Progetto - S10/L5

Malware Aanalysis

Natalino Imbrogno

Cybersecurity Specialist
EPICODE

COSA SI INTENDE CON MALWARE ANALYSIS

La *malware analysis* è il processo di studio e comprensione di un software dannoso, noto anche come *malware*. Lo scopo di questo tipo di analisi è quello di determinare la funzionalità, l'origine ed il potenziale impatto del malware.

Esistono due tecniche principali di malware analysis:

- l'*analisi statica*, che fornisce tecniche e strumenti per analizzare il comportamento di un software malevolo senza la necessità di eseguirlo;
- l'*analisi dinamica*, che presuppone l'esecuzione del malware all'interno di un ambiente controllato.

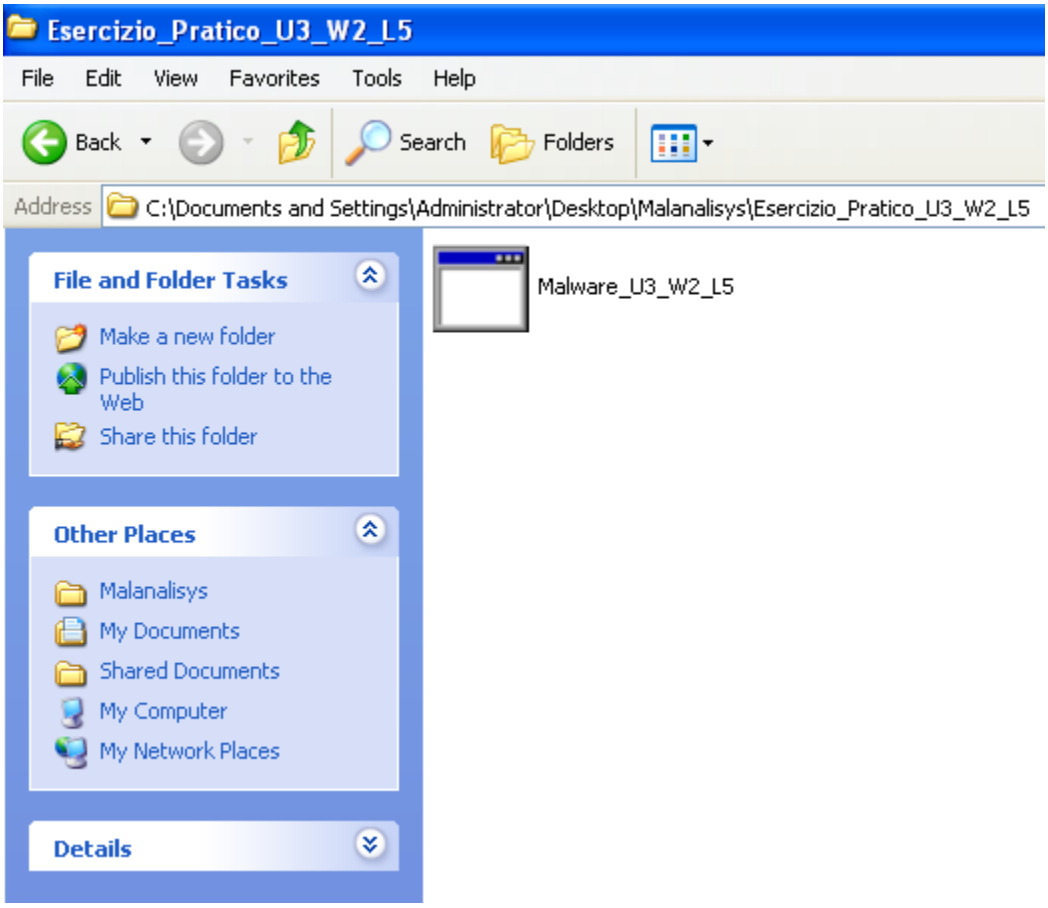
Le due tecniche sono tra di loro complementari. Infatti, al fine di condurre un'analisi efficace, i risultati di quelle statiche devono essere poi confermate dai risultati delle dinamiche.

Entrambe, inoltre, si dividono in *basica* e *avanzata*, ovvero, abbiamo:

- l'*analisi statica basica*, che consiste nell'esaminare un eseguibile senza vedere le istruzioni che lo compongono. Il suo scopo è quello 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 essere anche la più inefficiente, soprattutto con malware sofisticati;
- l'*analisi dinamica basica*, che 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 un ambiente sicuro e controllato 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;
- l'*analisi statica avanzata*, che 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*, i quali ricevono in input un file eseguibile e restituiscono in output il linguaggio *assembly*;
- l'*analisi dinamica avanzata*, che presuppone la conoscenza dei *debugger* per esaminare lo stato di un programma durante l'esecuzione.

TEST DI ANALISI STATICA BASICA

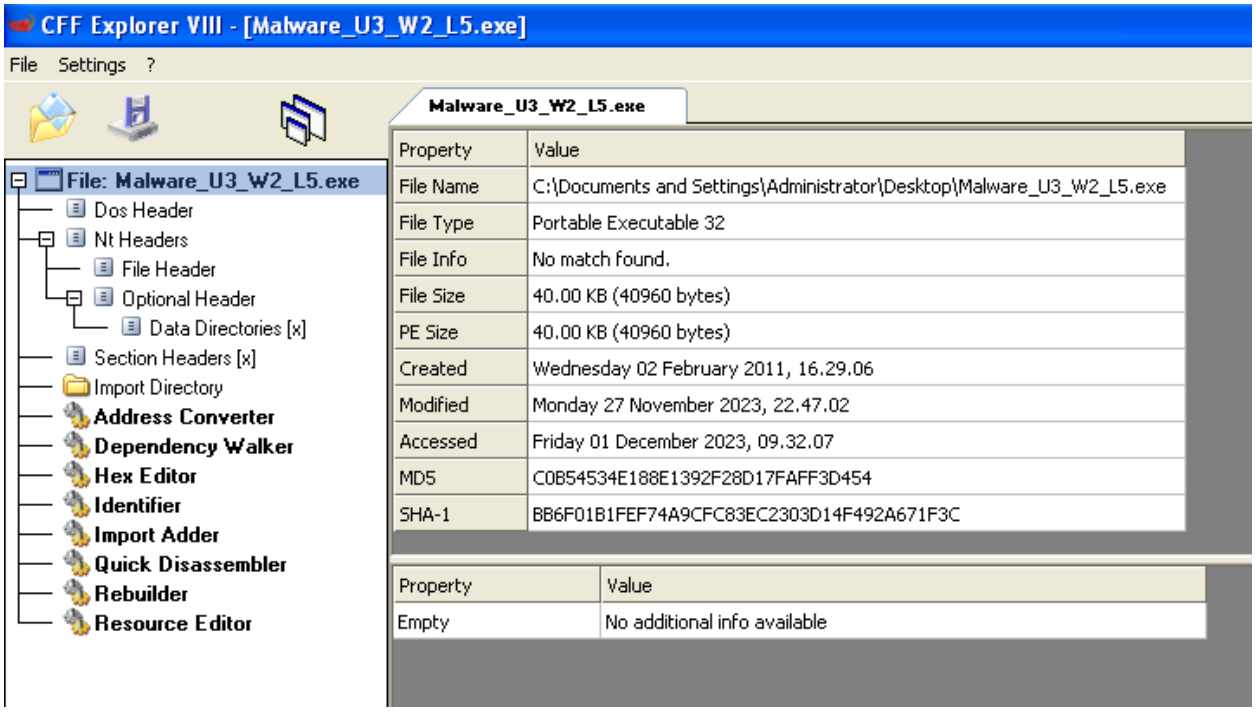
Eseguo ora un test di analisi statica basica analizzando il malware *Malware_U3_W2_L5*



Per farlo, mi avvalgo di un ambiente di test realizzato attraverso il tool *VirtualBox*, che consente l'esecuzione di macchine virtuali su sistemi operativi x86 e x86-64. In altre parole, consente di eseguire più sistemi operativi contemporaneamente sullo stesso computer.

Su VirtualBox installo quindi una macchina virtuale che monta il sistema operativo *Windows XP*.

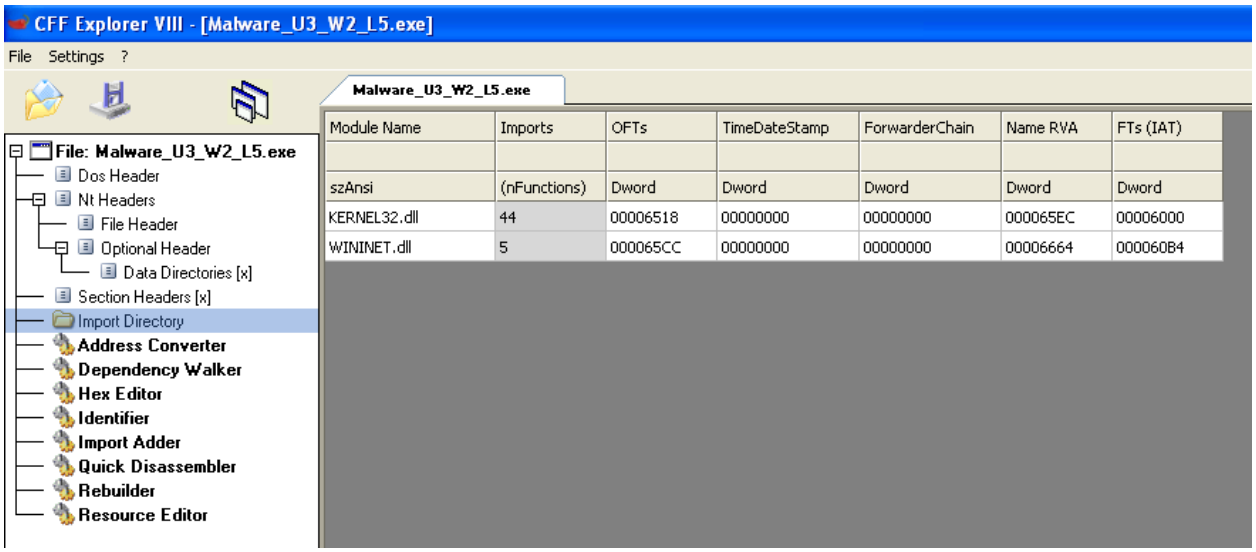
Per analizzare il malware, utilizzo il tool *CFF Explorer*, ovvero una suite di strumenti per l'analisi e la modifica di file eseguibili portatili (*PE*). E' uno strumento popolare tra gli ingegneri di retroingegneria e i *malware analyst* grazie alla sua capacità di fornire informazioni dettagliate sui file PE e alla sua facilità d'uso.



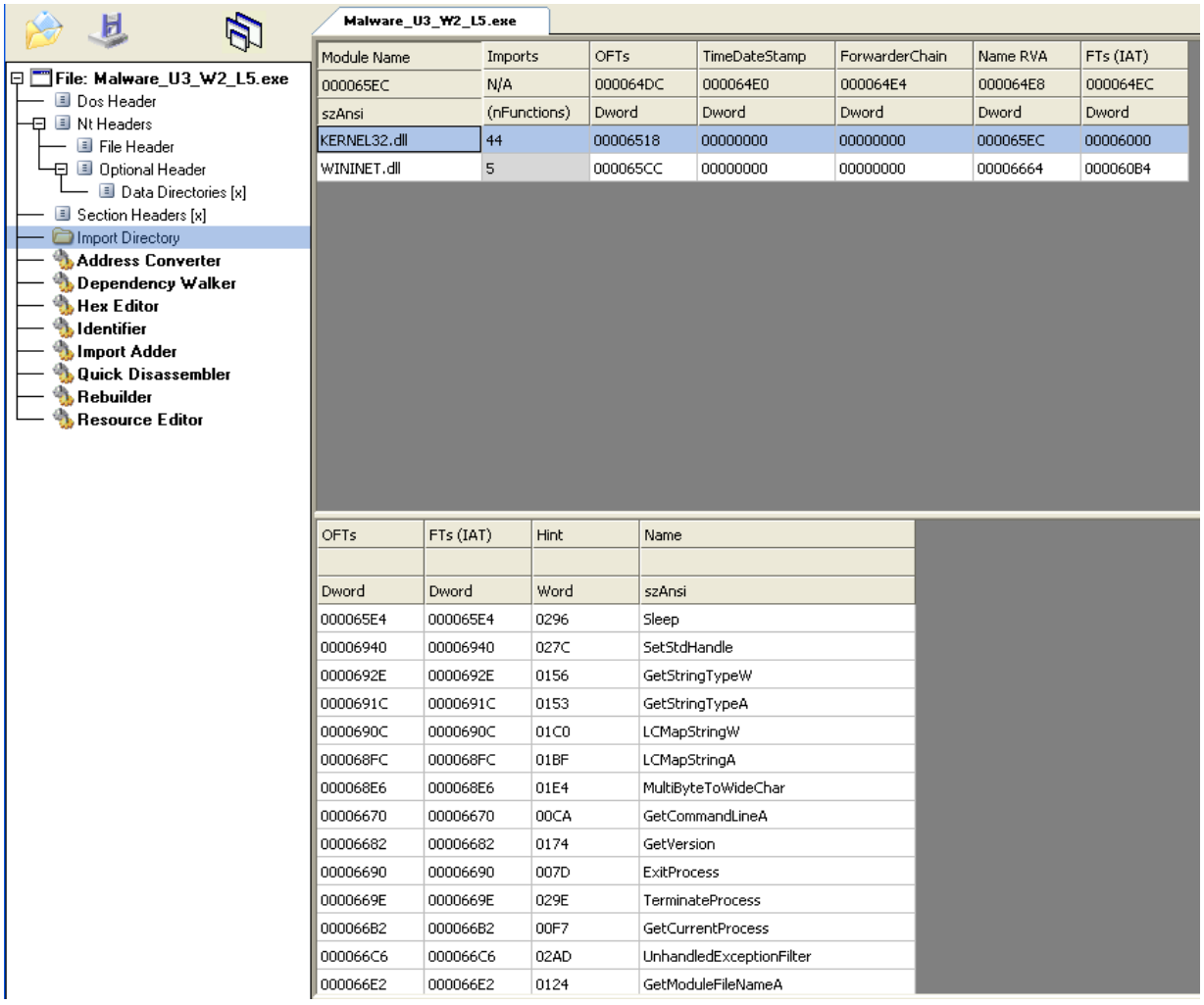
Il formato PE è tipico di Windows, e al suo interno contiene delle informazioni necessarie al sistema operativo per capire come gestire il codice del file, come ad esempio le librerie. Queste ultime contengono una serie di funzioni, e quando un programma necessita di una di esse, chiama una libreria al cui interno è definita la funzione necessaria. Di conseguenza, conoscere le librerie ed il loro scopo è una competenza fondamentale per la malware analysis.

Oltre alle funzioni importate, un file eseguibile può anche esportare funzioni, ovvero può mettere a disposizione di altri programmi o dell'utente delle funzioni da chiamare. Il formato PE contiene anche un elenco delle funzioni esportate da un eseguibile.

Perciò, dopo aver avviato CFF Explorer e aver caricato l'eseguibile del malware, mi reco nella sezione *Import Directory* per controllare le librerie e le funzioni importate

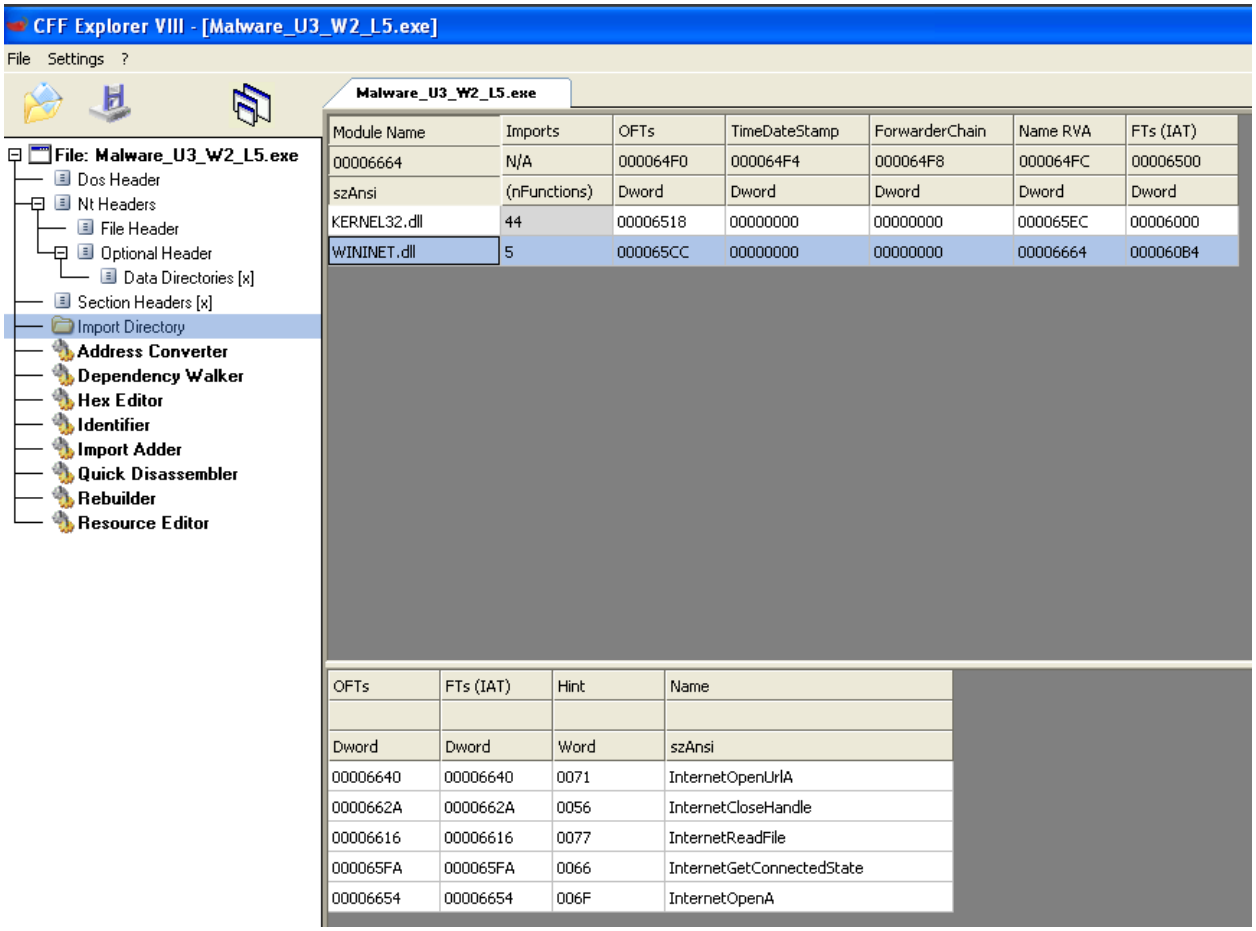


Mi sposto poi ad analizzare la prima libreria



KERNELL32.dll è una libreria di sistema di Windows che fornisce funzioni di base per il sistema operativo, come l'accesso alla memoria, l'esecuzione di processi e la gestione dei file.

Passo poi alla seconda libreria



WININET.dll è una libreria di sistema di Windows che fornisce funzioni per l'accesso a internet, come la connessione a siti web, il download di file e l'invio di richieste HTTP.

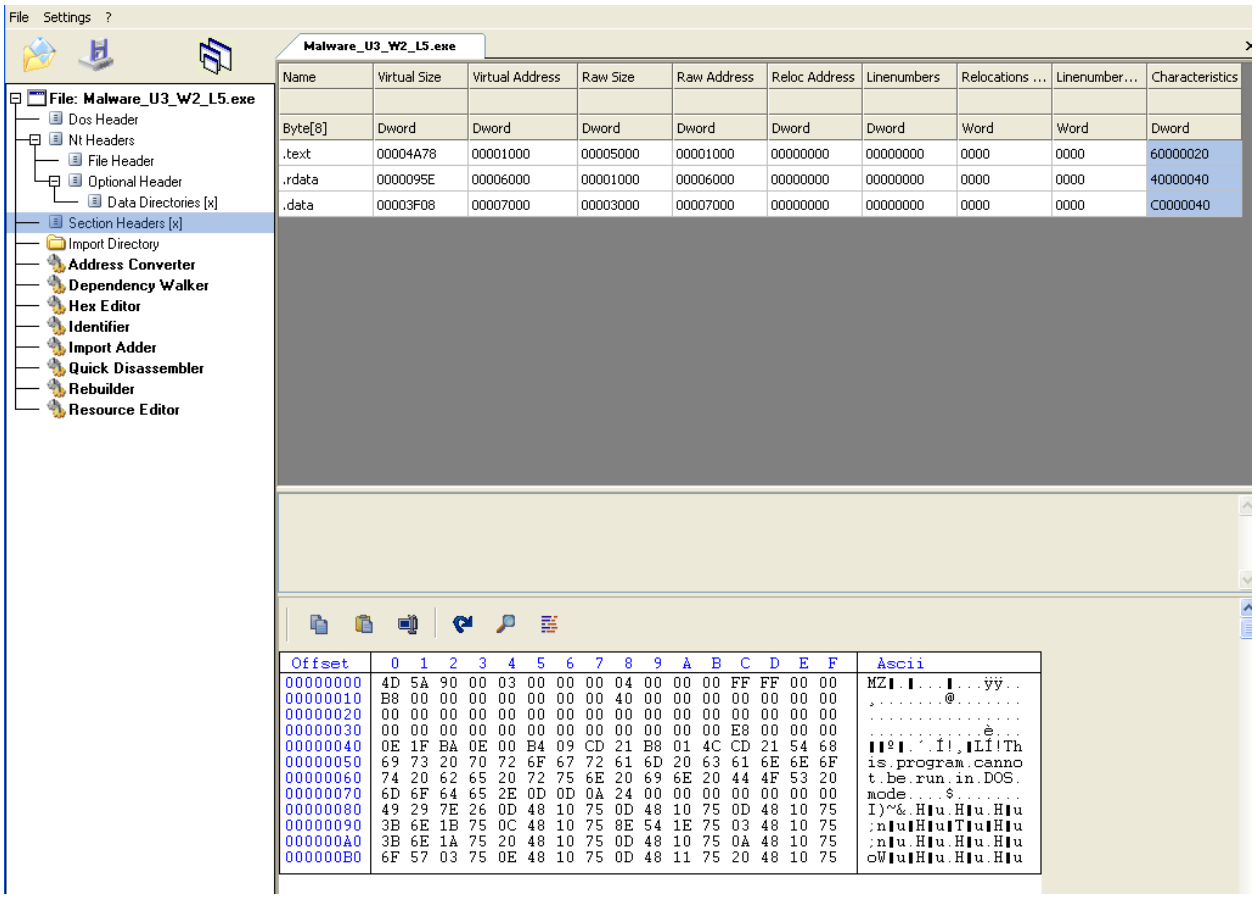
In base a queste informazioni, è possibile ipotizzare che questo malware utilizzi tali librerie per i seguenti scopi:

- eseguire azioni sul sistema operativo. Ad esempio, il malware potrebbe utilizzare le funzioni di KERNEL32.dll per accedere alla memoria dell'utente, eseguire processi o modificare i file;
- accedere a internet. Ad esempio, il malware potrebbe utilizzare le funzioni di WININET.dll per connettersi a siti web, scaricare file o inviare richieste HTTP.

Così facendo, in teoria, può:

- nascondersi da programmi antivirus o eseguire codice dannoso in background;
- scaricare file dannosi dal web o inviare informazioni sensibili all'attaccante.

Mi sposto ora nella sezione *Section Headers [x]* per analizzare le sezioni di cui si compone il malware



Le sezioni sono le parti che compongono un software e che svolgono funzioni specifiche.

In questo caso, abbiamo:

- *.text*, che contiene le istruzioni 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;

- *.data*, che contiene tipicamente i dati / le variabili globali del programma eseguibile, i quali devono essere disponibili da qualsiasi parte del programma. 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 all'interno dell'eseguibile.

In base a ciò, posso dedurre che questo malware potrebbe essere un *trojan horse* progettato per rubare informazioni dal computer dell'utente. Infatti, il presente file malevolo:

- è un eseguibile di 5000 byte;
- contiene codice per eseguire operazioni di base come l'apertura di file e la scrittura di dati su disco;
- contiene codice per comunicare con un server remoto.

Queste caratteristiche mi suggeriscono che il malware potrebbe essere progettato per eseguire le seguenti azioni:

- aprire i file e leggere i dati contenuti in essi;
- scrivere dati su disco, ad esempio per creare file nascosti o modificare i file esistenti;
- comunicare con un server remoto per inviare i dati rubati.

In particolare, esso potrebbe rubare le seguenti informazioni:

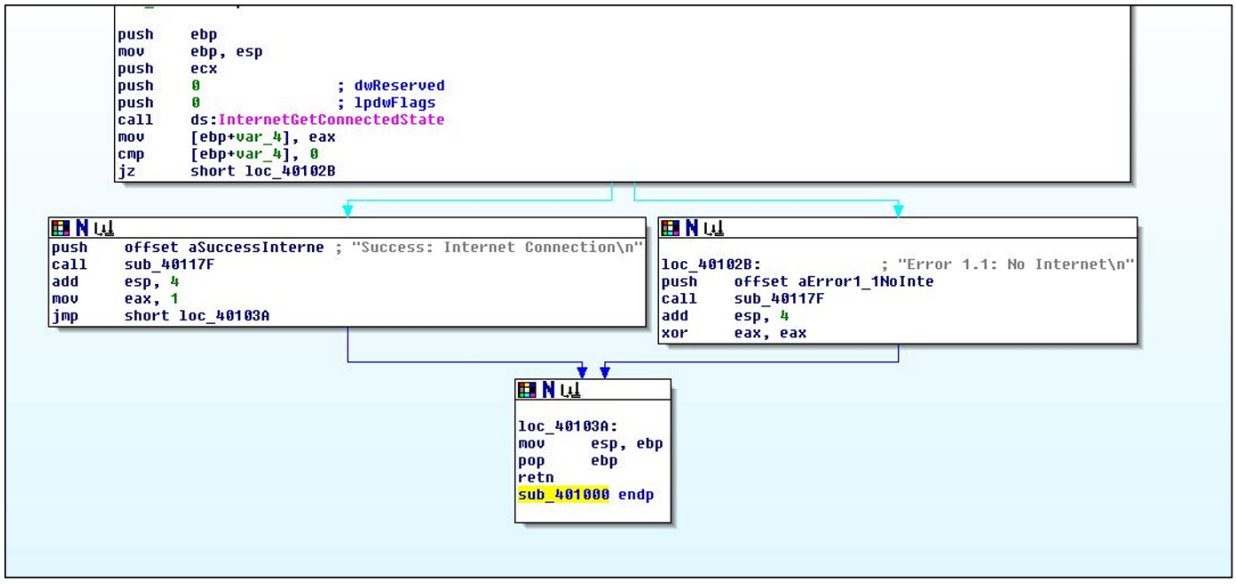
- dati personali come nome, cognome, indirizzo, numero di telefono e indirizzo email;
- informazioni finanziarie come dati di carte di credito e conti bancari;
- password e credenziali di accesso.

Riguardo al trojan horse, in generale, c'è da dire che è un tipo di malware il quale si presenta come un programma innocuo per indurre l'utente a scaricarlo ed eseguirlo. Una volta lanciato, il trojan può eseguire una serie di azioni dannose, come:

- rubare informazioni personali;
- rubare informazioni finanziarie;
- rubare password e credenziali di accesso;
- installare altri malware sul computer;
- prendere il controllo completo del computer.

ANALISI STATICA AVANZATA

Passo ora a studiare, attraverso le tecniche e le metodologie dell'analisi statica avanzata, uno snippet di codice scritto nel *linguaggio assembly*



Assembly è un linguaggio di programmazione di basso livello che viene utilizzato per comunicare direttamente con l'hardware di un computer. Le istruzioni assembly sono istruzioni specifiche per un particolare processore, e quindi ogni tipo di processore ha il proprio linguaggio assembly.

E' composto da *mnemonici*, ovvero parole chiave che rappresentano istruzioni specifiche. Assembly è un linguaggio potente e versatile, ma richiede una comprensione approfondita dell'hardware del dispositivo e del funzionamento del linguaggio macchina.

Viene utilizzato principalmente per la programmazione a livello di sistema, come i sistemi operativi, i driver di dispositivi e i sistemi embedded. Viene anche utilizzato per la programmazione di codice ottimizzato per le prestazioni, come il codice di gioco o il codice di elaborazione dei segnali.

I malware analyst possono avvalersi del linguaggio assembly per una serie di scopi, tra cui:

- analisi del codice: l'assembly può essere utilizzato per analizzare il codice malware ad un livello più granulare rispetto ai linguaggi di programmazione di

alto livello. Ciò può aiutare gli analisti a comprendere il funzionamento del malware e ad identificare le sue funzionalità;
- rilevamento del malware: l'assembly può essere utilizzato per rilevare il malware che è stato offuscato o crittografato per renderlo più difficile da analizzare;
- rimozione del malware: l'assembly può essere utilizzato anche per rimuovere il malware dal sistema infettato.

Tornando ora allo snippet preso in considerazione, comincio innanzitutto ad identificare i costrutti che vedo, ovvero:

- creazione dello stack: le prime due righe di codice, *push ebp* e *mov ebp, esp*, servono a creare lo stack. *push ebp* salva il valore di *ebp* sul fondo dello stack, in modo da poterlo ripristinare alla fine della funzione. *mov ebp, esp* imposta *ebp* all'indirizzo corrente dello stack, in modo da poter accedere agli argomenti e alle variabili locali della funzione;
- ciclo: la funzione contiene il ciclo condizionale *cmp [ebp+var_4], 0*. Questo ciclo verifica se il valore della variabile *var_4* è uguale a zero. Se lo è, il ciclo salta alla label *loc 401028*.

In generale, posso affermare che questa funzione controlla la connessione a internet. La variabile *var_4* contiene lo stato della connessione. Se quest'ultimo è uguale a zero, la connessione non è attiva. In questo caso, la funzione stampa un messaggio di errore e restituisce il valore *1*. Se lo stato è diverso da zero, la connessione è attiva. In questo caso, la funzione stampa un messaggio di successo e restituisce il valore *0*.

Difatti, esaminando il codice riga per riga, posso notare che:

- *push ebp* salva il valore di *ebp* sul fondo dello stack;
- *mov ebp, esp* imposta *ebp* all'indirizzo corrente dello stack;
- *push ecx* salva il valore di *ecx* sullo stack;
- *push 0* salva il valore zero sullo stack;
- *push lpdwFlags* salva l'indirizzo di *lpdwFlags* sullo stack;
- *call ds:InternetGetConnectedState* esegue la funzione *InternetGetConnectedState*;
- *mov [ebp+var_4], eax* salva il risultato della funzione *InternetGetConnectedState* nella variabile *var_4*;
- *cmp [ebp+var_4], 0* verifica se il valore della variabile *var_4* è uguale a zero;
- *jz loc 401028* salta alla label *loc 401028* se il valore è uguale a zero;
- *push offset aSuccessInternet* salva l'indirizzo della riga *Success: Internet Connection|n* sullo stack;

- *call sub 40117F* esegue la funzione *sub 40117F*, che stampa la stringa sullo standard output;
- *jmp short loc 40103A* salta alla label *loc 40103A*;
- *add esp, 4* rimuove 4 byte dallo stack;
- *push offset aError1_1NoInternet* salva l'indirizzo della stringa *Error 1.1: No Internet|n* sullo stack;
- *mov eax, 1* imposta il valore di *eax* a 1;
- *call sub 40117F* esegue la funzione *sub 40117F*, che stampa la stringa sullo standard output;
- *jmp short loc 40103A* salta alla label *loc 40103A*;
- *add esp, 4* rimuove 4 byte dallo stack;
- *xor eax, eax* imposta il valore di *eax* a zero;
- *pop ebp* ripristina il valore di *ebp* dallo stack;
- *retn* termina la funzione.