

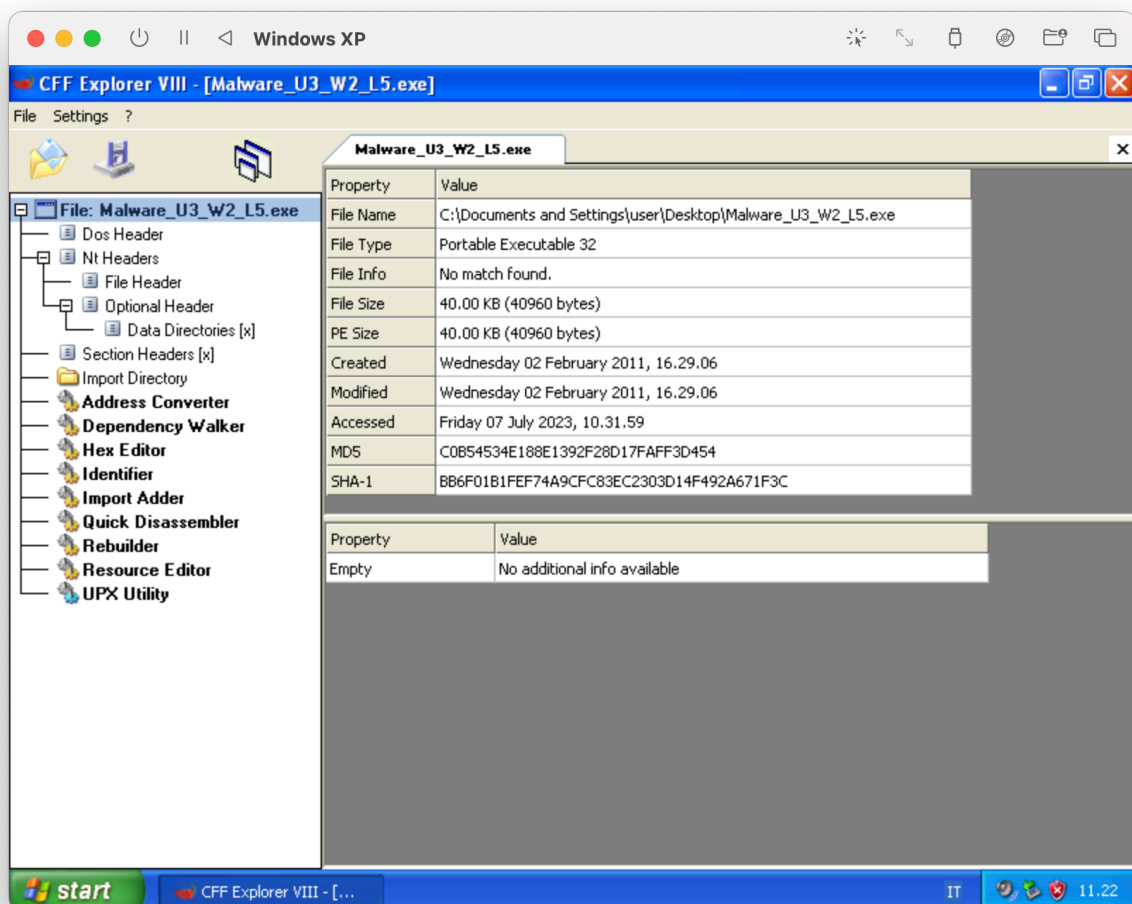
REPORT SETTIMANA 10

Esercizio 1

Nel primo esercizio veniva richiesto di identificare le librerie, insieme di funzioni pronte all'utilizzo se correttamente richiamate all'interno del codice, importate dal malware.

Per condurre questa analisi, il tool **CFF Explorer**, che permette di esaminare l'header del formato PE (Portable Executable). Questo formato è ampiamente utilizzato dal sistema operativo Windows per i file eseguibili e contiene le informazioni necessarie affinché il sistema operativo possa gestire il codice del file. All'interno dell'header del formato PE troviamo:

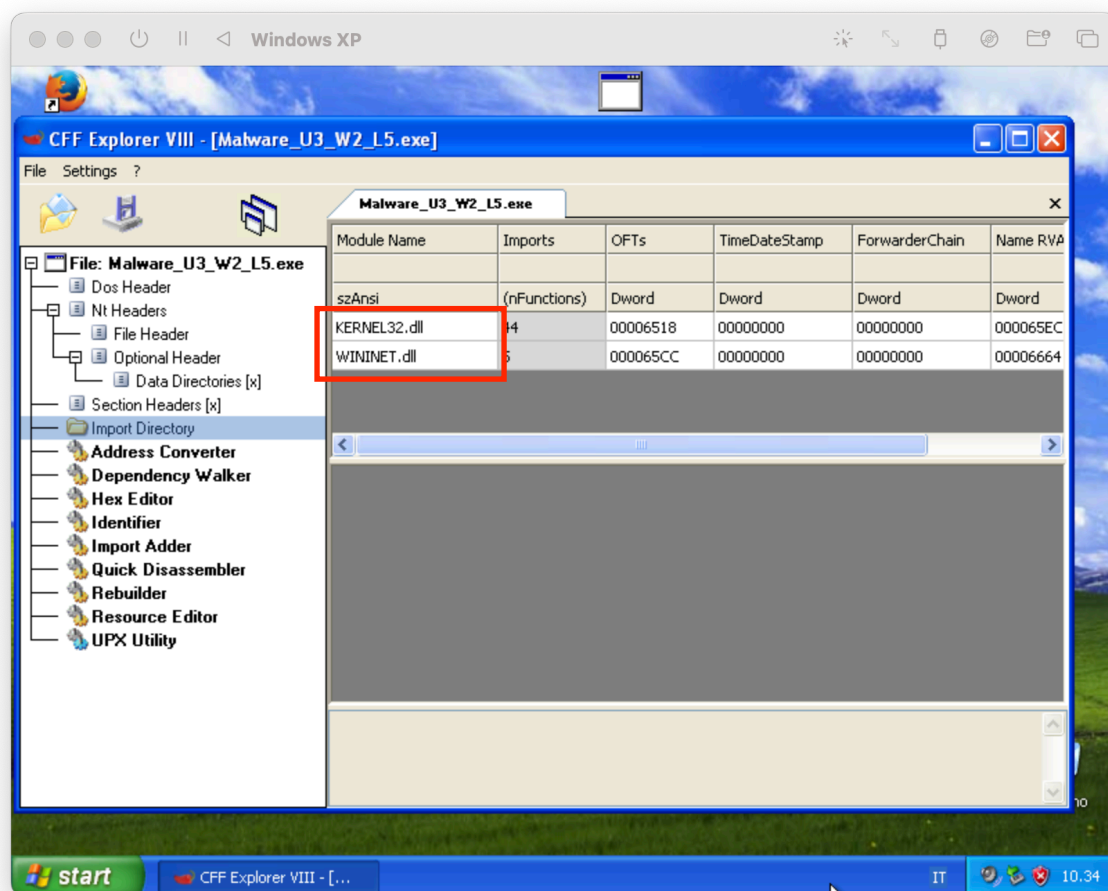
- Un elenco delle librerie importate e delle funzioni richieste dall'eseguibile.
- Eventuali funzioni esportate dall'eseguibile, ovvero messe a disposizione di altri programmi.
- Le sezioni che compongono il software.



Nella schermata iniziale CFF ci restituisce alcune informazioni tra cui le dimensioni, la data di creazione ed il suo hash nei formati MD5 e SHA-1, che possono essere utilizzati in seguito per cercare informazioni in database come VirusTotal.

Spostandoci nella sezione “**Import Directory**” possiamo vedere le librerie importate dal malware che in questo caso sono due:

- **KERNEL32.dll** : La libreria KERNEL32.dll è una delle librerie principali del sistema operativo Windows. Contiene numerose funzioni essenziali per il funzionamento del sistema operativo e per l'esecuzione di programmi come per esempio CreateProcess, OpenProcess, ReadFile, WriteFile, LoadLibrary, GetProcAddress ...
- **WININET.dll** : La libreria WININET.dll è una delle librerie fondamentali del sistema operativo Windows. Essa fornisce una serie di funzioni per la gestione delle operazioni di rete, inclusa la comunicazione via Internet come per esempio InternetOpen, InternetOpenUrl, InternetReadFile, InternetWriteFile ...

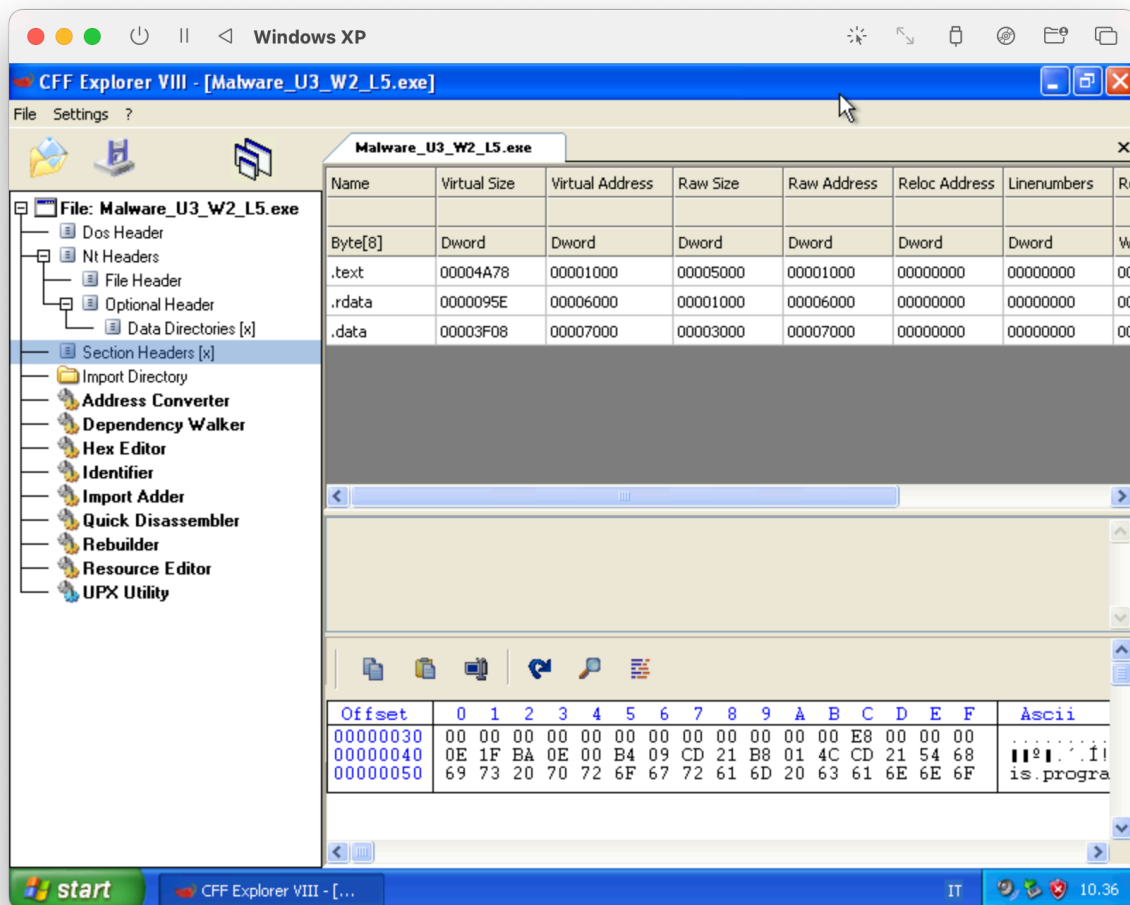


Cliccando poi sulle librerie è possibile visionare le funzioni utilizzate dal malware, per esempio nella libreria **KERNEL32.dll** vengono utilizzate le funzioni “**GetProcAddress**” e “**LoadLibraryA**” per richiamare la libreria solo quando serve ed essere meno rilevabile.

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006880	00006880	01A2	HeapReAlloc
0000688E	0000688E	013E	GetProcAddress
000068A0	000068A0	01C2	LoadLibraryA
000068B0	000068B0	011A	GetLastError
000068C0	000068C0	00AA	FlushFileBuffers
000068D4	000068D4	026A	SetFilePointer
00006950	00006950	001B	CloseHandle

Esercizio 2

Nel secondo esercizio veniva richiesto di individuare le sezioni di cui era composto il malware.

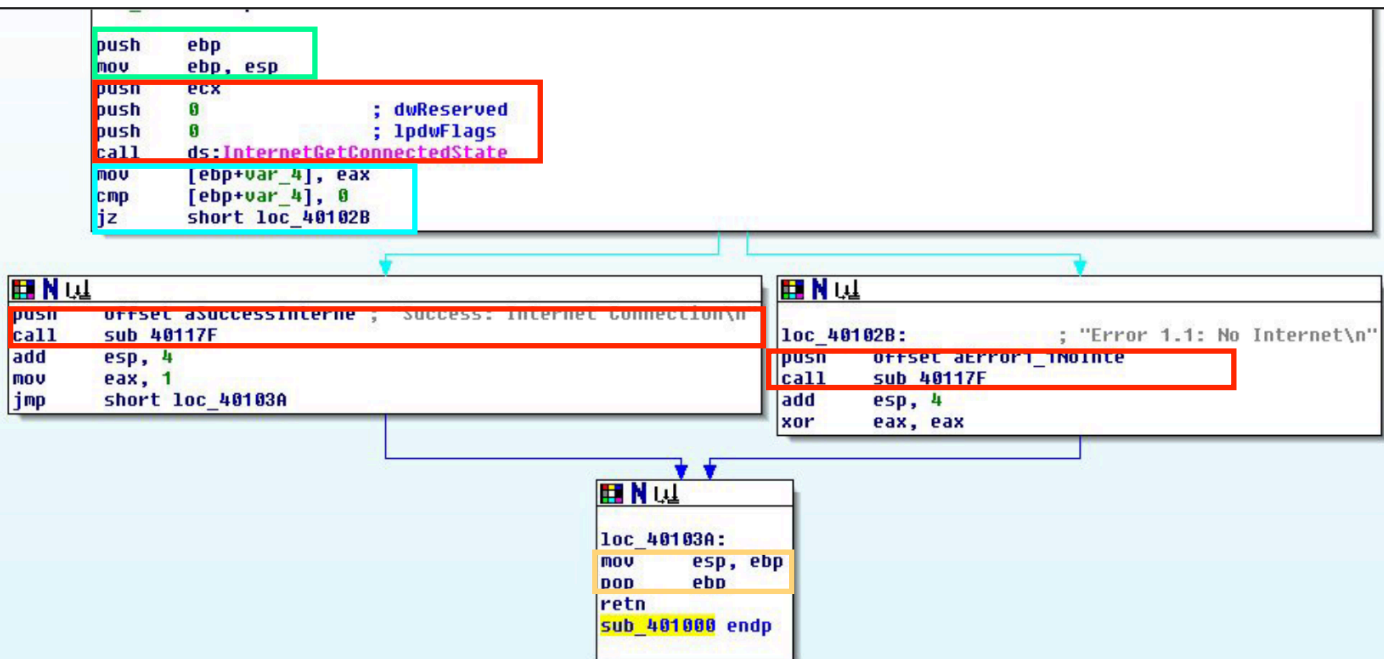


Mi sono quindi spostato nella sezione “**Section Headers**” e ho trovato 3 sezioni del malware:

- **.text** : contiene istruzioni (righe di codice) che la CPU eseguirà quando il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, perché tutte le altre sezioni contengono dati o informazioni a supporto
- **.rdata** : contiene informazioni sulle librerie e le funzioni importate ed esportate dall'eseguibile
- **.data** : contiene dati/variabili globali del programma eseguibile, che devono essere quindi disponibili da qualsiasi parte del programma (una variabile è globale quando non è definita all'interno di una funzione, ma è globalmente dichiarata ed è quindi accessibile da qualsiasi funzione dell'eseguibile)

Esercizio 3

Nel terzo esercizio veniva richiesto di identificare i costrutti noti nel codice Assembly in figura:



- In verde: Creazione dello stack.
- In rosso: Chiamata di Funzione.
- In azzurro: Istruzione IF.
- In giallo: Chiusura dello stack.

Esercizio 4

Nel quarto esercizio veniva richiesto di ipotizzare il comportamento della funzionalità implementata.

Il codice Assembly analizzato crea dapprima uno stack per le variabili locali e verifica se la macchina ha accesso a Internet utilizzando la funzione **"InternetGetConnectedState"**. In base al risultato della verifica, viene stampato un feedback sullo stato della connessione e se la connessione a Internet è attiva, il programma esegue alcune operazioni, incluso l'aumento dello stack, e poi salta incondizionatamente all'indirizzo di memoria 40103A. A quel punto, il programma chiude lo stack e ritorna alla funzione chiamante.

Se la connessione a Internet è assente, il valore del registro `eax` viene impostato a 0 utilizzando l'operazione XOR (`xor eax, eax`).

Una porzione di codice del genere potrebbe appartenere a vari tipi di malware che sfruttano la connettività internet per il loro funzionamento tra cui trojan, worm, spyware e botnet.

Esercizio 5

Nel quinto esercizio veniva richiesto di spiegare le istruzioni Assembly.

push ebp: Salva l'Extended Base Ponter in cima allo stack.

mov ebp, esp: Copia il valore dell'Extended stack pointer (ESP) nel Extended Base Ponter (EBP). Questa istruzione crea un nuovo frame di stack per la funzione corrente.

push ecx: Salva il valore attuale del registro ECX nello stack.

push 0: Mette nello stack il valore 0.

push 0: Mette nello stack il valore 0.

call ds:InternetGetConnectedState: Esegue una chiamata di funzione alla funzione "InternetGetConnectedState" presa dalla sezione dei dati del segmento di dati (DS).

mov [ebp+var_4], eax: Copia il valore presente in EAX nella posizione di memoria [EBP+var_4].

cmp [ebp+var_4], 0: Confronta il valore nella posizione di memoria [EBP+var_4] con 0.

jz short loc_40102B: Salta a "loc_40102B" (etichetta) se il confronto precedente è uguale a zero (zero flag impostato).

push offset sSuccessInterne: Mette nello stack l'indirizzo (offset) di una stringa "Success: Internet Connection\n".

call sub_40117F: Esegue una chiamata di funzione alla funzione "sub_40117F".

add esp, 4: Aggiunge 4 al registro dello stack pointer (ESP) per "liberare" lo spazio nello stack utilizzato dai parametri della chiamata di funzione.

mov eax, 1: Copia il valore 1 nel registro EAX.

jmp short loc_40103A: Salta incondizionatamente a "loc_40103A" (etichetta).

loc_40102B: Etichetta per il punto di destinazione della condizione di salto se il confronto precedente era uguale a zero.

push offset aError1_1NoInte: Mette nello stack l'indirizzo (offset) di una stringa "Error 1.1: No Internet\n".

call sub_40117F: Esegue una chiamata di funzione alla funzione "sub_40117F" per gestire l'output del messaggio di errore.

add esp, 4: Aggiunge 4 al registro dello stack pointer (ESP) per "liberare" lo spazio nello stack utilizzato dai parametri della chiamata di funzione.

xor eax, eax: Esegue un'operazione XOR tra il registro EAX e se stesso, impostando il registro EAX a zero.

loc 40103A: Etichetta per il punto di destinazione dopo il salto condizionale.

mov esp, ebp: Ripristina il valore originale dello stack pointer (ESP) copiando il valore di EBP in ESP.

pop ebp: Ripristina il valore originale del registro base dell'allocazione delle basi (EBP) recuperandolo dallo stack.

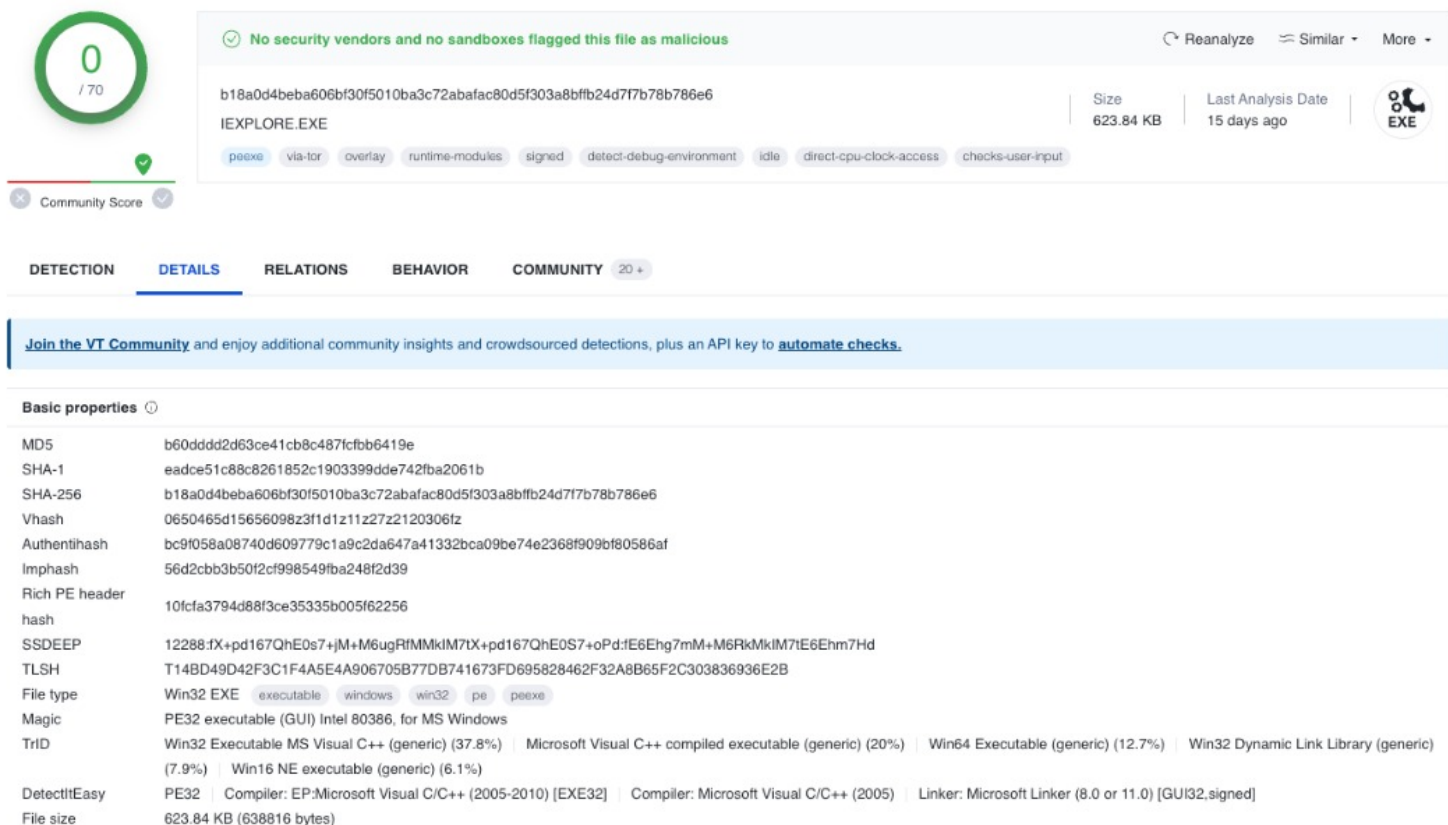
retn: Restituisce il controllo al chiamante della funzione corrente.

sub_401000 endp: Questa riga indica la fine della definizione della funzione denominata "sub_401000".

Esercizio Bonus

Nell'esercizio bonus veniva richiesto di immedesimarsi in membro senior del SOC per convincere un dipendente della non malevolenza del file "**IEXPLORE.EXE**" contenuto nella cartella "*C:\Program Files\Internet Explorer*".

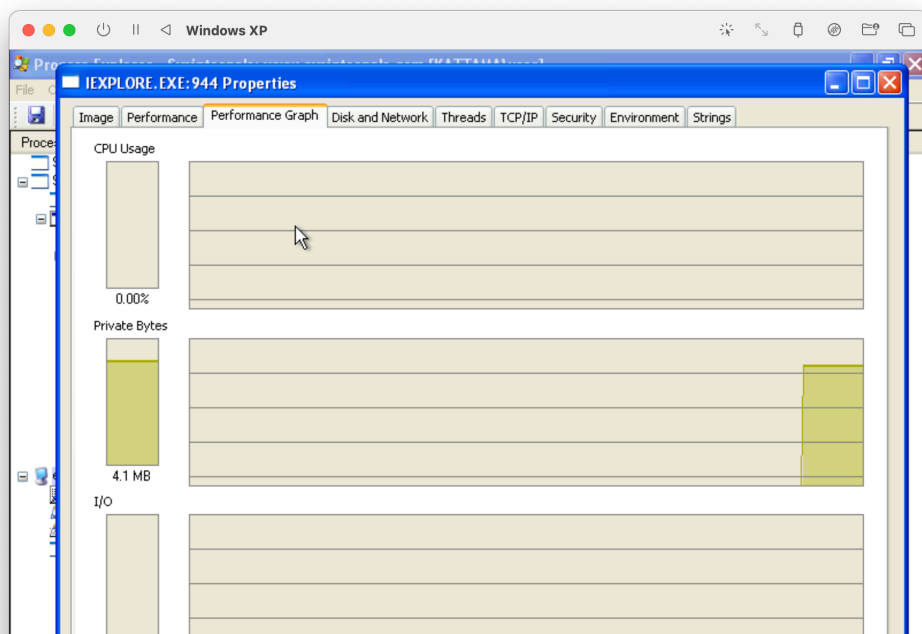
Come prima cosa ho analizzato l'hash del programma, ricavato con **md5deep**, su **VirusTotal** e come possiamo notare nella figura sottostante il programma è segnalato come completamente non malevolo.



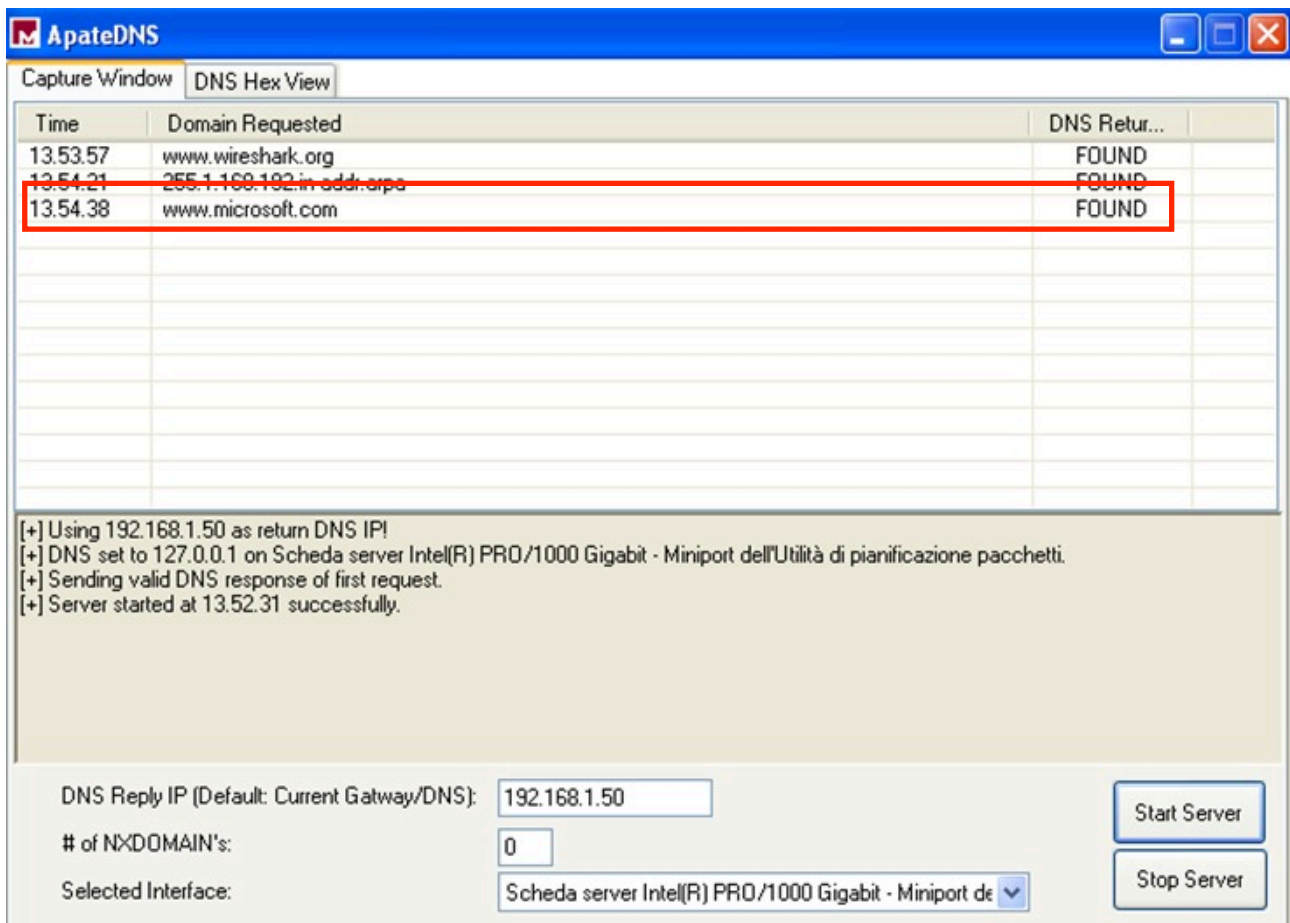
The screenshot shows the VirusTotal analysis interface for the file **IEXPLORE.EXE**. The file is identified by its MD5 hash: `b18a0d4beba606bf30f5010ba3c72abafac80d5f303a8bffb24d717b78b786e6`. The file size is 623.84 KB, and the last analysis was performed 15 days ago. The analysis shows that no security vendors or sandboxes flagged this file as malicious. The file is categorized as a Win32 EXE, executable, windows, win32, pe, peexe. The file type is PE32 executable (GUI) Intel 80386, for MS Windows. The file is identified as a Win32 Executable MS Visual C++ (generic) (37.8%), Microsoft Visual C++ compiled executable (generic) (20%), Win64 Executable (generic) (12.7%), Win32 Dynamic Link Library (generic) (7.9%), and Win16 NE executable (generic) (6.1%). The file is detected by DetectItEasy as PE32, compiled by EP:Microsoft Visual C/C++ (2005-2010) [EXE32], and linked by Microsoft Visual C/C++ (2005) [GUI32.signed]. The file size is 623.84 KB (638816 bytes).

Property	Value
MD5	b60ddd2d63ce41cb8c487fcfb6419e
SHA-1	eadce51c88c8261852c1903399dde742fba2061b
SHA-256	b18a0d4beba606bf30f5010ba3c72abafac80d5f303a8bffb24d717b78b786e6
Vhash	0650465d1565609823f1d1z11z27z2120306fz
Authentihash	bc9f058a08740d609779c1a9c2da647a41332bca09be74e2368f909bf80586af
Imphash	56d2cbb3b50f2cf998549fba248f2d39
Rich PE header hash	10fcfa3794d88f3ce35335b005f62256
SSDEEP	12288:fX+pd167QhE0s7+jM+M6ugRfMMkIM7IX+pd167QhE0S7+oPd:fE6Ehg7mM+M6RkMkIM7IE6Ehm7Hd
TLSH	T14BD49D42F3C1F4A5E4A906705B77DB741673FD695828462F32A8B65F2C303836936E2B
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (37.8%) Microsoft Visual C++ compiled executable (generic) (20%) Win64 Executable (generic) (12.7%) Win32 Dynamic Link Library (generic) (7.9%) Win16 NE executable (generic) (6.1%)
DetectItEasy	PE32 Compiler: EP:Microsoft Visual C/C++ (2005-2010) [EXE32] Compiler: Microsoft Visual C/C++ (2005) Linker: Microsoft Linker (8.0 or 11.0) [GUI32.signed]
File size	623.84 KB (638816 bytes)

Dopodiché ho utilizzato **ProcessExplorer** e attraverso il grafico delle performance ho notato che non consumava energia o rete in quantità anomale, altro segno che il programma non è malevolo.



Infine ho utilizzato **ApateDNS** per simulare un server DNS e intercettare tutte le richieste del malware verso domini internet e come possiamo notare, oltre alle prime due richieste che sono state fatte da **Wireshark** (che non funziona) l'unica richiesta fatta da **IEXPLORER.EXE** è la richiesta verso la homepage **microsoft.com**.



The screenshot shows the ApateDNS application window. It has a title bar with the name 'ApateDNS' and standard window controls. Below the title bar are two tabs: 'Capture Window' and 'DNS Hex View'. The 'Capture Window' tab is active, displaying a table of DNS requests. The table has three columns: 'Time', 'Domain Requested', and 'DNS Retur...'. The first three rows are highlighted with a red border. The first row shows a request for 'www.wireshark.org' at time 13.53.57 with a 'FOUND' status. The second row shows a request for '255.1.168.192.in-addr.arpa' at time 13.54.21 with a 'FOUND' status. The third row shows a request for 'www.microsoft.com' at time 13.54.38 with a 'FOUND' status. Below the table is a log area with several status messages. At the bottom of the window, there are input fields for 'DNS Reply IP (Default: Current Gateway/DNS):' set to '192.168.1.50', '# of NXDOMAIN's:' set to '0', and 'Selected Interface:' set to 'Scheda server Intel(R) PRO/1000 Gigabit - Miniport de'. There are also 'Start Server' and 'Stop Server' buttons.

Time	Domain Requested	DNS Retur...
13.53.57	www.wireshark.org	FOUND
13.54.21	255.1.168.192.in-addr.arpa	FOUND
13.54.38	www.microsoft.com	FOUND

[+] Using 192.168.1.50 as return DNS IP!
[+] DNS set to 127.0.0.1 on Scheda server Intel(R) PRO/1000 Gigabit - Miniport dell'Utilità di pianificazione pacchetti.
[+] Sending valid DNS response of first request.
[+] Server started at 13.52.31 successfully.

DNS Reply IP (Default: Current Gateway/DNS): 192.168.1.50
of NXDOMAIN's: 0
Selected Interface: Scheda server Intel(R) PRO/1000 Gigabit - Miniport de

Start Server
Stop Server