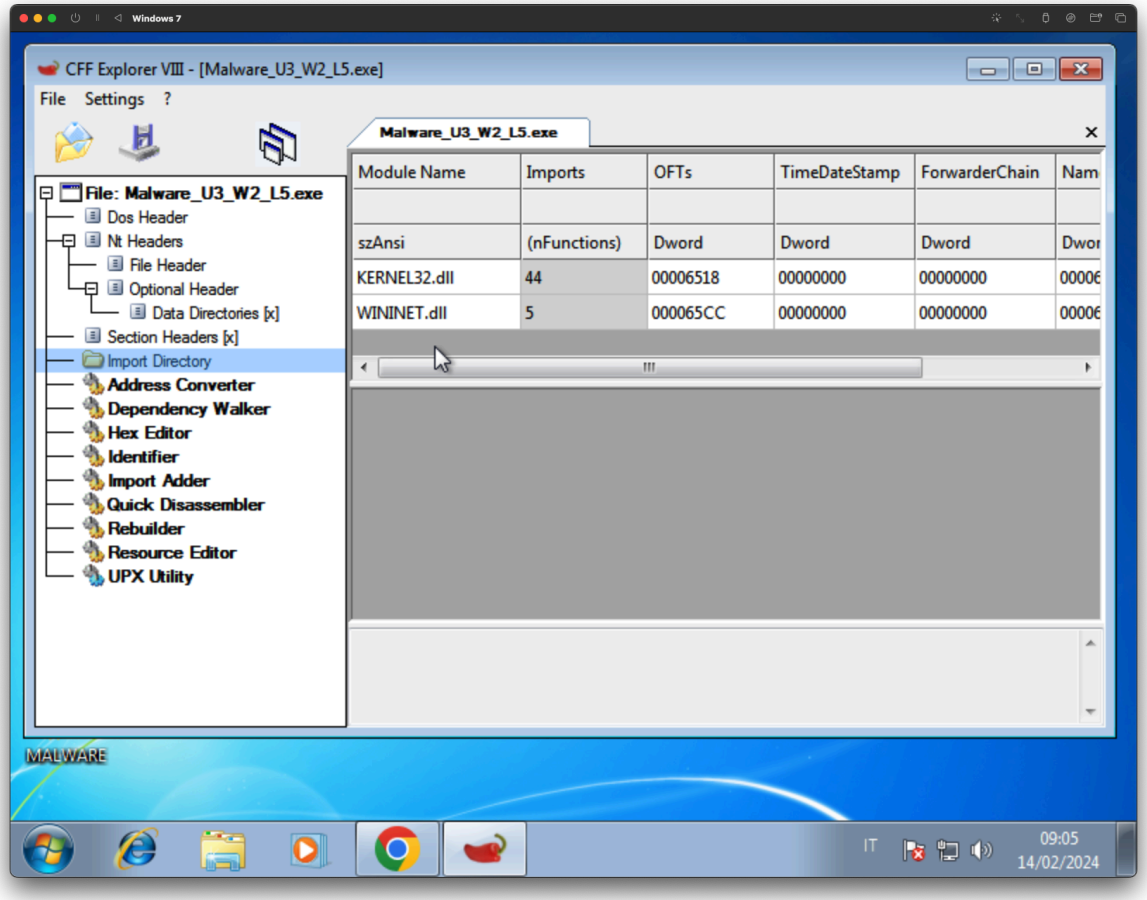
S10-L5 Analisi malware

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:
1. Quali librerie vengono importate dal file eseguibile?
2. Quali sono le sezioni di cui si compone il file eseguibile del malware? Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:
3. Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti)
4. Ipotizzare il comportamento della funzionalità implementata
5. BONUS fare tabella con significato delle singole righe di codice assembly

Punto 1 - Analisi librerie

Per iniziare l'analisi del malware utilizzeremo un software chiamato CFF explorer: questo software permette l'analisi di file (possibili malware) per controllare le funzioni importate ed esportate. A seguito dell'analisi questi sono i risultati.



Nell’immagine sono evidenziate le librerie importate dal software.
Ecco una spiegazione delle due librerie:
Kernel32.dll: Questa libreria è una delle librerie fondamentali di Microsoft Windows. Contiene una vasta gamma di funzioni che permettono alle applicazioni di interagire direttamente con il sistema operativo Windows. Alcune delle sue principali funzionalità includono:

- Manipolazione dei file: Kernel32.dll fornisce funzioni per creare, aprire, leggere, scrivere, chiudere, rinominare e eliminare file e directory sul sistema operativo.
- Gestione della memoria: La libreria offre funzioni per allocare e liberare memoria, così come per proteggere e modificare le regioni di memoria esistenti.
- Gestione dei processi e dei thread: Kernel32.dll contiene funzioni per creare, terminare, sospendere e riprendere processi e thread.
- Gestione del tempo e delle date: La libreria fornisce funzioni per ottenere e impostare l'ora di sistema, nonché per convertire date e ore tra formati diversi.

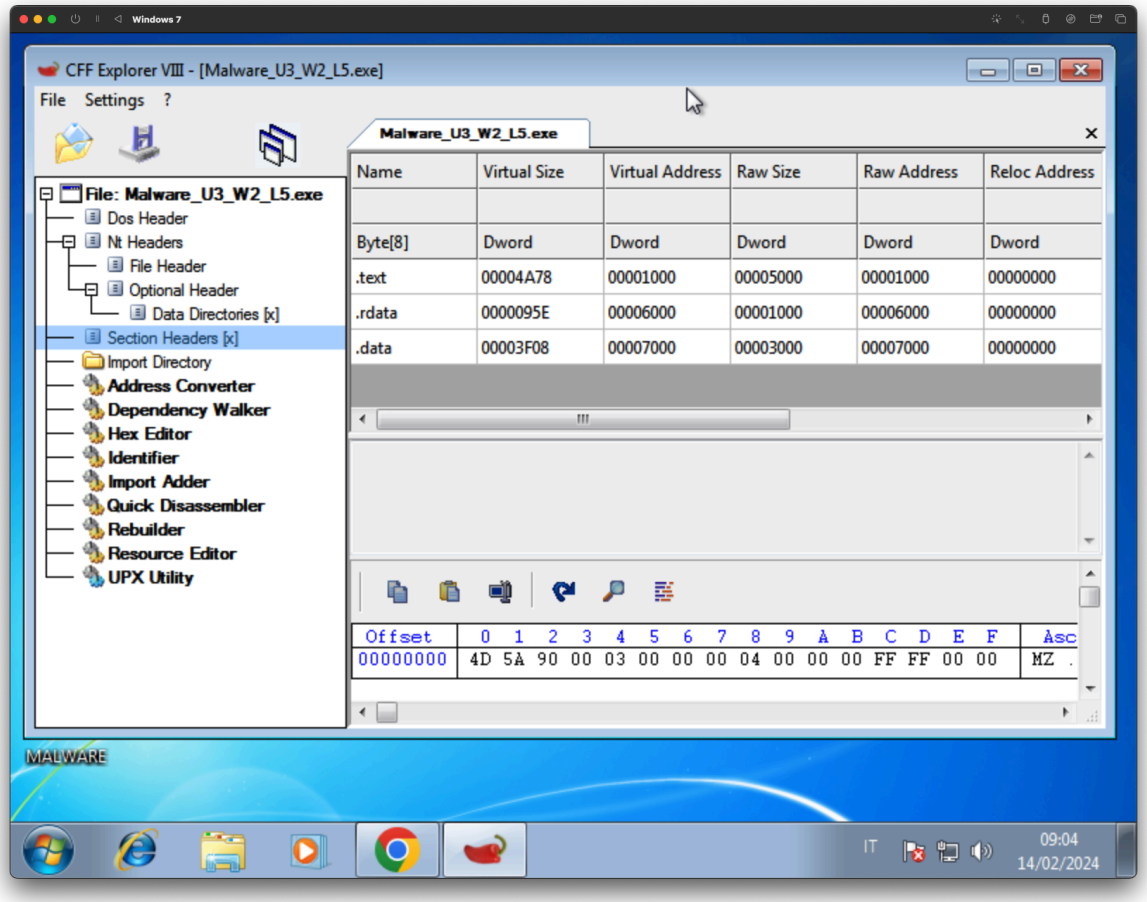
Wininet.dll: Questa libreria è specifica per l'accesso e la gestione delle risorse di rete su piattaforma Windows. Contiene un insieme di funzioni per implementare e gestire vari protocolli di rete, tra cui HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), e NTP (Network Time Protocol). Alcune delle sue funzionalità principali includono:

- Implementazione dei protocolli di rete: Wininet.dll offre funzioni per inizializzare e configurare sessioni di rete, inviare richieste HTTP e FTP ai server, e ricevere e gestire le risposte.
- Gestione dei cookie: La libreria include funzioni per la lettura, la scrittura e la gestione dei cookie, che sono importanti per mantenere lo stato della sessione durante le comunicazioni HTTP.
- Cache delle risorse: Wininet.dll supporta la gestione della cache delle risorse web, consentendo alle applicazioni di memorizzare in locale risorse scaricate da Internet e di recuperarle dalla cache quando necessario, riducendo così il traffico di rete e migliorando le prestazioni.

In sintesi, Kernel32.dll fornisce funzioni di basso livello per l'interazione con il sistema operativo, come la manipolazione dei file e la gestione della memoria, mentre Wininet.dll è specifica per l'accesso e la gestione delle risorse di rete, come HTTP, FTP e NTP.

Punto 2 - Analisi delle sezioni

Nella seconda immagine possiamo notare le sezioni di cui è composto il software:
Certamente, posso fornirti una spiegazione più dettagliata sulle diverse sezioni di un file eseguibile:

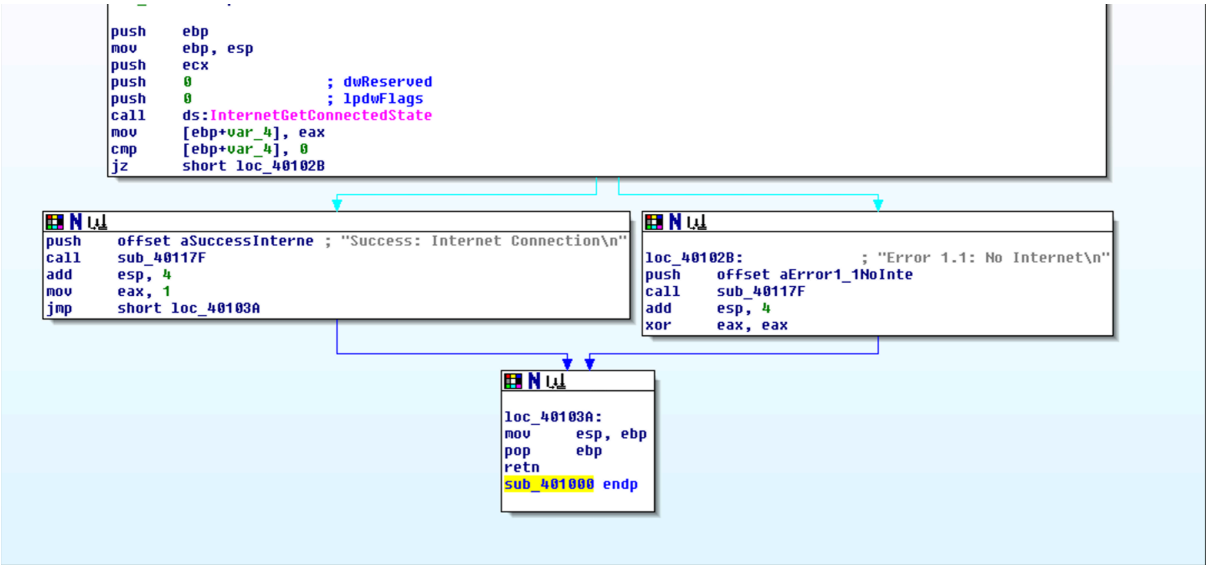


.text: Questa sezione contiene il codice eseguibile del programma. È dove sono memorizzate le istruzioni che la CPU eseguirà una volta che il software sarà avviato. Il codice scritto in questa sezione definisce le operazioni e le istruzioni che il programma deve eseguire per svolgere le sue funzioni.

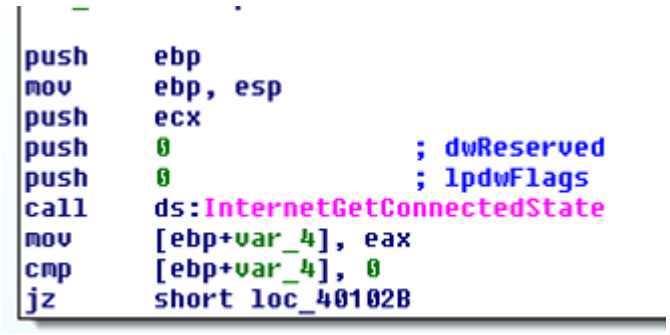
.rdata: Questa sezione include generalmente informazioni riguardanti librerie e funzioni importate ed esportate dall'eseguibile. Può contenere dati costanti che sono necessari per il funzionamento del programma, come ad esempio stringhe di testo, tabelle di lookup, o altre costanti utilizzate dal programma durante l'esecuzione. Questi dati sono di solito di sola lettura, quindi la sezione è chiamata ".rdata", che sta per "read-only data".

.data: La sezione .data contiene tipicamente dati e variabili globali del programma eseguibile.Queste variabili globali sono memorizzate in una porzione di memoria dedicata all'esecuzione del programma e possono essere lette e scritte durante l'esecuzione del programma. Questa sezione è spesso utilizzata per memorizzare dati che devono essere condivisi tra diverse parti del programma o che devono mantenere il loro stato anche dopo la terminazione di una funzione.

Punto 3 - Costrutti presenti nel codice



Lo schema proposto ha all'interno alcuni costrutti noti, in questa parte dello schema è possibile notarne alcuni:



Analizzando la prima parte di codice è possibile notare nelle prime due righe la creazione dello stack.
Il valore del puntatore base (EBP) viene salvato nello stack con "push ebp", quindi il valore dello stack pointer (ESP) viene copiato in EBP con mov ebp, esp. Questo stabilisce un nuovo frame dello stack per la funzione corrente.
Nelle quattro righe successive possiamo notare la chiamata di funzione.
Nelle ultime due righe possiamo invece riconoscere un costrutto che può essere paragonato a un ciclo if nei linguaggi di alto livello.
Il codice indicato è un esempio di un costrutto di controllo condizionale, che verifica se il valore memorizzato in " [ebp+var_4] " è uguale a zero. Se il valore è zero, il programma salta a `loc_40102B`, altrimenti procede con l'esecuzione delle istruzioni successive.

Punto 4 - Che cosa svolge questo codice?

Questo codice in assembly è una parte di un programma che controlla lo stato della connessione Internet e agisce di conseguenza. Ecco come opera:

Fase 1: In questa fase, vengono eseguite le seguenti operazioni:

- Viene configurato lo stack frame iniziale.
- Vengono pushati alcuni valori nello stack per essere utilizzati come argomenti per la chiamata alla funzione “ InternetGetConnectedState ”.
- - Viene chiamata la funzione “ InternetGetConnectedState “ per ottenere lo stato della connessione Internet.
- Il valore restituito dalla funzione “ InternetGetConnectedState “ viene memorizzato nella variabile “ [ebp+var_4] ”.
- Viene effettuato un confronto per verificare se il valore restituito è uguale a zero.
- In base al risultato del confronto, il controllo del programma passa a uno dei possibili risultati.

Fase 2: In base ai possibili risultati della fase 1 abbiamo diversi comportamenti:

- Se lo stato della connessione Internet è positivo, viene eseguita una serie di operazioni per gestire il successo dell’operazione.
- Se lo stato della connessione Internet è negativo, viene eseguita una serie di operazioni per gestire l'errore.

Fase 3 : Nella terza fase indipendentemente dal risultato ottenuto, il controllo del programma viene trasferito a “ loc_40103A “, dove vengono ripristinati lo stack frame e il controllo viene restituito al chiamante tramite l'istruzione “ retn “.

In sintesi, questo codice controlla lo stato della connessione Internet e gestisce il risultato ottenuto, eseguendo operazioni diverse in base al risultato ottenuto.

Punto 5 - Bonus

Fase 1	
Push ebp	Salva il valore corrente del puntatore base (EBP) nello stack.
Mov ebp, esp	Imposta il puntatore base (EBP) al valore corrente dello stack pointer (ESP).
Push ecx	Salva il valore corrente del registro ECX nello stack.
Push 0 ; dwReserved	Mette sullo stack il valore 0 come argomento per la funzione InternetGetConnectedState.
Push 0 ; 1pdwFlags	Mette sullo stack il valore 0 come argomento per la funzione InternetGetConnectedState.
Call ds: InternetGetConnectedState	Chiama la funzione InternetGetConnectedState per controllare lo stato della connessione.
Mov [ebp+var_4], eax	Memorizza il risultato della chiamata di funzione in [ebp+var_4].

Cmp [ebp+var_4], 0	Confronta il valore memorizzato in [ebp+var_4] con 0.
Jz [ebp+var_4], 0 short loc_40102B	Salta a loc_40102B se il valore in [ebp+var_4] è uguale a 0 (quindi nessuna connessione).
Possibili risultati della fase 1	
Risultato 1	Se lo stato di connessione Internet è positivo:
Push offset aSuccessInterne	Mette sullo stack l'offset della stringa "Success: Internet Connection\n".
Call sub_40117F	Chiama la subroutine sub_40117F.
Add esp, 4	Dealloca lo spazio utilizzato per i parametri passati alla funzione.
Mov eax, 1	Imposta il registro EAX a 1.
Jmp short loc_40103A	Salta incondizionatamente a loc_40103A.
Risultato 2	Se lo stato della connessione Internet è negativo:
loc_40102B:	Etichetta per l'errore di connessione.
push offset aError1_1NoInte	Mette sullo stack l'offset della stringa "Error 1.1: No Internet\n".
Call sub_40117F esp, 4	Chiama la subroutine sub_40117F.
Add esp, 4	Dealloca lo spazio utilizzato per i parametri passati alla funzione.
Xor eax, eax	Azzera il registro EAX.
Risultato finale	
loc_40103A:	Etichetta per il ritorno.
Mov esp, ebp	Ripristina il puntatore dello stack (ESP) al valore salvato nel puntatore base (EBP).
Pop ebp	Ripristina il puntatore base (EBP) dallo stack.
retn	Restituisce il controllo al chiamante.
sub_431333 endp	Fine della subroutine.