

Report S10-L5

Analisi statica e dinamica: Un approccio pratico

Traccia.....	1
Abstract	2
Introduzione Teorica Generale	2
Malware e Malware Analysis.....	2
1°task – Analisi statica basica dell’eseguiabile di un malware.....	4
Formato PE del file del Malware	4
Ambiente di lavoro e tool	5
Svolgimento	5
Librerie importate.....	7
Sezioni dell’eseguiabile.....	9
2°task – Analisi statica avanzata.....	10
Assembly.....	10
Svolgimento	11
Conclusioni	13

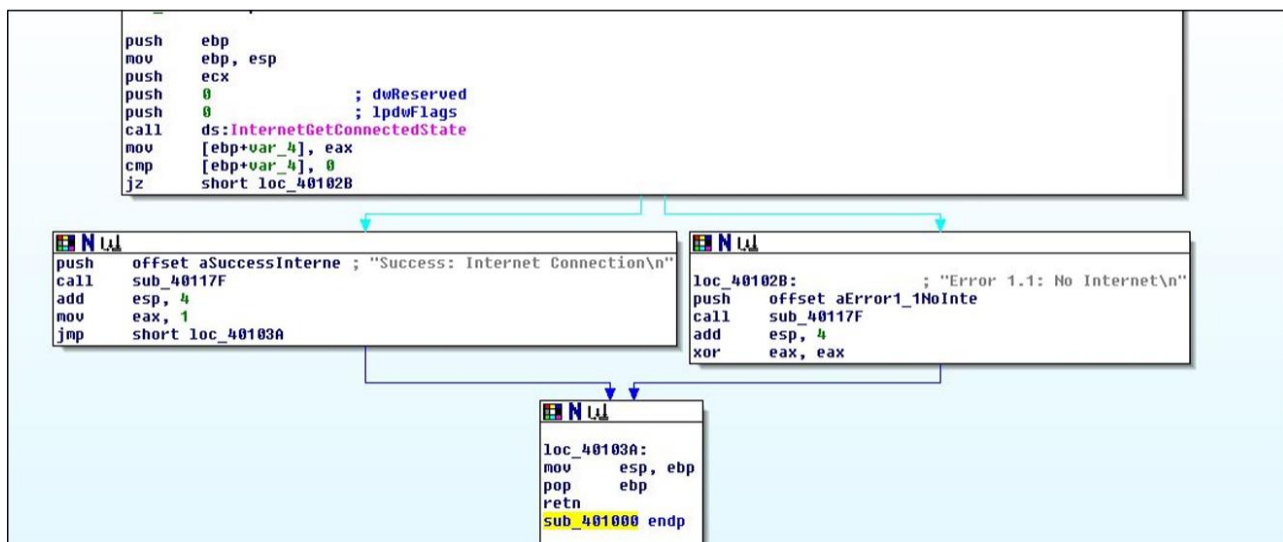
Traccia

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 di seguito, rispondere ai seguenti quesiti:

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



Abstract

Le due esercitazioni di Malware Analysis, svolte e poi dettagliate nel presente report, consistono nell'analisi statica basica dell'eseguibile di un malware, condotta mediante il tool CFF Explorer, e nell'analisi statica avanzata del codice Assembly di quello che sembra essere lo stesso malware.

Per consentire al lettore la comprensione delle singole operazioni, si è deciso di strutturare il presente scritto fornendo un'introduzione teorica generale relativa alla Malware Analysis, con spiegazione dei concetti chiave che si incontrano nel corso delle esercitazioni, e di svolgere in sequenza le due esercitazioni, fornendo, preliminarmente, un'introduzione teorica specifica per contestualizzare più nel dettaglio le attività pratica svolte.

Infine, si sono riportate delle brevi conclusioni che ripercorrono a grandi linee il contenuto del report con il tentativo, sorretto dalle evidenze riscontrate, di profilazione del malware in analisi.

Introduzione Teorica Generale

Uno degli scopi fondamentali delle Security Operations nell'ambito enterprise è prevenire il verificarsi di attacchi informatici, ponendo in essere misure e azioni preventive di sicurezza.

Malgrado l'implementazione delle azioni preventive, c'è la probabilità che un incidente di sicurezza possa verificarsi.

Per questo motivo, le aziende mettono in pratica piani di Business Continuity e Disaster Recovery per assicurare la continuità dei servizi critici ai propri utenti.

L'incidente di sicurezza in sé viene gestito tramite le procedure di «incident response», dove il CSIRT, team dedicato, mette in campo le azioni per contrastare la propagazione dell'incidente.

Ad oggi, la maggior parte degli incidenti di sicurezza è dovuta ad attacchi dall'esterno e, in particolar modo, ad attacchi malware.

Malware e Malware Analysis

I **Malware** (malicious software) includono una vasta gamma di **programmi scritti per arrecare danno a sistemi informatici**, spesso a scopo di lucro.

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.

Queste competenze sono fondamentali per i membri tecnici del CSIRT durante la risposta agli incidenti di sicurezza.

Sono due le tecniche principali di analisi:

- **Analisi statica:** comporta l'esame del codice sorgente o del file eseguibile del malware senza eseguirlo.
- **Analisi dinamica:** presuppone l'esecuzione del malware in ambiente controllato, come una sandbox, per monitorare il suo comportamento in tempo reale.

Mentre l'analisi dinamica presuppone 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.

Entrambe le tecniche si dividono in «**basica**» e «**avanzata**», pertanto si hanno:

- **Analisi statica basica**

L'analisi statica basica consiste nell'***esaminare un eseguibile senza vedere le istruzioni che lo compongono.***

Lo **scopo** dell'analisi basica statica è di confermare se un dato file è malevolo e fornire informazioni generiche circa le sue funzionalità.

L'analisi statica basica è sicuramente la **più intuitiva** e **semplice** da mettere in pratica, **ma** risulta anche essere la **più inefficiente** soprattutto contro malware sofisticati.

- **Analisi statica avanzata**

L'analisi statica avanzata presuppone la ***conoscenza dei fondamenti di «reverse-engineering»*** al fine di identificare il comportamento di un malware a partire dall'analisi delle istruzioni che lo compongono.

In questa fase vengono utilizzati dei **tool** chiamati «**disassembler**» che ricevono in input un file eseguibile e restituiscono in output il linguaggio «Assembly».

- **Analisi dinamica basica**

L'analisi dinamica basica presuppone l'***esecuzione del malware in modo tale da osservare il suo comportamento sul sistema infetto al fine di rimuovere l'infezione.***

I malware devono essere eseguiti in ambiente sicuro e controllato, come una sandbox, in modo tale da eliminare ogni rischio di arrecare danno a sistemi o all'intera rete.

Così come per l'analisi statica basica, l'analisi dinamica basica è piuttosto semplice da mettere in pratica, ma **non è molto efficace** quando ci si trova ad analizzare **malware sofisticati**.

- **Analisi dinamica avanzata**

L'analisi dinamica avanzata presuppone la **conoscenza dei debugger** per esaminare lo stato di un programma durante l'esecuzione.

Lo studio e la comprensione del comportamento esatto di un malware è un compito piuttosto complicato.

Tuttavia, **può essere semplificato identificando il tipo di malware** che si sta analizzando.

Un esempio, è quello di un malware che si mette in ascolto su una porta TCP e garantisce una shell a chi si connette, il che fa presumere che si tratti di una backdoor.

Allo stesso modo, un malware che contatta un dominio per scaricare un altro file eseguibile, potrebbe essere un «downloader», ovvero un malware che scarica altri malware.

Capire preventivamente il tipo di malware da alcuni caratteri generali può, dunque, aiutare nella comprensione del comportamento generale.

1°task – Analisi statica basica dell'eseguibile di un malware

Si è anticipato nell'abstract che la prima parte della traccia, richiedendo di individuare le librerie importate dall'eseguibile di un malware e le sezioni che lo compongono, comporta lo svolgimento di un'analisi statica basica del malware.

L'**analisi statica basica** è quella metodologia di analisi che, *senza eseguirlo, analizza un file o il codice sorgente di un malware passivamente, per esempio la struttura* come l'header o le stringhe incorporate, al fine di comprendere se si tratti effettivamente di un software dannoso o meno, e, in caso di risposta positiva, fornire informazioni generiche sulle funzionalità dello stesso.

È la tipologia di analisi più intuitiva ma più inefficace perché, non coinvolgendo la visione delle istruzioni del codice Assembly (statica avanzata) o l'esecuzione del malware (analisi dinamica in generale) con l'uso di tool specifici, non è in grado di profilare malware sofisticati, progettati per eludere le rilevazioni passive nascondendo le librerie importate (chiamate solo all'occorrenza e durante l'esecuzione) o oscurando le sezioni.

Nel caso in esame, la traccia indica come file eseguibile da analizzare staticamente il file eseguibile "**Malware_U3_W2_L5**", di cui si devono individuare le librerie importate e le sezioni che lo compongono.

Si parte spiegando che il primo passo dell'analisi statica consiste nell'individuare il formato dell'eseguibile, che è differente a seconda del sistema operativo.

Poiché l'esercitazione è svolta su una macchina Windows, il formato dell'eseguibile è PE.

Formato PE del file del Malware

Il formato PE dei file del malware, come altri file legittimi di Windows, al suo interno contiene delle **informazioni** necessarie al sistema operativo *per capire come gestire il codice del file*.

In particolare, nell'**header** del formato PE possono essere visualizzate:

- Le **librerie importate** e ed **esportate** (condivise) dal malware, con le relative **funzioni**
- Le **sezioni** di cui si compone il software

Librerie

Le librerie (anche chiamate moduli) sono insieme di funzioni predefinite pronte per l'uso, senza doverle riscrivere ogni volta.

Quando un programma ha bisogno di una funzione «chiama» una libreria al cui interno è definita la funzione necessaria.

Si dice anche che il programma ha **importato una libreria**.

Gli eseguibili, come il nostro malware, possono chiamare le librerie in tre modi diversi:

- **Importazione statica:** in cui non fanno altro che ***copiare tutto il contenuto della libreria*** all'interno del proprio codice.

Questa importazione aumenta la dimensione del file ma è quello che rende difficile, per gli analisti di sicurezza, distinguere il codice della libreria dal codice del programma.

- **Importazione a tempo di esecuzione (Runtime):** in questa casistica *gli eseguibili richiamano la libreria solamente quando necessitano di una particolare funzione.*

Questo comportamento è **ampiamente utilizzato dai malware**, che importano una determinata funzione solo all'occorrenza, **per risultare quanto meno invasivi e rilevabili possibile.**

Per importare la libreria all'occorrenza si utilizzano delle funzioni messe a disposizione dal sistema operativo come «**LoadLibrary**» e «**GetProcAddress**».

- **Importazione dinamica:** in cui le *librerie* importate dinamicamente vengono **caricate dal sistema operativo quando l'eseguibile è avviato** e, quando necessario, la funzione viene chiamata ed eseguita all'interno della libreria.

Si tratta della casistica più interessante per gli analisti di sicurezza ed anche la **più comune.**

Quindi, controllando l'header del PE dell'eseguibile di un malware, è **possibile sapere quali librerie e funzioni sono importate**, il che è fondamentale **per capire lo scopo dei malware.**

Ambiente di lavoro e tool

L'analisi statica basica è condotta sulla **macchina virtuale «Malware Analysis»**, la quale, una volta scaricata in formato .ova, va semplicemente importata in Oracle VirtualBox (è sufficiente cliccare due volte sul file .ova della VM per determinarne l'importazione).

NB: La macchina **deve essere configurata in modalità di rete interna** affinché non abbia connessioni con il mondo esterno, in quanto, essendo utilizzata come **macchina di test per lo studio dei malware**, contiene al suo interno eseguibili dannosi che, se non gestiti correttamente, creano danni importanti. **Ovviamente, nel caso dell'analisi statica basica non si esegue il file del malware, ma il seguente consiglio sarà valido per le analisi dinamiche.**

Per l'analisi statica basica dell'eseguibile si è utilizzato il tool CFF Explorer, che, installato sulla VM, è un programma utilizzato per la **visualizzazione, l'analisi e la modifica della struttura e del contenuto dei file eseguibili**, quali i file formato PE.

In particolare, CFF Explorer consente di **individuare le librerie** importate ed esportate dall'eseguibile, con le relative **funzioni** e **le sezioni di un file eseguibile.**

Svolgimento

La richiesta della traccia è di recuperare informazioni su un file eseguibile tramite l'analisi statica basica. In particolare, in relazione al file eseguibile **Malware_U3_W2_L5**, contenuto nella cartella del desktop «**Esercizio_Pratico_U3_W2_L5**» della VM «Malware Analysis», si chiede di indicare le librerie importate dal malware e le sezioni di cui si compone.

Per ottenere queste informazioni si utilizza il tool CFF Explorer, di analisi dei file eseguibili, preinstallato sulla VM, di cui si riportano i passaggi effettuati.

Una volta cliccato sull'icona rossa del tool nel desktop della macchina, nella schermata che si apre, si ricerca il file eseguibile di cui si vuole analizzare il contenuto: si seleziona la prima icona a sinistra e, nel desktop, la cartella «**Esercizio_Pratico_U3_W2_L5**» (**figura 1**). In questo modo, con un semplice doppio click, si imposta il file "**Malware_U3_W2_L5**" come eseguibile da analizzare con il tool (**figura 2**).

Figura 1

Malware Analysis_Final (Istantanea 1) [In esecuzione] - Oracle VM VirtualBox

File Macchina Visualizza Inserimento Dispositivi Aiuto

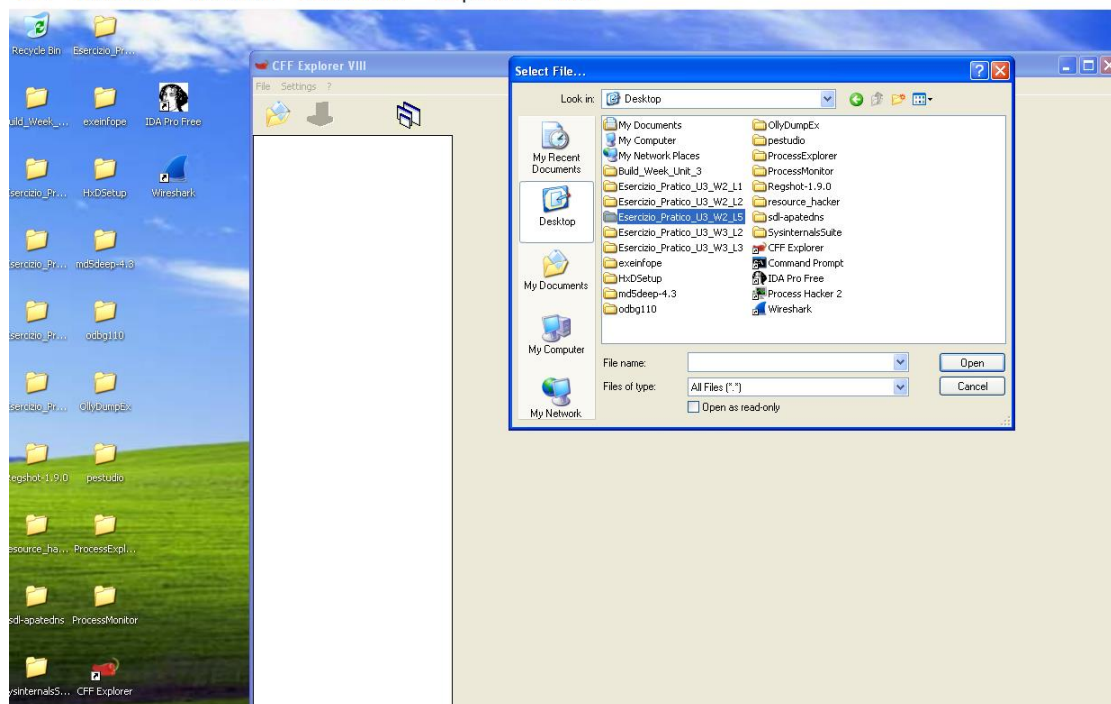
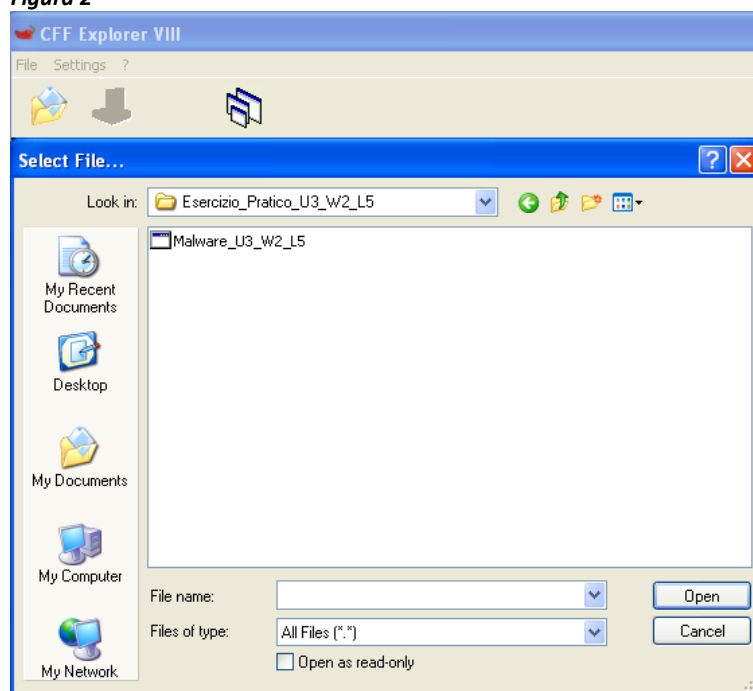


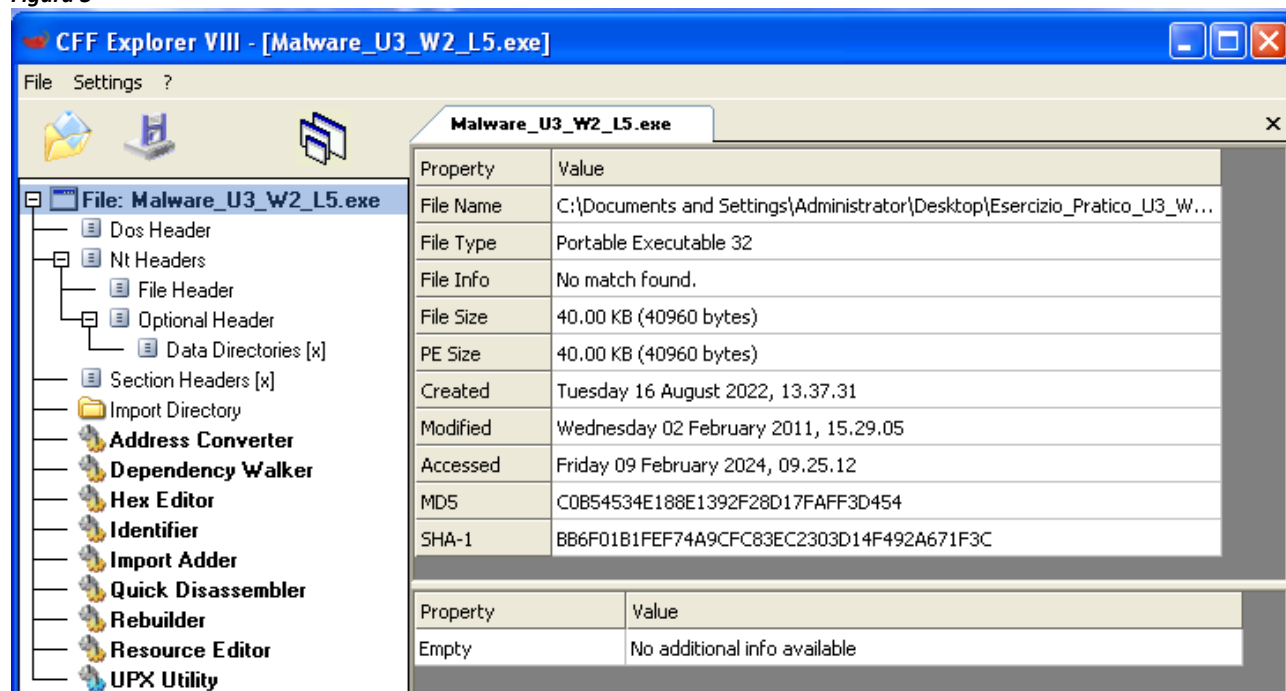
Figura 2



Con il doppio click, si apre la schermata iniziale di CFF Explorer relativa al file eseguibile selezionato. Come si può vedere, sulla sinistra è presente un menù con varie opzioni per visualizzare e analizzare la struttura del file eseguibile, formato .exe (comunemente associato ai file PE).

Si deve selezionare l'opzione **"Import Directory"**, per vedere le librerie che il malware ha importato.

Figura 3

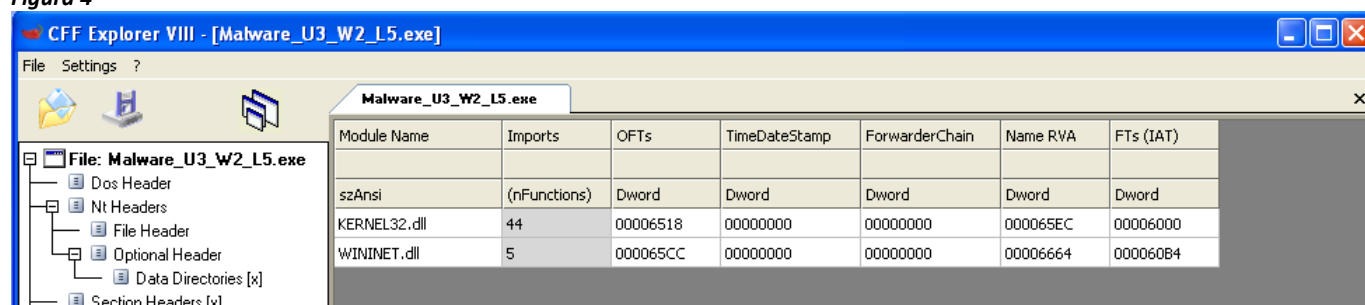


Librerie importate

In questo modo, si può vedere che l'eseguibile in oggetto ha importato due librerie (figura 4):

- **KERNEL32.dll**: è una delle principali librerie di sistema di Windows utilizzata per effettuare operazioni di base del sistema operativo. Questa libreria offre funzioni di basso livello, come la gestione della memoria, operazioni di input/output, manipolazione dei file, creazione e gestione di processi e thread e molte altre.
- **WININET.dll**: è una libreria che fornisce funzioni di alto livello per l'accesso a protocolli Internet come HTTP e FTP. Con funzionalità che includono il supporto per navigazione web, gestione di richieste HTTP e FTP, manipolazione dei cookie e accesso a risorse online, WININET è spesso utilizzata per la comunicazione di rete, inclusa la possibilità di scaricare o caricare dati da e verso Internet.

Figura 4



Selezionando le due librerie, KERNEL32.dll (figura 5) e WININET.dll (figura 6), tramite un pannello inferiore si possono vedere le funzioni che sono incorporate nelle due librerie.

Di particolare interesse (figura 5) è che KERNEL32.dll, fra le tante funzioni, ne presenta due sospette: **Load Library** e **GetProcAddress**.

Infatti, queste due funzioni vengono utilizzate dai malware per effettuare l'**importazione** di librerie a **tempo di esecuzione o runtime**, ovvero richiamando librerie all'occorrenza, solamente quando hanno necessità di una particolare funzione.

Questa importazione rivela non solo che **il file potrebbe essere un malware che intende essere il meno invasivo e rilevabile possibile**, ma anche che, con l'analisi statica basica, non si possono sapere quali librerie il malware importi per i suoi scopi malevoli.

Figura 5

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
000065B0	00006098	0000688E	00006890
Dword	Dword	Word	szAnsi
0000684E	0000684E	00BF	GetCPInfo
0000685A	0000685A	00B9	GetACP
00006864	00006864	0131	GetOEMCP
00006870	00006870	02BB	VirtualAlloc
00006880	00006880	01A2	HeapReAlloc
0000688E	0000688E	013E	GetProcAddress
000068A0	000068A0	01C2	LoadLibraryA

Figura 6

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

Per visualizzare le sezioni del file eseguibile ci si sposta nella "Section Headers" del menù a sinistra dell'interfaccia principale.

Le sezioni che compongono l'eseguibile in esame sono tre (**figura 7**):

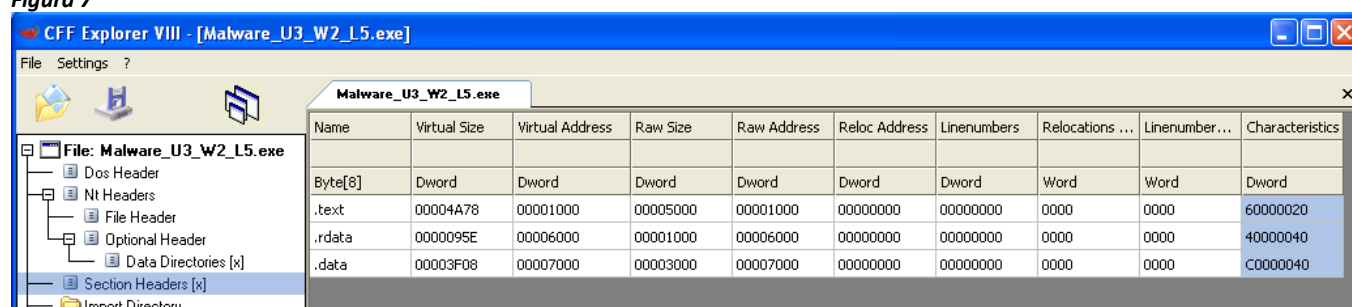
- **.text**: la sezione «text» contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato. Generalmente, questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
- **.rdata**: la sezione «rdata» include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile.
- **.data**: la sezione «data» contiene tipicamente i dati/le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma.

Osservando le sezioni tramite il pannello di controllo si può notare che nell'Ascii della sezione ".data" sono presenti dei messaggi particolari:

1. **Error .1.1 no Internet**: messaggio di errore nel tentativo di stabilire una connessione a Internet.
2. **Success Internet connection**: messaggio di successo nello stabilire una connessione a Internet.
3. **Error 2.3 Fail to get Command**: errore nel tentativo di ottenere comandi o istruzioni. Questo potrebbe essere un passo cruciale per il malware per ricevere nuove istruzioni o aggiornamenti dal server di comando e controllo.
4. **Error 2.2 Fail to read file**: errore nella lettura di file. Questo potrebbe indicare che il l'eseguibile cerca di eseguire operazioni specifiche sui file presenti sul sistema compromesso.
5. **Error 2.1. Fail to openUrl http://www.practicalmalwareanalysis.com/cc.htm**: messaggio di fallimento del tentativo di aprire l'URL <http://www.practicalmalwareanalysis.com/cc.htm>.

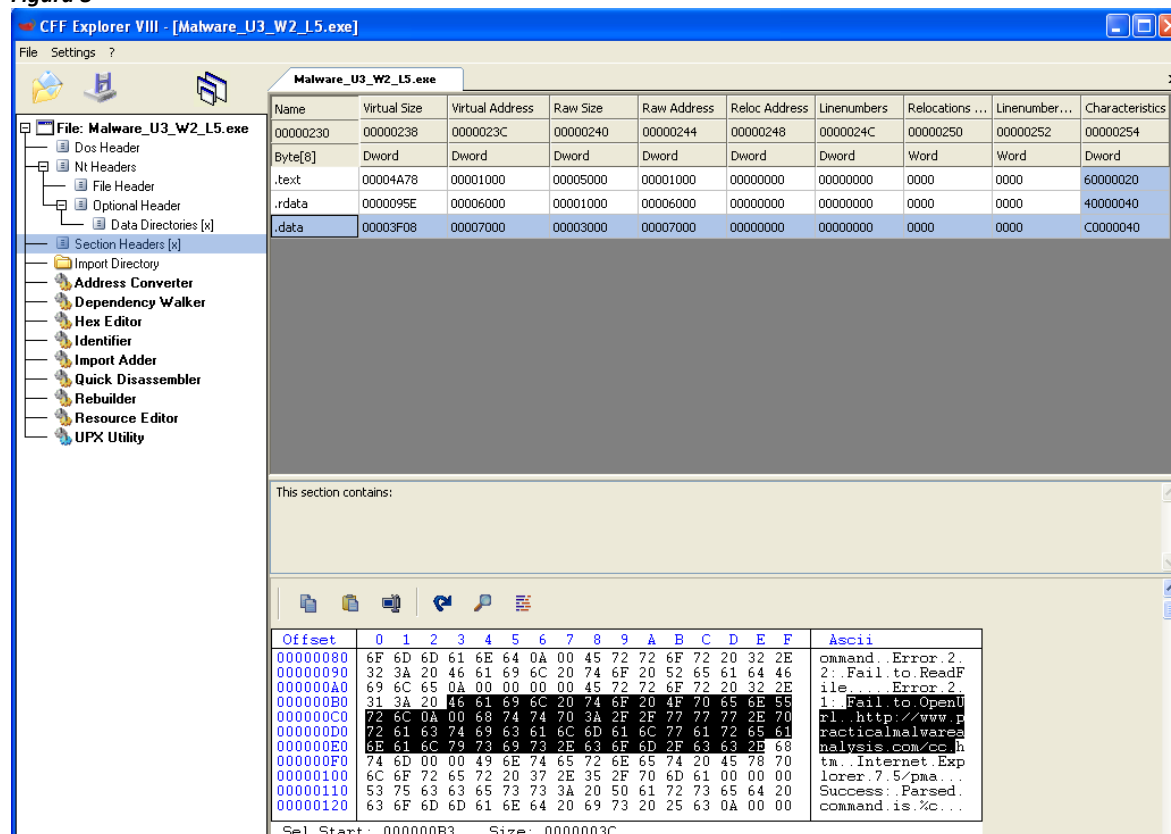
Da questi messaggi si può ipotizzare che il file eseguibile, anche in base alle librerie viste precedentemente, cerchi di manipolare il file system della VM (fail to read fail, KERNEL32.dll) per estrapolare informazioni sul sistema e stia, allo stesso tempo, cercando di valutare se la macchina abbia accesso ad internet (informazioni sulla rete) per tentare di connettersi all'URL <http://www.practicalmalwareanalysis.com/cc.htm>.

Figura 7



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

Figura 8



2°task – Analisi statica avanzata

La seconda parte della traccia, fornisce sezioni del codice in assembly dello stesso malware, analizzato in modo basilare precedentemente.

Richiedendo di individuare costrutti noti e di ipotizzare il comportamento della funzionalità implementata, ciò comporta lo svolgimento di un'analisi statica avanzata.

L'**analisi statica avanzata** presuppone la conoscenza di un particolare tipo di linguaggio, chiamato linguaggio Assembly, il quale non è univoco ma cambia da architettura ad architettura dei processori. Innanzitutto, gli analisti di sicurezza utilizzano i **Disassembler**, tool che sono programmati per tradurre le istruzioni binarie eseguite dalla CPU in istruzioni più leggibili dall'uomo, che è proprio il linguaggio Assembly.

Dunque, la conoscenza del linguaggio Assembly servirà per «leggere» le istruzioni eseguite dalla CPU in formato leggibile dall'uomo. In particolare, gli analisti analizzano il codice assembly di un file per rintracciare costrutti, funzioni e istruzioni, come quelli di C o C++, che spieghino il funzionamento del malware.

Assembly

Quindi, Assembly è il linguaggio di programmazione di basso livello vicino al linguaggio macchina e specifico per ciascuna architettura hardware.

Composto da un insieme di istruzioni elementari (**Istruzioni elementari**), il linguaggio Assembly è eseguito direttamente dalla CPU dopo una conversione in linguaggio macchina, il linguaggio binario comprensibile direttamente dalla macchina (**traduzione diretta**).

Quando un programma scritto in linguaggio assembly viene eseguito, le istruzioni assembly vengono convertite in linguaggio macchina, il linguaggio binario comprensibile direttamente dalla CPU.

La **CPU** (Unità di Elaborazione Centrale) interpreta e esegue queste istruzioni in modo sequenziale, manipolando i dati e controllando il flusso del programma (**Esecuzione sequenziale**).

Quindi, ogni istruzione Assembly, eseguita direttamente dalla CPU, svolge un'operazione, e il complesso delle operazioni svolte dall'insieme delle istruzioni compone un programma completo.

In sostanza, il linguaggio assembly agisce come **un'interfaccia o un ponte tra il linguaggio di alto livello** (più comprensibile agli sviluppatori) **e il linguaggio macchina della CPU** (più comprensibile alla macchina stessa), offrendo un modo comprensibile, per gli sviluppatori, di scrivere o leggere le istruzioni, tradotte direttamente in istruzioni binarie eseguibili dalla CPU.

Gli sviluppatori scrivono in linguaggio assembly per avere un maggiore controllo sulle operazioni effettuate dalla CPU, spesso per ottimizzare le prestazioni o implementare funzionalità specifiche a un livello di dettaglio più fine.

Infatti, l'**accesso diretto all'hardware** offerto dal linguaggio Assembly consente agli sviluppatori di gestire direttamente la memoria, i registri e altri aspetti dell'architettura del processore, garantendo un controllo dettagliato sulle risorse del computer.

Istruzioni elementari: Le istruzioni elementari offerte da Assembly sono comandi di base che corrispondono direttamente alle operazioni eseguite dalla CPU di un computer.

Queste istruzioni sono composte da due parti:

- Un codice mnemonico che identifica l'istruzione da eseguire
- Uno o più operandi (che può essere un valore numerico in esadecimale, un registro o un indirizzo di memoria).

Registri e memoria: Il processore di un computer è dotato di registri interni e di una memoria, elementi essenziali per l'esecuzione delle istruzioni assembly.

I **registri** sono piccole aree di memoria integrate direttamente nel processore, e per tale motivo caratterizzate da un accesso estremamente veloce.

Essi offrono l'archiviazione temporanea dei dati, solitamente variabili, che vengono utilizzati dalla CPU durante l'esecuzione delle istruzioni.

Le istruzioni assembly operano spesso direttamente su questi registri e sulla memoria al fine di eseguire **operazioni di spostamento di dati** (mov), **aritmetiche** (tra cui somme, sottrazioni), **logiche** (xor, or e and), **di controllo del flusso** (cmp e jmp o jz, jnz etc), **di gestione dei registri e delle memorie** (push e pop) e **di chiamata di funzione** (call).

Ad esempio, l'istruzione assembly potrebbe richiedere il caricamento di un valore dalla memoria in un registro, l'effettuazione di una somma tra due registri e infine, la memorizzazione del risultato in un'altra locazione di memoria.

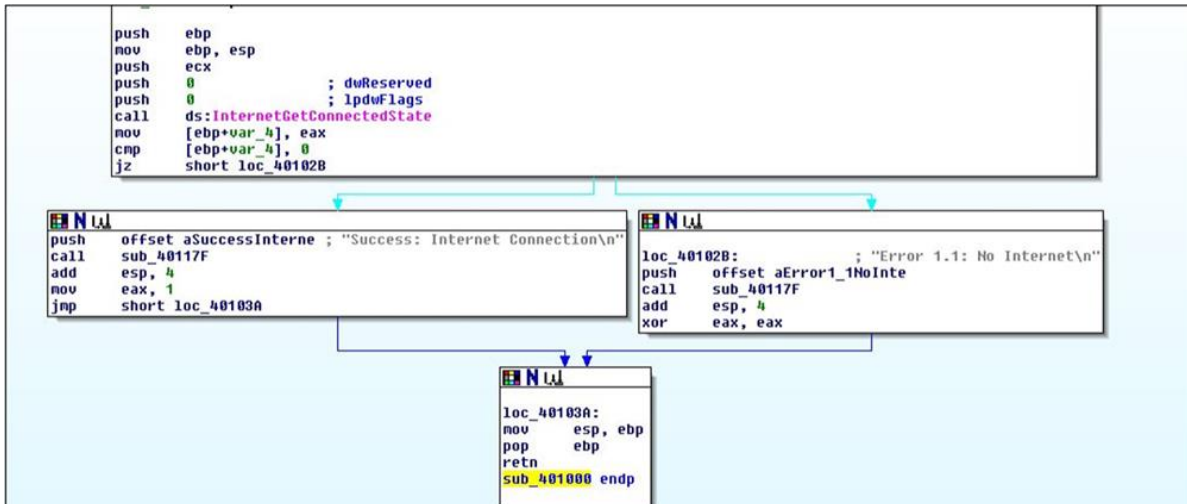
Questo approccio è cruciale per garantire un'esecuzione efficiente delle istruzioni, poiché i registri consentono di minimizzare il tempo impiegato per l'accesso ai dati.

Svolgimento

La traccia chiede di effettuare un'analisi del codice Assembly di quello che sembra essere lo stesso malware analizzato precedentemente, stavolta potendo vederne le istruzioni.

In particolare, si richiede di individuare nel codice i costrutti noti (creazione dello stack, eventuali cicli, costrutti) e di ipotizzare il comportamento della funzionalità implementata.

Per facilitare la comprensione si riporta nuovamente l'immagine allegata nella traccia, dalla quale si può vedere che il codice Assembly viene sezionato in quattro blocchi.



1° Blocco – creazione dello stack e chiamata di funzione per verificare connessione a Internet

```

push    ebp
mov     ebp, esp
  
```

Il primo blocco si occupa della **creazione dello stack**, una regione della memoria temporanea per la funzione "InternetGetConnectedState", che viene chiamata in seguito.

```

push    ecx
push    0           ; dwReserved
push    0           ; lpdwFlags
  
```

Successivamente, vengono inseriti alcuni valori nello stack tramite operazione push (aggiungere un elemento in cima allo stack): si tratta dei parametri di cui necessiterà la funzione chiamata per svolgere la sua attività.

```

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

Viene quindi chiamata una funzione **internetGetConnectedState** (funzione chiamata) per verificare lo stato della connessione a Internet.

Il risultato di questa chiamata, cioè se la connessione è presente o meno, viene memorizzato in una variabile locale denominata **var_4**.

```

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

Successivamente, il codice, con l'istruzione cmp (compare) verifica se il valore di **var_4** è zero e, in caso affermativo, salta (jz) a una sezione di codice specificata da **loc_40102B**.

2° Blocco, a sinistra: Connessione a Internet attiva

Se il valore di **var_4** non è zero, il flusso di esecuzione passa a questo blocco.

```

push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
  
```

Qui, viene preparato un messaggio di successo, indicante che la **connessione a Internet è attiva**, che viene passato a una funzione chiamata **sub_40117F**, presumibilmente un printf in C, che stampa il messaggio a schermo.

```
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

Dopo la chiamata, il codice imposta il registro **eax** a 1 e salta a una sezione di codice specificata da **loc_40103A**.

3° Blocco: Connessione a Internet non disponibile

```
loc_40102B:                ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
```

Se il **valore** di **var_4** è **zero**, il flusso di esecuzione salta a questo blocco, che gestisce il caso in cui **la connessione a Internet non è disponibile**.

Viene preparato un messaggio di errore e passato alla stessa funzione **sub_40117F**, che stampa il messaggio a schermo.

Dopo la chiamata, **eax** viene azzerato, indicando un errore.

4° Blocco: Pulizia e rimozione dello stack della funzione chiamata

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

Indipendentemente dal percorso seguito nei blocchi precedenti, il flusso di esecuzione arriva al punto della cosiddetta “pulizia dello stack”, nella quale si ha la rimozione dello stack della funzione chiamata che ha esaurito il suo compito (retn, ovvero return).

Conclusioni

In sintesi, il codice sembra progettato, tramite chiamata alla funzione **internetGetConnectedState**, per verificare se c'è connessione ad Internet e agire di conseguenza, stampando messaggi di successo o errore a schermo.

Questa considerazione, derivante dall'analisi del codice Assembly, si chiarisce ulteriormente alla luce delle evidenze riscontrate nell'analisi statica basica dell'eseguibile.

Si tratta probabilmente di un malware che gestisce le situazioni di presenza o assenza di connessione ad Internet con l'intento di connettersi ad un URL specificato <http://www.practicalmalwareanalysis.com/cc.htm>, per inviare informazioni sul sistema operativo o sulla rete ad un server remoto sotto il controllo dell'attaccante oppure per scaricare risorse da Internet (per esempio altri malware).

In più, l'ipotesi che si tratti di un malware è ulteriormente confermata dal fatto che il software usa le funzioni load library e GetProcAddress, tipiche di caricamento dinamico di librerie a tempo di esecuzione, che suggeriscono un approccio stealth, discreto per rendere più difficile l'analisi statica ed evitare la rilevazione.