

S1015

malware

Analisyg

Presented by Gioele  
Giardina

# Indice

- Traccia
- Analisi librerie
- Sezioni malware
- Analisi costrutti
- Conclusioni

# 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 ? Fare anche una descrizione
2. Quali sono le sezioni di cui si compone il file eseguibile del malware?  
Fare anche una descrizione
3. Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:  
Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti)
4. Ipotizzare il comportamento della funzionalità implementata altri costrutti
5. Fare una tabella per spiegare il significato delle singole righe di codice

# Analisi Librerie

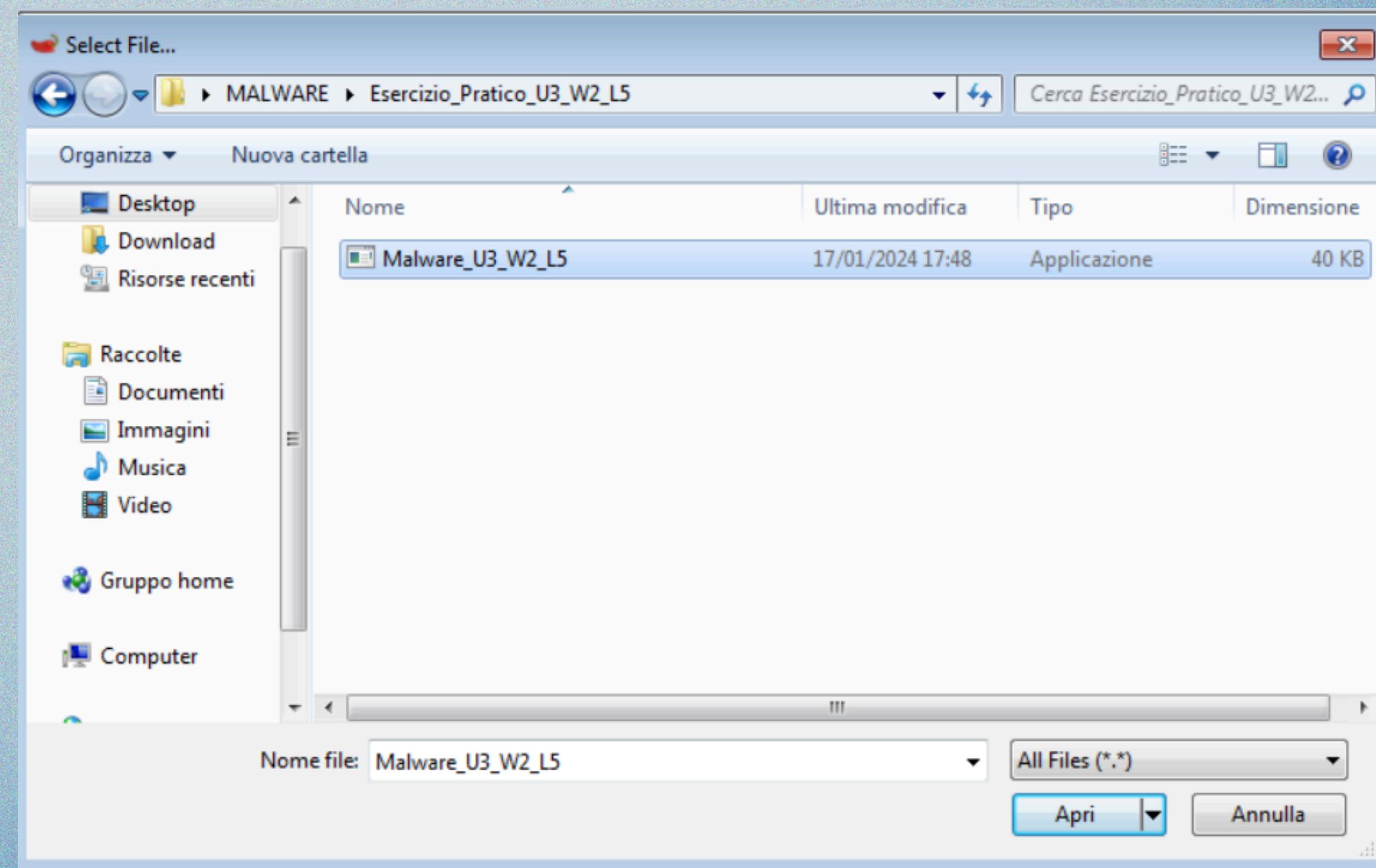
L'analisi delle librerie è una parte importante del processo di analisi del malware.

Questo processo è volto a identificare e comprendere le funzioni chiamate o importate da un malware durante la sua esecuzione, permettendoci di comprendere alcuni possibili funzionamenti del malware ancor prima di avviarlo.

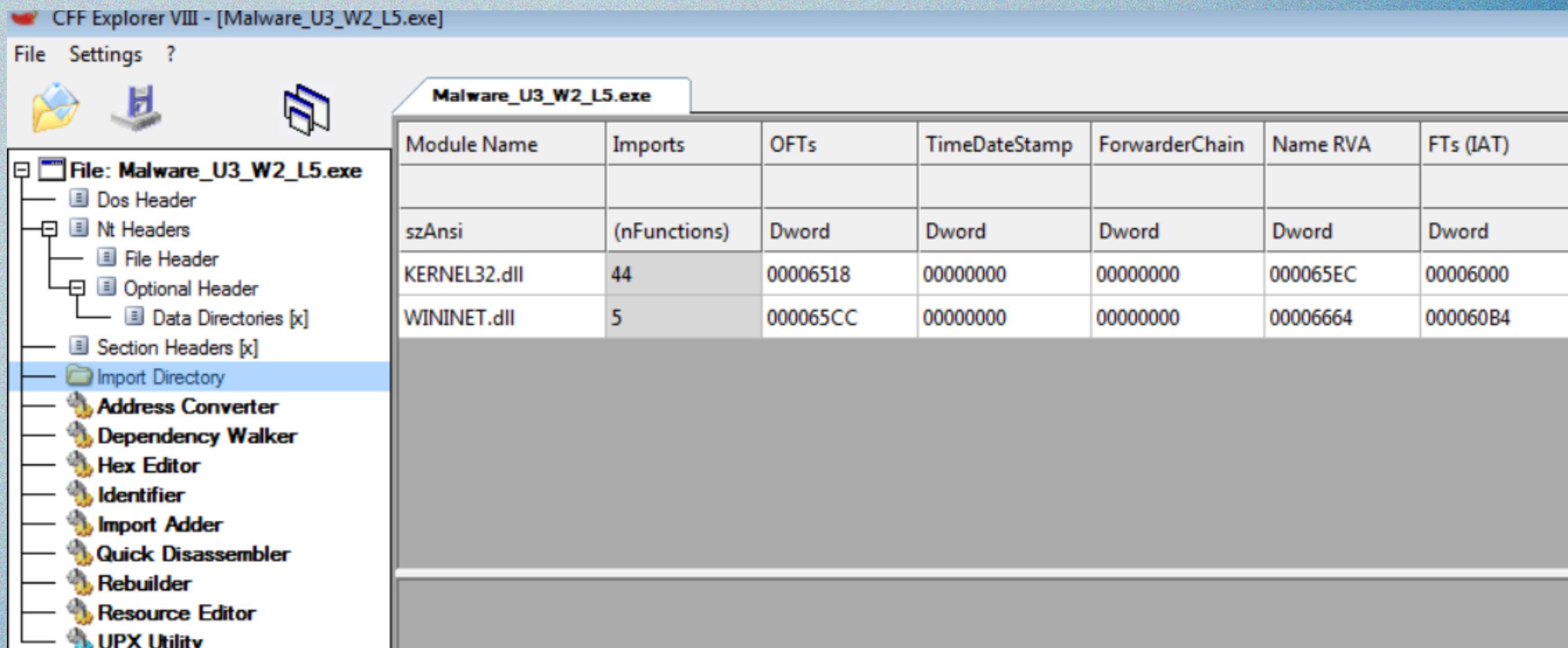
# CFF Explorer

Per l'analisi di oggi utilizzeremo CFF Explorer, un software gratuito e avanzato di esplorazione di file PE (Portable Executable). È utilizzato principalmente per analizzare e modificare file eseguibili in formato PE, come eseguibili Windows (EXE), dynamic link libraries (DLL) e altri formati correlati.

Una volta aperto CFF Explorer  
importiamo l'eseguibile del malware  
al suo interno e clicchiamo “Open”



Successivamente, tramite il menù a sinistra clicchiamo la voce “**import directory**” che mostrerà la schermata seguente come da immagine, le librerie importate da questo malware sono “**KERNEL32.dll**” e “**WININET.dll**”



# Approfondimento DLL

Un "**Dynamic Link Library**" o DLL è un tipo di file eseguibile in ambiente Windows che contiene codice e dati che possono essere utilizzati da più programmi contemporaneamente.

A differenza di un'applicazione autonoma, una DLL non può essere eseguita direttamente, ma fornisce funzionalità condivise che possono essere richiamate da diverse applicazioni.

# **Kernel32.dll**

- Fornisce funzioni di basso livello che gestiscono memoria, file, processi e altri aspetti fondamentali del sistema operativo.
- Include funzioni per la gestione dei processi, sincronizzazione, allocazione di memoria e la gestione delle eccezioni.
- È una delle DLL principali di Windows e molte altre DLL fanno riferimento a sue funzioni

# **Wininet.dll:**

- Offre funzioni per la gestione delle operazioni di rete, inclusi protocoli come HTTP, FTP e altri.
- È coinvolta nella gestione delle connessioni Internet e fornisce un'interfaccia per le operazioni di navigazione e download.
- Utilizzata da molti programmi per interagire con il Web e le risorse Internet.

# Sezione Malware

Il codice del malware in questione, come ogni eseguibile windows, è diviso in diverse sezioni ognuna con la sua funzione specifica

Sempre dallo stesso menù di CFF Explorer è possibile accedere alla voce “**Section Headers**” che fornisce informazioni dettagliate sulle varie sezioni del file eseguibile e queste informazioni includono, i nomi delle sezioni, gli indirizzi in memoria in cui sono allocate, le dimensioni delle sezioni, i permessi di accesso (come lettura, scrittura, esecuzione) e altre informazioni utili per comprendere la struttura del file e, di conseguenza, visualizzare le sezioni dell'eseguibile che sono le seguenti: **.text .rdata .data**

The screenshot shows the CFF Explorer interface with the title bar "CFF Explorer VIII - [Malware\_U3\_W2\_L5.exe]". The menu bar includes "File", "Settings", and "?". On the left, there's a tree view of the file structure under "File: Malware\_U3\_W2\_L5.exe", showing sections like Dos Header, Nt Headers, File Header, Optional Header, Data Directories, and Section Headers. The "Section Headers" node is selected. Below the tree is a list of tools: Address Converter, Dependency Walker, Hex Editor, Identifier, Import Adder, Quick Disassembler, Rebuilder, Resource Editor, and UPX Utility. The main pane displays a table titled "Malware\_U3\_W2\_L5.exe" with the following data:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	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

# Approfondimento sezioni

**.text:** La sezione "text" comprende le istruzioni, ovvero le linee di codice, che la CPU eseguirà una volta avviato il software. Di solito, questa è l'unica sezione di un file eseguibile che la CPU esegue, poiché tutte le altre sezioni contengono dati o informazioni di supporto.

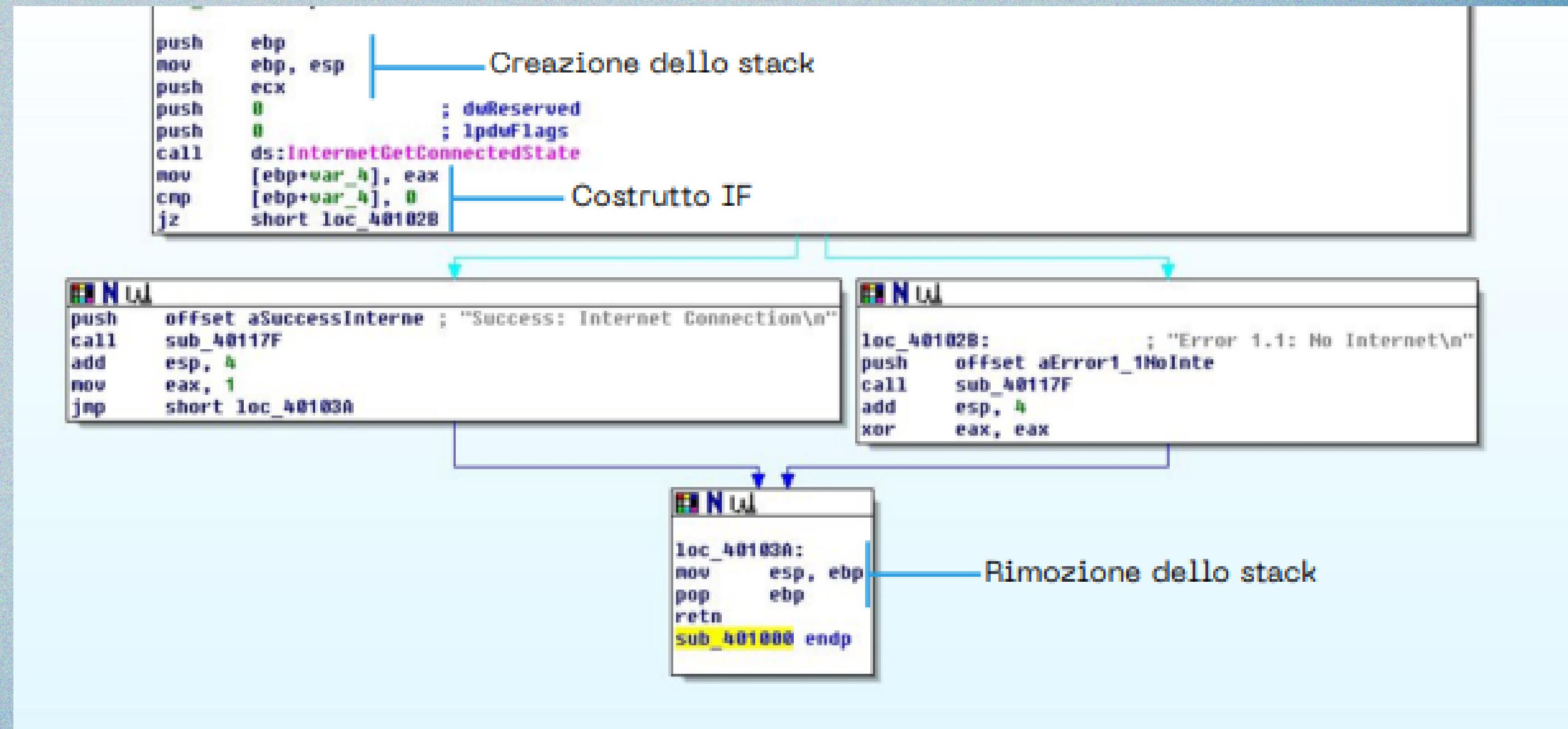
**.rdata:** La sezione "rdata" contiene informazioni sulle librerie e sulle funzioni importate o esportate dall'eseguibile.

**.data:** La sezione "data" contiene i dati o le variabili globali del programma eseguibile. Si noti che una variabile viene considerata globale quando non è definita all'interno del contesto di una funzione, ma è dichiarata globalmente e, di conseguenza, è accessibile da qualsiasi funzione dell'eseguibile.

# Analisi costrutti

Questa sezione si occupa di analizzare i costrutti assembly all'interno del malware, spiegando il loro funzionamento.

Come si può ben vedere nello screen, i costrutti trovati sono stati evidenziati



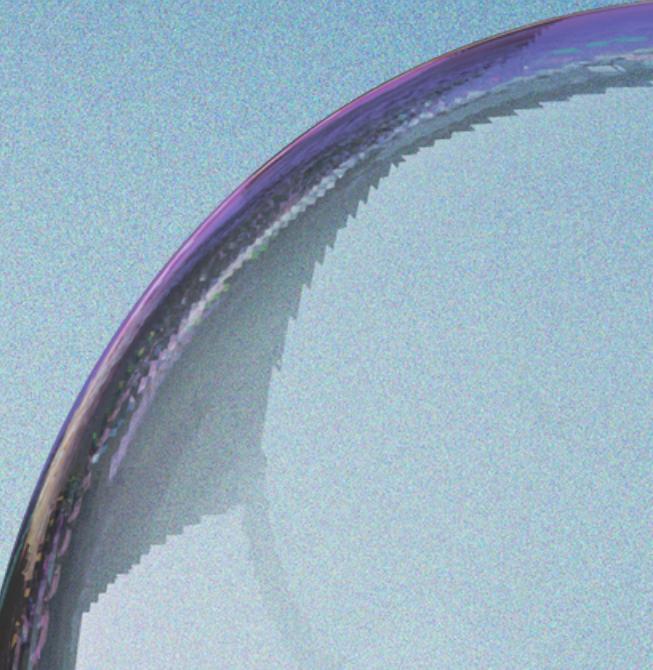
# Conclusioni

All'interno del codice assembly assegnato ci sono molti indizi sul funzionamento di questo malware. Esso infatti richiama una funzione della libreria wininet32, “**InternetGetConnectedState**” , ovvero controlla se la macchina in cui viene avviato l'eseguibile è connessa ad internet o meno.

Tramite il costrutto if posto subito dopo il malware da due possibili risultati a seconda della presenza di questa connessione:

se la connessione è presente passa come parametro “**Success: internet connection**” altrimenti “**Error 1.1: no internet**”

Con tutta probabilità queste stringhe vengono passate come parametri ad un secondo eseguibile, che verifica che la macchina bersaglio sia connessa ad internet prima di attuare un attacco/connessione malevola.



# *Analisi e spiegazione del codice Assembly x86*

## Prima parte di codice:

**push ebp** – Salva il base pointer corrente nello stack. Serve per mantenere il contesto della chiamata della funzione.

1. **mov ebp, esp** – Imposta il base pointer per puntare al top dello stack. Questo è utilizzato per iniziare un nuovo frame dello stack per questa funzione.

2. **push ecx** – Salva il valore corrente del registro ecx nello stack.

3. **push 0** – Mette il valore 0 nello stack, che sarà usato come argomento per la funzione InternetGetConnectedState.

4. **push 0** – Mette un altro 0 nello stack, anch'esso per InternetGetConnectedState.

5. **call ds:InternetGetConnectedState** – Chiama la funzione InternetGetConnectedState che verifica lo stato della connessione internet. I valori 0 precedentemente messi nello stack sono gli argomenti per questa funzione.

6. **mov [ebp+var\_4], eax** – Salva il risultato di InternetGetConnectedState (ritornato nel registro eax) in una variabile locale definita nel frame dello stack.

7. **cmp [ebp+var\_4], 0** – Confronta il valore salvato con 0 per determinare se la connessione è attiva o no.

8. **jz short loc\_40192B** – Salta all'indirizzo loc\_40192B se il risultato del confronto è zero (cioè se non c'è connessione).

## Parte intermedia del codice:

- **push offset aSuccessInterne** – Mette l'indirizzo di una stringa ("Success: Internet Connection\n") nello stack.
- **call sub\_40117F** – Chiama una subroutine che probabilmente stampa la stringa o esegue un'azione basata sul successo della connessione.
- **add esp, 4** – Aggiusta lo stack pointer per rimuovere l'argomento usato nella chiamata precedente.
- **mov eax, 1** – Imposta eax a 1, probabilmente indicando un risultato positivo.
- **jmp short loc\_40193A** – Salta all'indirizzo loc\_40193A, presumibilmente la fine della funzione.

Se il salto condizionale jz viene preso a causa della mancanza di connessione:

- **push offset aError1\_1NoInterne** – Mette l'indirizzo di un'altra stringa ("Error 1-1: No Internet\n") nello stack.
- **call sub\_40117F** – Chiama la stessa subroutine per gestire l'errore.
- **add esp, 4** – Pulisce lo stack rimuovendo l'argomento della chiamata.
- **xor eax, eax** – Imposta eax a 0, probabilmente indicando un risultato negativo o un errore.

## Parte finale del codice:

- **mov esp, ebp** – Ripristina il puntatore dello stack.
- **pop ebp** – Ripristina il base pointer al suo valore originale.
- **retn** – Ritorna dalla subroutine.

Questo codice gestisce sia il caso di successo che di fallimento della connessione internet e adotta azioni appropriate in base allo stato.