il risultato del confronto non viene utilizzato direttamente, ma piuttosto si utilizza un'istruzione di salto condizionato (jz) che esegue un salto a un'etichetta specificata (loc_40102B) solo se il risultato del confronto è zero, altrimenti continua l'esecuzione in modo lineare.

Costrutto 3 (riquadro blu)



```
loc_40103A:  
mov     esp, ebp  
pop     ebp  
ret     4  
sub_401000 endp
```

Le istruzioni "**mov esp, ebp**" e "**pop ebp**" nel linguaggio assembly x86 sono utilizzate alla fine di una funzione per ripristinare lo stato dello stack al suo valore precedente prima dell'esecuzione della funzione stessa. La prima istruzione ripristina il puntatore dello stack (ESP) al suo valore precedente, che era stato salvato nel registro base (EBP) all'inizio della funzione. La seconda istruzione estrae il valore corrente dallo stack e lo carica nel registro base (EBP), ripristinando così il valore originale del registro base. In sintesi, queste istruzioni consentono di "pulire" il frame dello stack creato per la funzione, assicurando che lo stack ritorni al suo stato precedente all'esecuzione della funzione.