



# **Esercizio**

## **Week 24 Day 4**

**04/08/2024**

**Cybersecurity Analyst**

**Studente: Orazio Ciccozzi**

Sommario

Quesito esercizio ..... 3

Quanti parametri sono passati alla funzione Main()?..... 3

    Parametro ..... 3

    Funzione ..... 3

    Analisi del Malware ..... 4

Quante variabili sono dichiarate all'interno della funzione Main()? .....5

Quali sezioni sono presenti all'interno del file eseguibile? .....5

Quali librerie importa il Malware? .....7

Conclusioni .....10

## Quesito esercizio

Con riferimento al file eseguibile `Malware_Build_Week_U3`, rispondere ai seguenti quesiti utilizzando i tool e le tecniche apprese nelle lezioni teoriche:

1. Quanti parametri sono passati alla funzione `Main()`?
2. Quante variabili sono dichiarate all'interno della funzione `Main()`?
3. Quali sezioni sono presenti all'interno del file eseguibile?
4. Descrivete brevemente almeno 2 di quelle identificate
5. Quali librerie importa il Malware? Per ognuna delle librerie importate, fate delle ipotesi sulla base della sola analisi statica delle funzionalità che il Malware potrebbe implementare. Utilizzate le funzioni che sono richiamate all'interno delle librerie per supportare le vostre ipotesi.

## Quanti parametri sono passati alla funzione `Main()`?

### Parametro

In Assembly, un parametro di una funzione è un valore passato alla funzione quando viene chiamata. Solitamente, i parametri vengono passati attraverso lo stack.

**Ad esempio:**

***assembly***

***main:***

***push 0x08***  
***call my\_function***

***my\_function:***

***pop %eax***  
***...***  
***ret***

In questo esempio, **`0x08`** è un parametro passato alla funzione **`my_function`**.

### Funzione

In Assembly, una funzione è un insieme di istruzioni che esegue un compito specifico e può ricevere valori in ingresso, chiamati parametri, oltre a poter restituire un valore. Le funzioni in Assembly sono simili a quelle in altri linguaggi di programmazione, ma a causa della natura di basso livello dell'Assembly, la loro gestione può risultare più complessa.

Per chiamare una funzione in Assembly, si utilizza l'istruzione call **"nome funzione"**. I parametri sono solitamente passati tramite registri o tramite lo stack. Dopo che la funzione ha completato l'esecuzione, il controllo ritorna al punto del programma da cui è stata chiamata.

È importante notare che l'istruzione call posiziona l'indirizzo di ritorno nello stack, quindi eseguendo **pop %eax** si ottiene l'indirizzo di ritorno, non il parametro. Per accedere al parametro, si utilizza l'istruzione **mov eax, [esp+4]**, riferendosi alla posizione appropriata nello stack.

Avendo chiaro il concetto di funzione e dei suoi parametri, possiamo procedere con l'analisi.

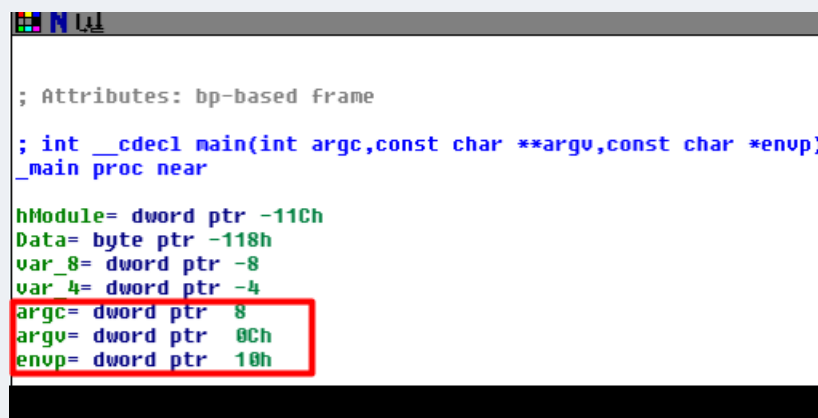
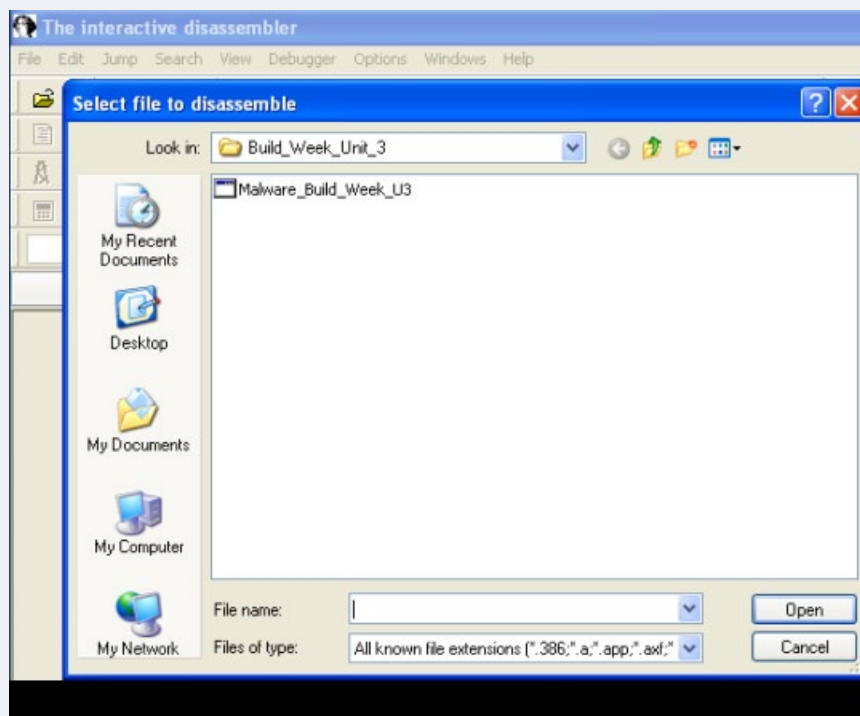
## Analisi del Malware

Usando IDA (Interactive Disassembler), possiamo verificare i parametri della funzione main. Nel disassembly, i parametri della funzione main sono evidenziati come segue:

assembly

**int \_\_cdecl main (parametro1, parametro2, parametro3) \_main proc near**

Dall'analisi, possiamo notare che la funzione main accetta tre parametri.



Con riferimento alla seconda figura, possiamo notare che i parametri della funzione main del codice sono 3, segnalati da un riquadro rosso.

## Quante variabili sono dichiarate all'interno della funzione Main()?

In Assembly, una variabile è fondamentalmente un'etichetta per uno spazio di memoria utilizzato per conservare dati che il programma deve manipolare.

Le variabili in Assembly vengono dichiarate nella sezione .data del programma, specificando lo spazio che occuperanno. Un esempio di dichiarazione di una variabile è il seguente:

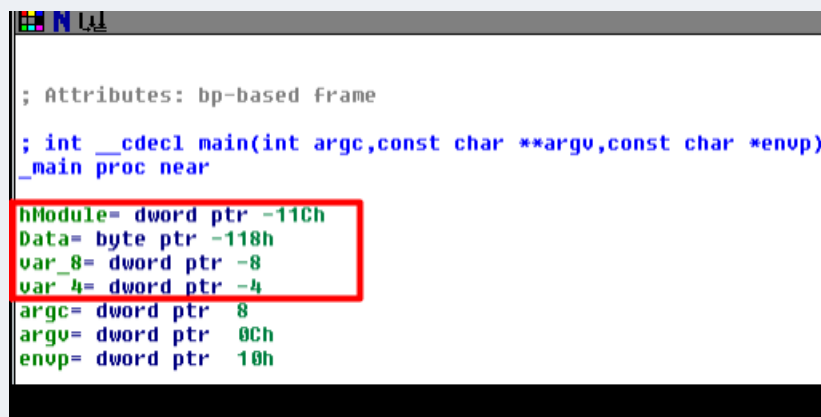
***"nome\_variabile" = dword ptr [x]***

Dove x è un valore intero negativo che indica l'offset della variabile. Se x è positivo, l'istruzione rappresenta la dichiarazione di un parametro.

È cruciale in Assembly eseguire operazioni principalmente attraverso i registri. Pertanto, per utilizzare una variabile, è consigliabile spostare prima il suo valore in un registro.

Per analizzare le variabili utilizzate dalla funzione main, possiamo usare lo stesso metodo impiegato per l'analisi dei parametri. Apriamo IDA, ci spostiamo nel pannello di disassemblaggio e analizziamo la funzione main.

Come mostrato nella seguente, possiamo vedere che le variabili della funzione main nel codice sono 4, indicate da un riquadro rosso.



```
; Attributes: bp-based frame
; int __cdecl main(int argc,const char **argv,const char *envp)
_main proc near
hModule= dword ptr -11Ch
Data= byte ptr -118h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
```

## Quali sezioni sono presenti all'interno del file eseguibile?

Prima di esaminare le sezioni presenti all'interno del file, è utile chiarire cosa si intende per "sezione".

### Sezioni di un file .exe

Le sezioni di un file .exe sono parti strutturate che contengono varie informazioni essenziali per la corretta esecuzione del programma. In un file PE (Portable Executable), le sezioni sono elementi fondamentali che memorizzano tipi specifici di dati e codice, giocando un ruolo cruciale nell'organizzazione e gestione delle funzionalità del programma.

Anche se esistono molte sezioni, è importante comprendere alcune di esse per l'analisi di base dei malware. Di seguito sono descritte le principali sezioni da considerare:

**.text:** Contiene il codice eseguibile del programma. Qui sono archiviate le istruzioni che indicano al computer cosa fare. Analizzare questa sezione può aiutare a identificare chiamate di funzione sospette, riferimenti a dati e potenziali attività dannose.

**.rdata:** Contiene dati di sola lettura come stringhe, costanti e risorse utilizzate dal programma. Il malware potrebbe incorporare stringhe offuscate o sfruttare vulnerabilità all'interno di questa sezione. Esaminarla può rivelare messaggi nascosti, dati di configurazione o indizi sulla sua origine.

**.data:** Memorizza variabili di dati inizializzate utilizzate dal programma durante l'esecuzione. Il malware potrebbe utilizzare questa sezione per archiviare dati temporanei o impostazioni di configurazione relative alle sue attività dannose. Analizzarla può aiutare a scoprire variabili nascoste, canali di comunicazione o codice iniettato.

**.rsrc:** Contiene le risorse del programma, come icone, immagini e menu. Sebbene apparentemente innocua, questa sezione potrebbe essere sfruttata dagli aggressori per nascondere codice dannoso o sfruttare vulnerabilità nelle librerie di analisi delle risorse. Esaminarla può rivelare payload nascosti o funzionalità mascherate.

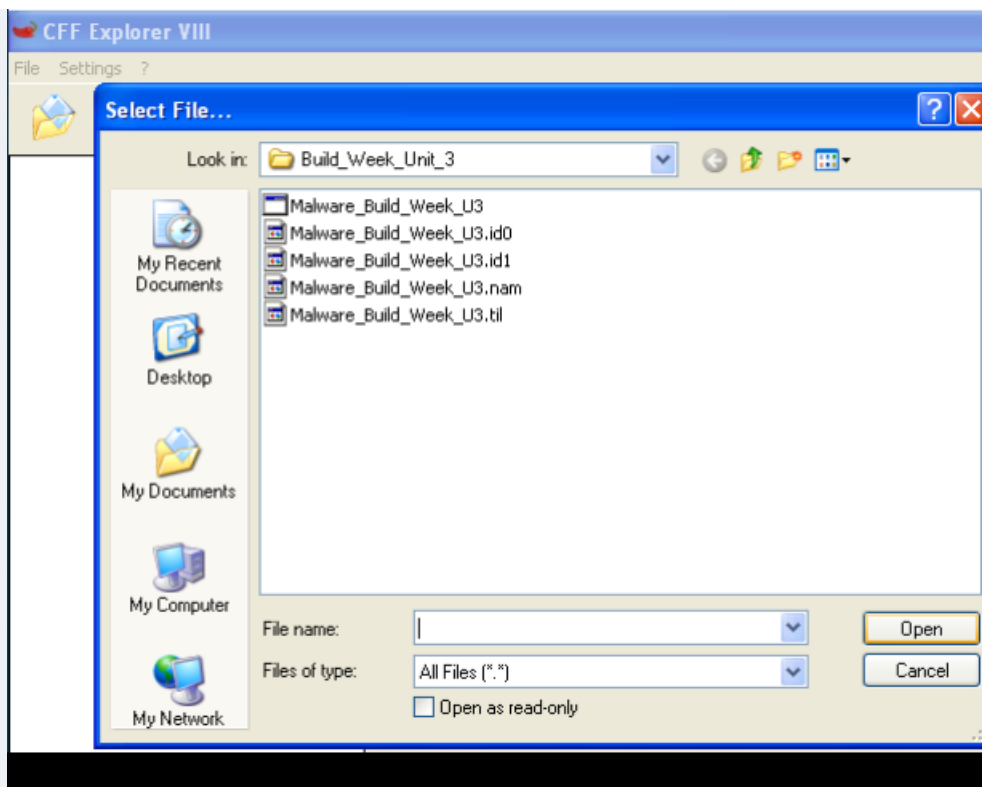
**.idata:** Contiene informazioni sulle funzioni importate da altre librerie (DLL). Analizzare questa sezione può aiutare a identificare importazioni sospette o inaspettate, che potrebbero suggerire le capacità e le dipendenze del malware.

**.edata:** Memorizza variabili di dati non inizializzate, a volte utilizzate per conservare dati di configurazione o valori temporanei nel malware.

**.reloc:** Contiene informazioni di rilocazione per regolare gli indirizzi dopo il caricamento del programma in memoria. Il malware potrebbe utilizzare questa sezione per l'iniezione di codice o tecniche anti-analisi.

Per analizzare le sezioni presenti nel malware, abbiamo utilizzato CFF Explorer, un tool che permette di esaminare le sezioni di un eseguibile. Avviamo CFF Explorer cliccando due volte sulla sua icona. Una volta avviato, cliccando sull'icona a forma di cartella in alto a sinistra (o selezionando File -> Open dal menu), si aprirà una finestra di dialogo per selezionare il file da analizzare. Navigando fino alla directory contenente il file `Malware_Build_Week_U3`, clicchiamo su Open per importarlo.

Dopo l'acquisizione del file, CFF Explorer mostrerà un menu di operazioni sulla parte sinistra dello schermo. Per visualizzare le sezioni presenti, scegliamo la voce "Section Headers".



Le sezioni importate sono mostrate nella Figura seguente ; ogni sezione è stata precedentemente approfondita. Di particolare rilievo è la sezione .rsrc, che solitamente è presente quando si analizza un tipo di malware chiamato dropper.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumb...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00005646	00001000	00006000	00001000	00000000	00000000	0000	0000	60000020
.rdata	000009AE	00007000	00001000	00007000	00000000	00000000	0000	0000	40000040
.data	00003EA8	00008000	00003000	00008000	00000000	00000000	0000	0000	C0000040
.rsrc	00001A70	0000C000	00002000	0000B000	00000000	00000000	0000	0000	40000040

## Quali librerie importa il Malware?

Come sempre, prima di procedere con l'analisi, spieghiamo cosa si intende per libreria.

### Cos'è una Libreria API Win32?

Le funzioni Win32 API sono un insieme di funzioni fornite da Microsoft che permettono alle applicazioni di interagire con il sistema operativo Windows. Queste funzioni sono contenute in librerie dinamiche (DLL), come kernel32.dll, user32.dll e gdi32.dll, e offrono una vasta gamma di funzionalità, tra cui l'accesso diretto alle funzionalità di sistema e all'hardware.

Per esempio, si può utilizzare una funzione Win32 API per creare una finestra, leggere un

file o inviare un messaggio di rete. Queste funzioni possono essere chiamate da qualsiasi linguaggio di programmazione che supporta l'interfaccia con il codice C.

Tuttavia, è importante ricordare che le API Win32 sono di basso livello. Molti sviluppatori preferiscono utilizzare linguaggi di programmazione di alto livello, come C# o Java, che offrono astrazioni di più alto livello rispetto alle API di basso livello come Win32.

### Cos'è una DLL?

Un file DLL, o Libreria di collegamento dinamico, è un tipo di file che contiene codice e dati che possono essere utilizzati da più programmi nello stesso momento. Questi file vengono caricati nel programma che li utilizza mentre il programma è in esecuzione.

Le DLL permettono di condividere il codice tra diversi programmi, evitando così la duplicazione del codice e risparmiando memoria. Ad esempio, se due programmi utilizzano la stessa funzione per stampare un documento, quella funzione può essere inclusa in una DLL e entrambi i programmi possono accedere a quella funzione attraverso la DLL.

Un altro vantaggio delle DLL è che permettono di aggiornare facilmente le funzioni o i dati. Se una funzione in una DLL viene migliorata o corretta, tutti i programmi che utilizzano quella DLL beneficeranno dell'aggiornamento.

Tuttavia, l'uso delle DLL può presentare delle sfide in termini di compatibilità e versionamento, poiché un programma può dipendere da una versione specifica di una DLL.

Come già visto nel passaggio precedente, anche per la consultazione delle librerie possiamo utilizzare CFF Explorer. La voce che andremo a selezionare dal menù, questa volta, sarà però Import Directory.

Come possiamo vedere in figura seguente, le librerie importate sono KERNEL32.dll e ADVAPI32.dll.

Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
000076D0	N/A	00007500	00007504	00007508	0000750C	00007510
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

KERNEL32.dll è un elemento vitale del sistema operativo Windows. È una libreria di collegamento dinamico (DLL), che è una raccolta di funzioni utilizzate da vari programmi sul tuo computer.

Nel dettaglio, le operazioni che gestisce KERNEL32.dll sono:

- **Gestione della memoria:** KERNEL32.dll ha un ruolo fondamentale nella gestione della memoria in Windows. Quando Windows si avvia, KERNEL32.dll viene caricato in una zona protetta della memoria per evitare interferenze con altri programmi.
- **Input/Output (I/O):** KERNEL32.dll gestisce le operazioni di input e output, che sono fondamentali per la comunicazione tra il computer e l'hardware esterno, come la



tastiera, il mouse o la stampante.

- **Gestione dei file:** KERNEL32.dll è anche responsabile della gestione dei file. Questo include la creazione, la lettura e la scrittura di file sul disco rigido.
- **Comunicazione con l'hardware:** KERNEL32.dll facilita la comunicazione tra i programmi e l'hardware del computer. Questo è essenziale per le operazioni come l'invio di comandi alla CPU o la lettura dei dati dalla RAM.

Se KERNEL32.dll manca o è danneggiato, potrebbero riscontrarsi vari problemi, poiché molti programmi dipendono da esso per funzionare correttamente. Ad esempio, potresti vedere messaggi di errore che indicano che KERNEL32.dll non può essere trovato o inizializzato.

Dalla Figura seguente,

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0000769E	N/A	000074EC	000074F0	000074F4	000074F8	000074FC
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00007632	00007632	0295	SizeofResource
00007644	00007644	01D5	LockResource
00007654	00007654	01C7	LoadResource

possiamo notare che dalla libreria Kernel32.dll vengono importate ben 51 funzioni, tuttavia quelle più interessanti sono le 3 evidenziate in rosso, vediamole nel dettaglio:

- **SizeofResource:** Questa funzione recupera la dimensione, in byte, della risorsa specificata.
- **LockResource:** Questa funzione recupera un puntatore alla risorsa specificata in memoria. Nonostante il nome, LockResource non blocca effettivamente la memoria; viene usato solo per ottenere un puntatore alla memoria contenente i dati della risorsa.
- **LoadResource:** Questa funzione recupera un handle che può essere usato per ottenere un puntatore al primo byte della risorsa specificata in memoria

ADVAPI32.dll è una libreria di collegamento dinamico (DLL) fondamentale nei sistemi Windows, che fornisce funzioni essenziali per sicurezza, autenticazione e controllo degli accessi. Agisce come ponte tra le applicazioni e i meccanismi di sicurezza sottostanti del sistema operativo.

Tra le funzionalità, di particolare importanza è il controllo dell'accesso al

registro, che permette la gestione delle autorizzazioni di accesso per le chiavi e i valori del registro, garantendo la sicurezza dei dati.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000076D0	N/A	00007500	00007504	00007508	0000750C	00007510
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000076AC	000076AC	0186	RegSetValueExA
000076BE	000076BE	015F	RegCreateKeyExA

Come possiamo notare dalla figura qui sopra le funzioni utilizzate, importate da ADVAPI32.DLL sono:

- RegSetValueExA : Questa funzione imposta i dati e il tipo di un valore specificato in una chiave del Registro di sistema.
- RegCreateKeyExA : Questa funzione crea una specifica chiave del Registro di sistema. Se la chiave esiste già, la funzione la apre.

## Conclusioni

Guardando nel dettaglio le funzioni importate dalle singole librerie, possiamo ipotizzare che il malware modifichi il registro di sistema di Windows, creando delle nuove chiavi e assegnando loro un valore, in questo modo ipotizziamo che il malware riesca ad ottenere la persistenza nel nostro sistema operativo.

Inoltre, tra le funzioni importate dalla libreria Kernel32.dll troviamo le più comuni per la realizzazione di un dropper, vediamo nel dettaglio cosa fa.