Progetto S10/L5

Questo progetto si divide in due fasi secondo il file fornito dall'esercitazione,

Nella prima fase ci viene richiesto di:

1) identificare le librerie che vengono importate dal file eseguibile
2) identificare le sezioni di cui si compone il file eseguibile del malware

Nella seconda fase ci viene richiesto di analizzare il codice riportato e di:

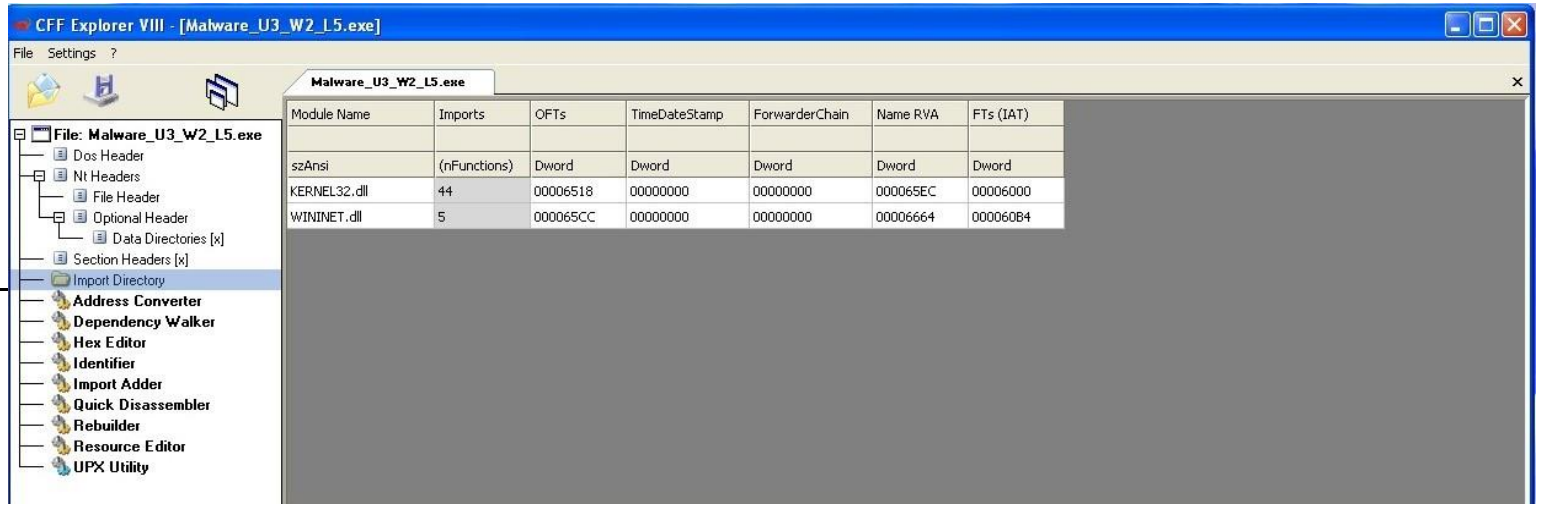
3) identificare i costrutti noti
4) ipotizzare il comportamento della funzionalità implementata
5) BONUS fare una tabella con il significato delle singole righe di codice Assembly

La prima fase si basa sull'**analisi statica basica** che ci consente di determinare se un file è malevolo o meno. Si valuta, dunque, il codice sorgente senza eseguire il programma. Tra i tool che possono essere utilizzati, per questa specifica procedura noi andremo ad usare CFF Explorer che ci darà le informazioni richieste ai punti 1 e 2.

Fase 1

1)

Librerie



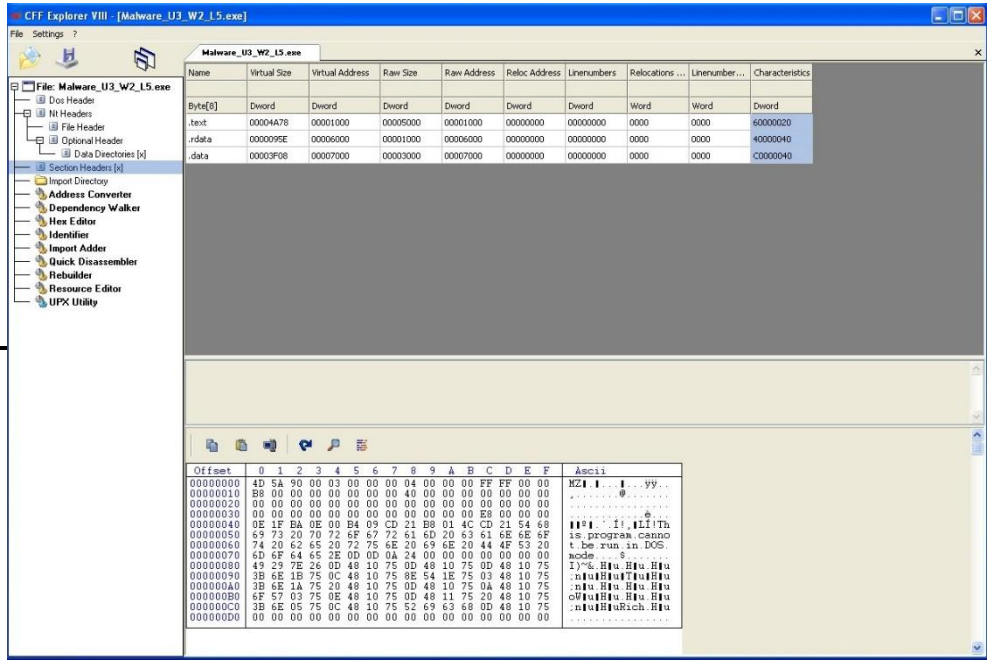
Come possiamo vedere dall'immagine, andando ad analizzare il malware, le librerie presenti sono:

- **KERNEL 32.dll**
  È una libreria presente nei sistemi operativi Windows, fa parte del kernel (il nucleo del sistema operativo) di Windows. Questa libreria fornisce molte funzioni di base essenziali per il funzionamento del sistema operativo e delle applicazioni, ad esempio manipolazione dei file e gestione della memoria

- **WININET.dll**
  E' una libreria che fornisce funzionalità di networking per le applicazioni di Windows. Viene spesso utilizzata per effettuare operazioni di rete come l'accesso a risorse su internet, download e upload di file e la gestione delle connessioni HTTP, FTP e NTP

2)

Sezioni



Dall'immagine emerge che le sezioni di cui si compone il file sono:

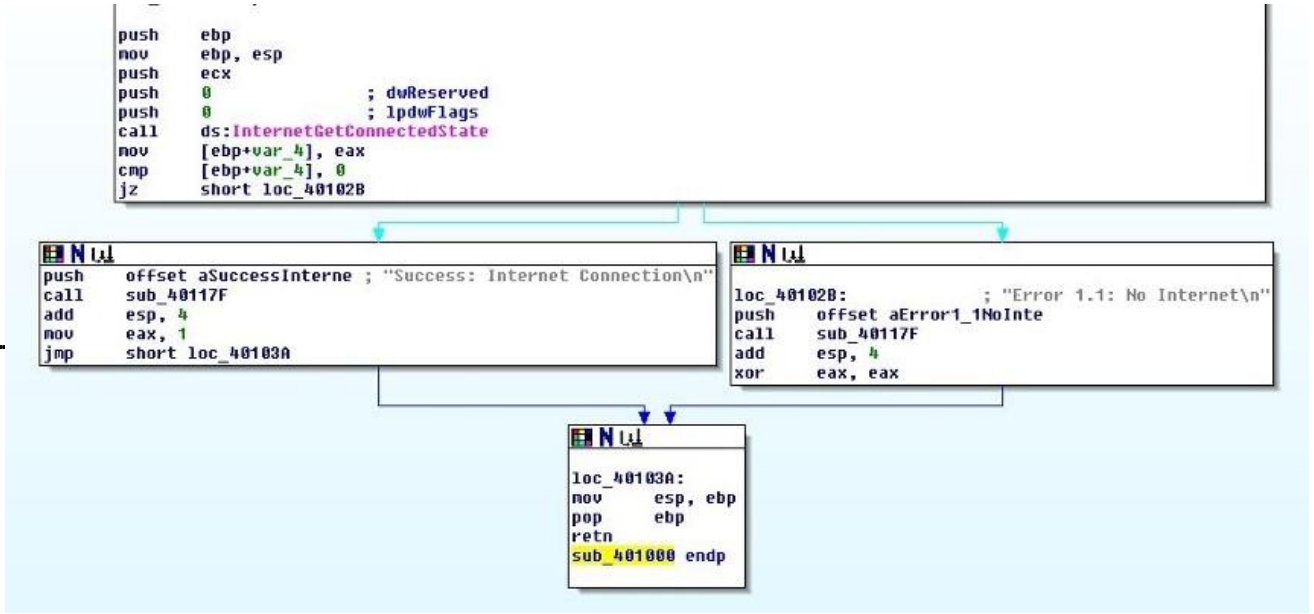
- **.text**
  contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto

- **.rdata**
  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**
  contiene tipicamente i dati / le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma.

3)

Codice



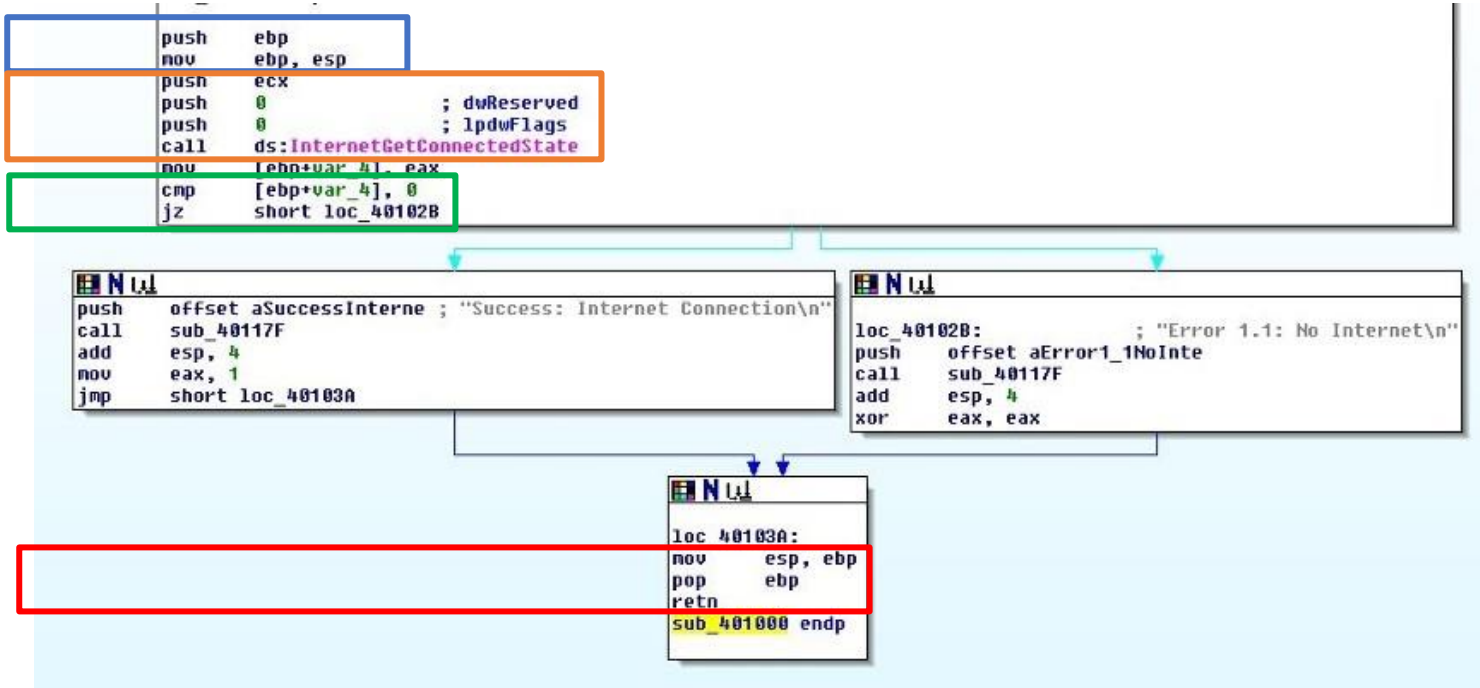
In base al codice fornito possiamo determinare che i costrutti sono così suddivisi:

Creazione dello stack

Chiamata di funzione

Ciclo if

Chiusura dello stack



In base allo schema possiamo confermare che il codice contiene:

- **Creazione dello stack**
- **Chiamata di funzione**
- **Ciclo if**
- **Chiusura dello stack**

I costrutti svolgono diverse funzioni nella scrittura e nell'esecuzione dei programmi, sono di fatto le istruzioni che vengono utilizzate per scrivere il codice.

Creazione dello stack:

Salva il valore corrente del registro base (EBP) nello stack e imposta il registro base al valore corrente dello stack (ESP), in sintesi spinge il valore attuale del registro di base EBP sulla cima dello stack e lo copia nel registro ESP. Queste due istruzioni creano un frame di stack per la funzione in modo che possa utilizzare variabili locali.

Chiamata di funzione:

Quando si utilizza una chiamata di funzione il controllo del programma viene temporaneamente trasferito a un'altra parte del codice. Nel nostro specifico caso l'istruzione "push" va a preparare i dati per una chiamata di funzione mentre "**call**" viene utilizzata per effettuare la chiamata di funzione, questa istruzione indica che il programma sta chiamando la funzione "InternetGetConnectedState". Questa istruzione salva l'indirizzo di ritorno nello stack e trasferisce il controllo all'inizio della funzione chiamata

Ciclo if:

Il ciclo if viene utilizzato per eseguire un blocco di istruzioni solo se una determinata condizione è vera. Dall'estratto di questo codice lo si può individuare nelle diciture "cmp [ebp+var_4], 0" e "jz short loc_40102B", infatti questo codice fa un confronto (cmp) tra il valore situato all'indirizzo [ebp+var_4] e zero, successivamente l'istruzione "jz" (jump if zero) eseguirà un salto corto (short) alla posizione specificata (loc_40102B). Quindi, se il valore ZF (zero flag) è settato, se restituisce zero (cioè i due valori sono uguali), allora il salto condizionato jz viene eseguito.

Chiusura dello stack:

Quando una funzione termina o quando il programma completo termina, lo spazio di memoria utilizzato dallo stack viene deallocato. In altre parole, è il processo di liberazione dello spazio di memoria precedentemente allocato per uno stack.

4)
In base alle informazioni che abbiamo questo malware sembra determinare la verifica di una connessione ad internet, lo si può vedere, come già spiegato nella chiamata di funzione, dall'istruzione "call" che indica che il programma sta chiamando la funzione "InternetGetConnectedState", la quale viene utilizzata per determinare se il sistema è connesso ad internet.
inoltre nei blocchi successivi possiamo vedere che viene probabilmente richiesta una stampa a schermo dell'esito della procedura, quindi "Success internet connection" qualora si verificasse la connessione e "Error 1.1: No Internet" qualora, in caso contrario, la connessione non fosse andata a buon fine.

5) **BONUS**

Specifiche delle istruzioni:

push ebp (salva il valore del registro base EBP nello stack)

mov ebp, esp (sposta il valore del registro ESP nel registro base EBP)

push ecx (salva il valore del registro ECX nello stack)

push 0 ; dwReserved (l'istruzione push 0 indica che il valore zero viene inserito nello stack. Questo valore sembra associato ad un parametro denominato dwReserved)

push 0 ; lpdwFlags (salva il valore zero nello stack. Probabilmente lodwFlags verrà impostato a zero)

call ds:InternetGetConnectedState ("call è un'istruzione che esegue una chiamata quindi questa istruzione indica che il programma sta chiamando la funzione "InternetGetConnectedState", che viene utilizzata per determinare se il sistema è connesso ad internet)

mov [ebp+var_4], eax (copia il valore contenuto nel registro EAX in una posizione specifica dello stack – [ebp+var_4] – dove EBP rappresenta il registro e var_4 la posizione)

cmp [ebp+var_4], 0 (confronta il valore memorizzato nella variabile locale – [ebp+var_4] – con zero)

jz short loc_40102B ("jump if zero" rappresenta un'istruzione di salto condizionato quindi, come detto, a seguito della precedente istruzione, questa istruzione effettuerà un breve salto se il valore ZF - zero flag - è settato, se restituisce cioè zero, ovvero se i due valori sono uguali)

push offset aSuccessInterne ; "Success: Internet Connection\n" (questa istruzione inserisce l'indirizzo di memoria della stringa "Success:Internet Connection\n" nello stack, e potrebbe essere utilizzato per una funzione che stampa la stringa a schermo)

call sub_40105F (questa istruzione è utilizzata per chiamare una subroutine etichettata come 40105F)

add esp, 4 (aggiunge il valore 4 al registro ESP. In questo caso, dopo la chiamata, può deallocare lo spazio sullo stack per liberarlo)

mov eax, 1 (assegna il valore 1 al registro EAX)

jmp short loc_40103A (indica un salto breve che trasferisce il programma all'indirizzo 40103°

loc_40102b (rappresenta un punto specifico nel codice)

push offset aError1_NoInte (mette la stringa "Error 1.1: No Internet\n" nello stack)

call sub_40117F (chiama la funzione alla locazione 40117F)

add esp, 4 (viene assegnato il valore 4 al registro ESP)

xor eax, eax (imposta il registro EAX a zero)

loc_40103A: (prepara la funzione per tornare al chiamante)

mov esp, ebp (ripristina il valore del registro ESP copiandovi il valore di EBP)

pop ebp (ripristina il registro di base EBP)

retn (restituisce il controllo alla funzione chiamante)

sub_401000 endp (chiude il codice)