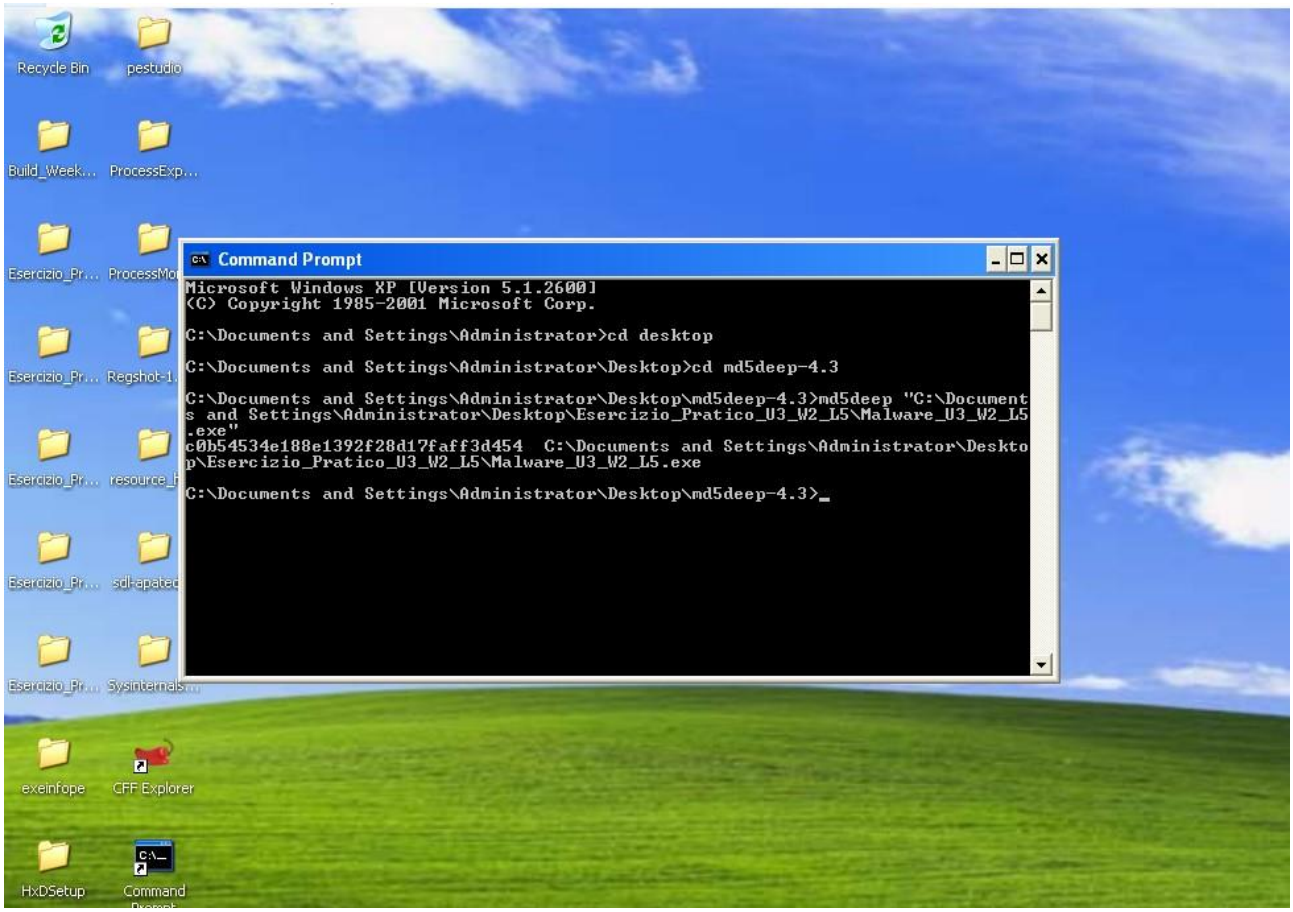
MALWARE_U3_W2_L5

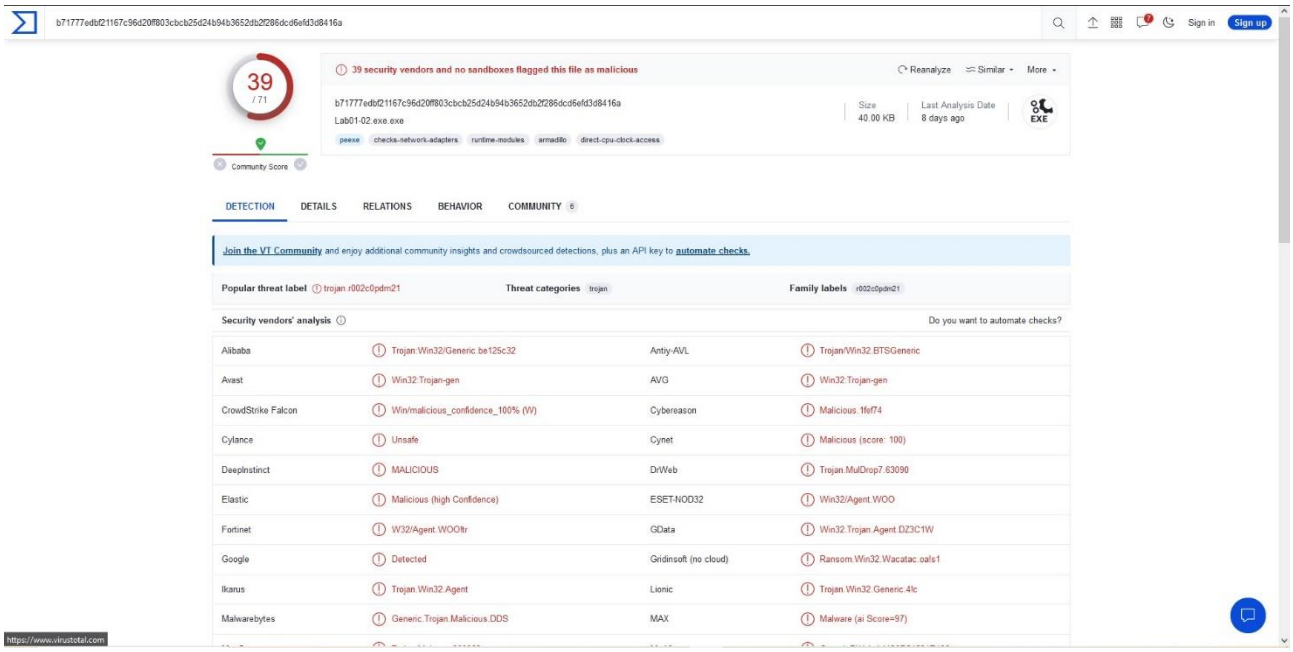
Dobbiamo studiare un potenziale malware. Per farlo andremo ad effettuare diversi tipi di analisi per cercare più informazioni possibili e poi andremo a scremare e selezionare quelle che ci interessano.

ANALISI STATICA BASICA

Sono diverse le operazioni che possiamo effettuare su questo eseguibile. Potremmo partire da un veloce check sul web caricando il file su virustotal, ma siamo in una macchina senza connessione per questioni di sicurezza. Procediamo dunque con le operazioni offline sul file, come ad esempio il controllo della firma o *hash* del file usando **md5deep**. Ci spostiamo nella cartella dove è contenuta l'utility e diamo il comando <md5deep "percorso assoluto file"> da prompt.

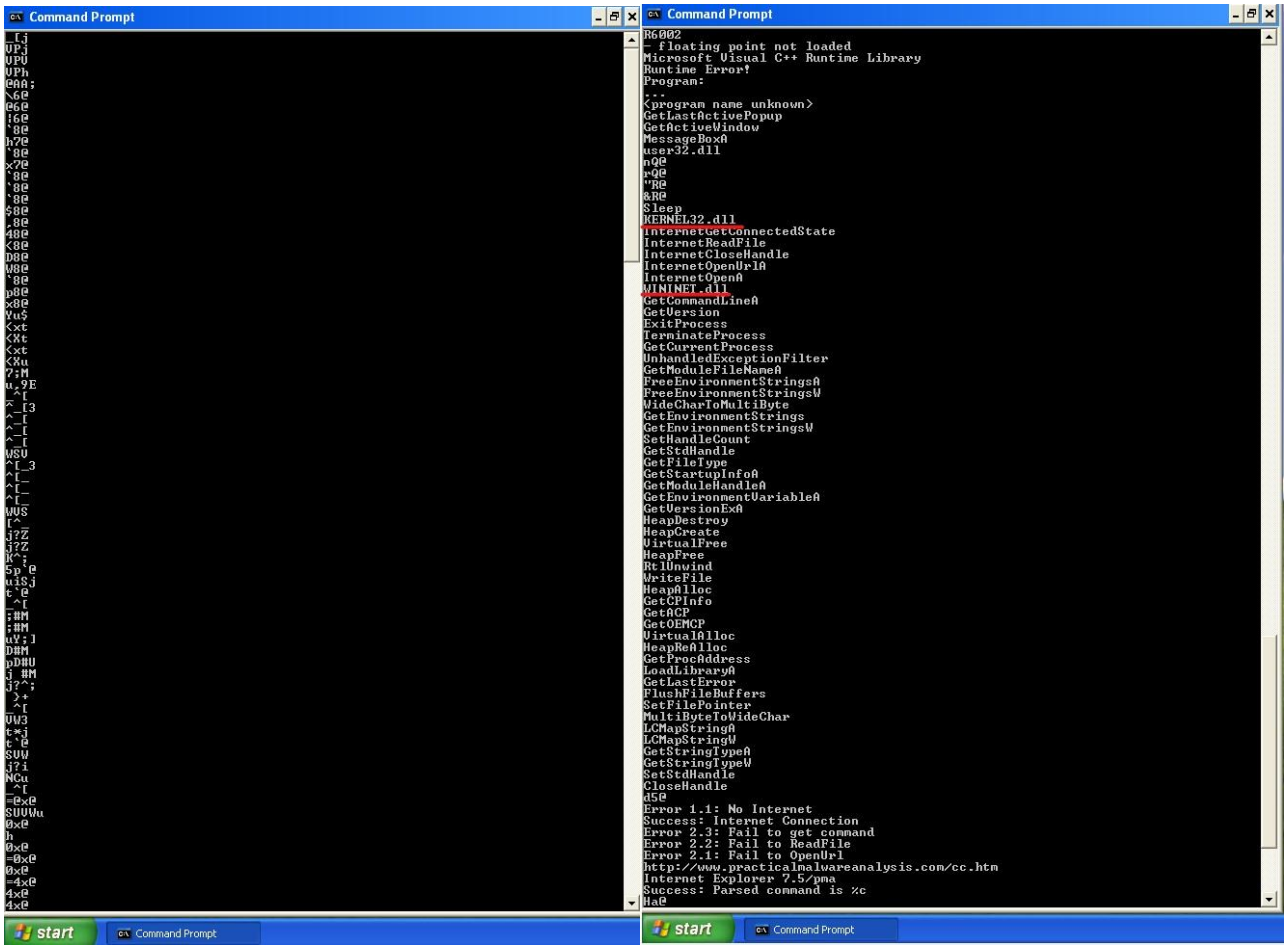


Trovato l'hash possiamo andarlo a cercare esternamente sempre su virustotal. La schermata che ci restituisce il sito dalla ricerca dell'hash è la seguente:



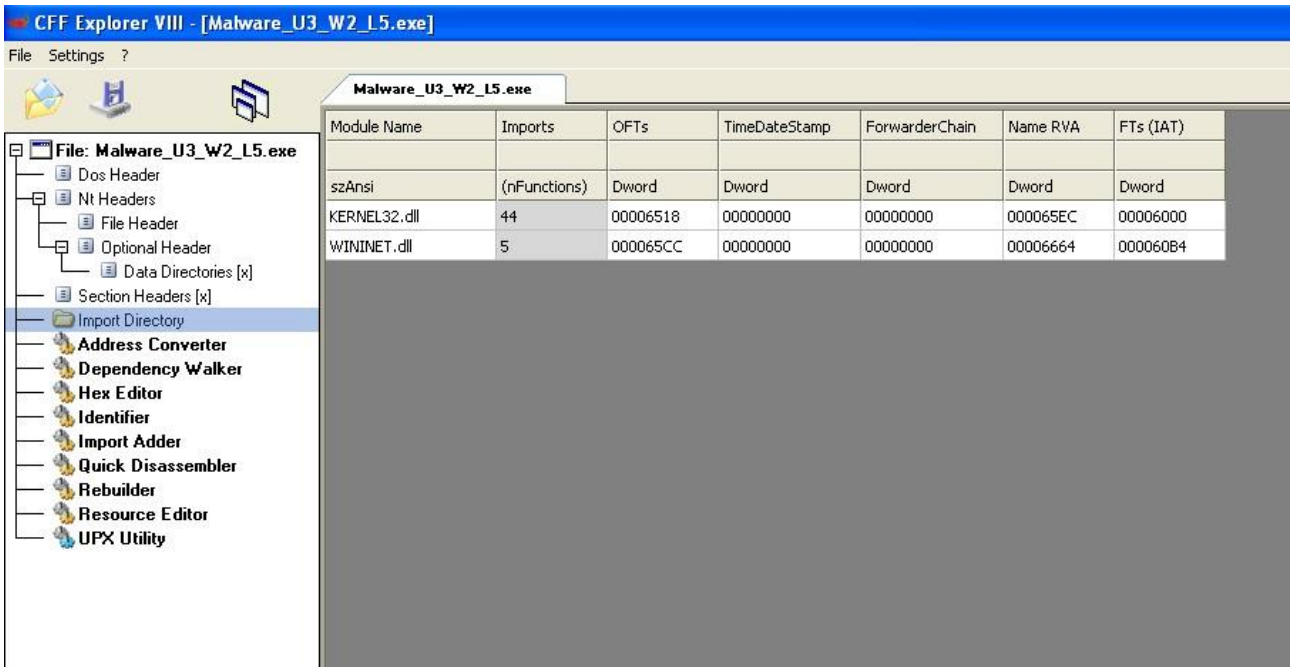
Si tratta sicuramente di un file malevolo, a quanto pare di tipo trojan. Proviamo a cercare informazioni aggiuntive procedendo nell'analisi statica basica, usando questa volta l'utility **strings** di sysinternals per analizzare tutte le *stringhe* di codice contenute nell'eseguibile.

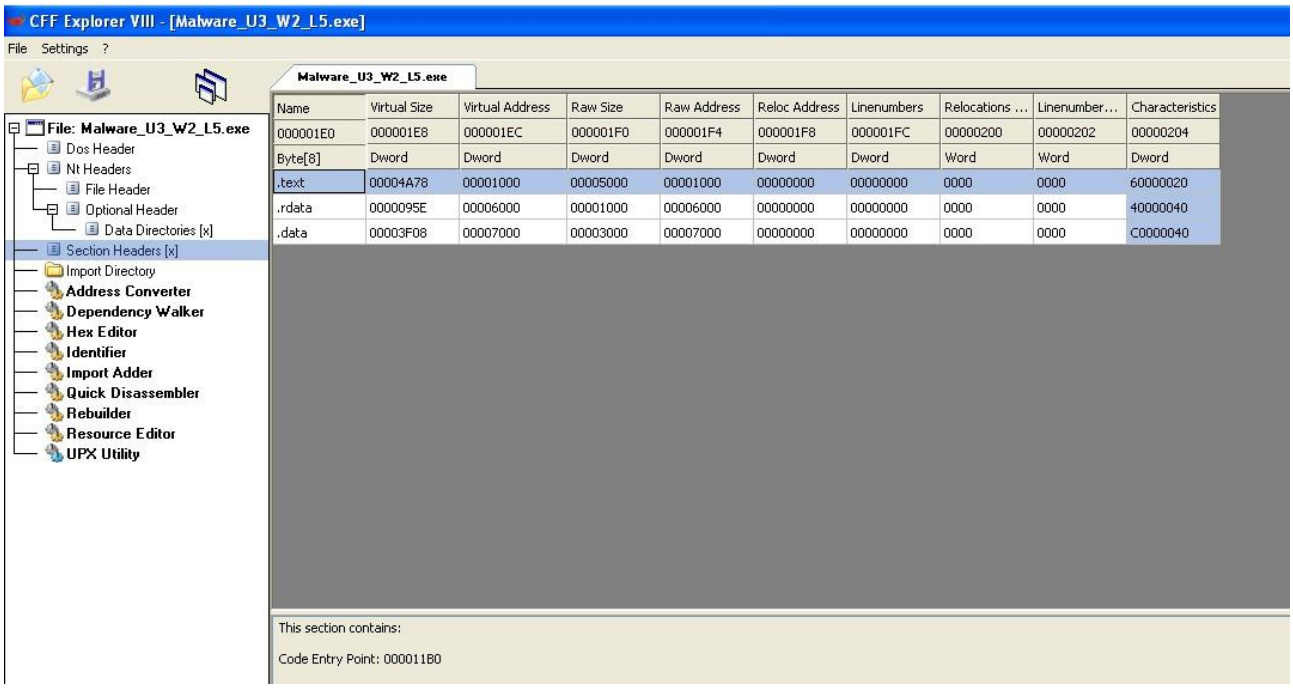
Sempre dal prompt dei comandi, ci spostiamo nella cartella dell'utility e scriviamo il comando <strings "file eseguibile".



Riceviamo una lista di stringhe, e tra molte inutili o incomprensibili (a sinistra) ritroviamo invece delle stringhe che possono confermarci la natura malevola del file. Nell'immagine a destra evidenziate ci sono le librerie richiamate dell'eseguibile, rispettivamente kernel32 e wininet e successivamente infatti sembra che il malware contenga tutte stringhe adibite all'implementazioni di protocolli di rete, probabilmente per creare una backdoor.

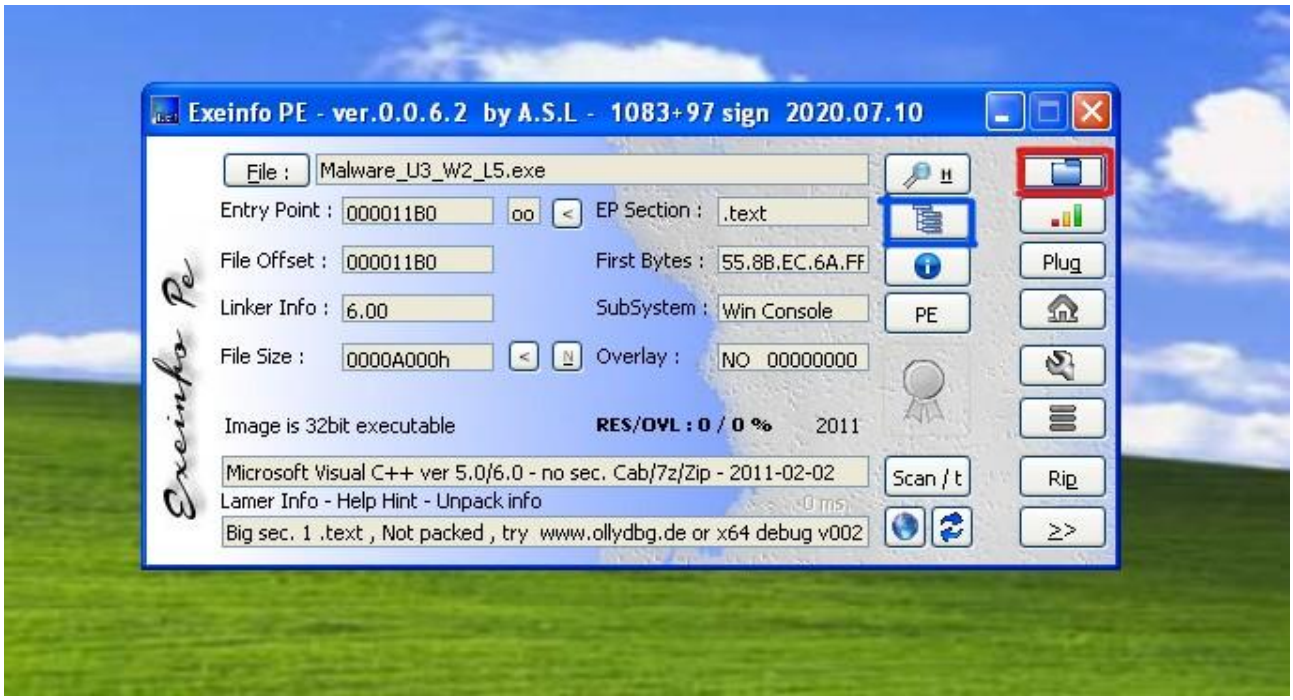
Il prossimo tool che andiamo ad utilizzare è **CFF explorer** per controllare dall'header del formato PE (portable executable) le funzioni importate ed esportate dal malware. Dalla finestra principale del programma apriamo il nostro malware ed analizziamo le varie finestre:



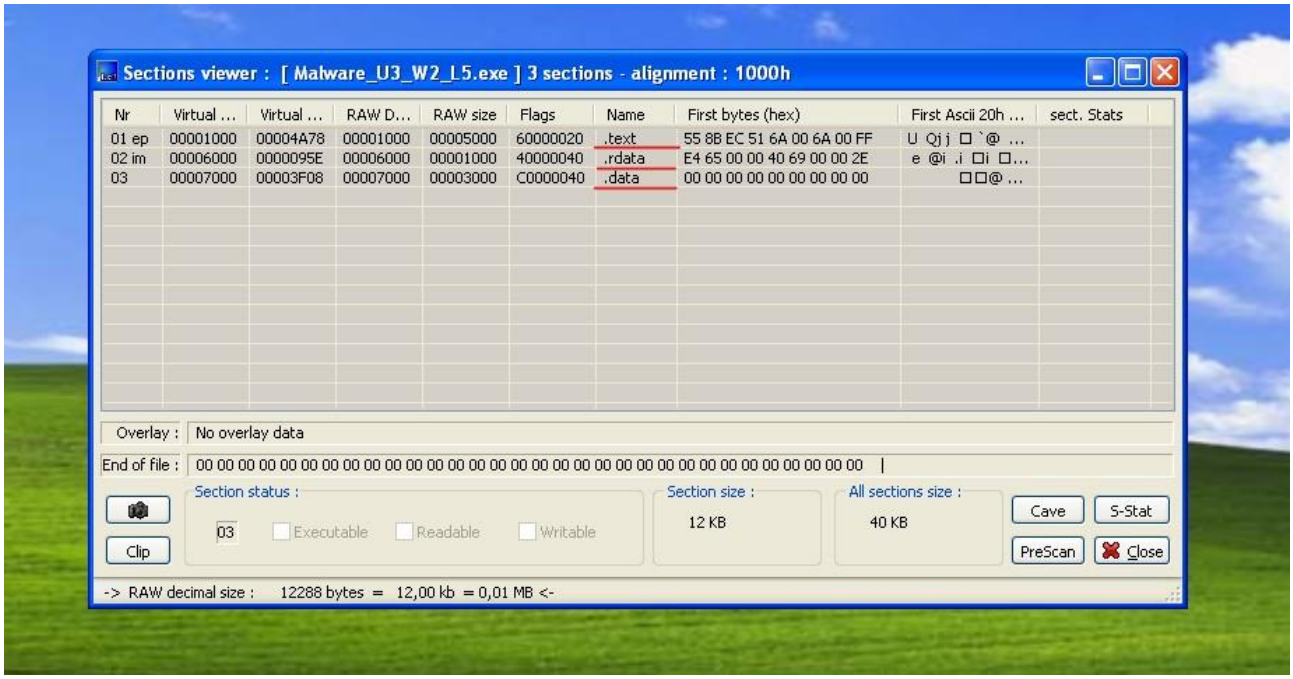


Ritroviamo anche qui le librerie importate *kernel* e *wininet*, nonché le *sezioni* di cui è composto il file, ovvero "*.text*", "*.rdata*" e "*.data*". Andiamo ad utilizzare un ultimo tool similare a quello appena visto, per

vedere se possiamo ricavare qualche informazione aggiuntiva: **ExeinfoPE**. Una volta aperto, andiamo a caricare il file dall'opzione in alto a destra in rosso e successivamente andiamo a visualizzare le diverse sezioni dal pulsante in blu.



Il risultato è il seguente:



Quelle evidenziate in rosso sono effettivamente le *sezioni* del file viste prima, con aggiunta delle virtual e raw size, nonché altre informazioni potenzialmente utili. Ora abbiamo una panoramica completa dal punto di vista dell'analisi statica che ci permette di stilare delle conclusioni.

1. LIBRERIE IMPORTATE

Abbiamo visto che il malware sembra importare due **LIBRERIE** importanti:

- *KERNEL32.DLL*: una delle librerie più comuni di windows, che contiene tutte le funzioni principali per le interazioni con l'OS come la creazione e modifica dei file, o la gestione della memoria
- WININET.DLL: una libreria che contiene le funzioni per l'implementazione dei protocolli di rete (http, FTP…)

Come si evince anche dal risultato dell'analisi delle stringhe dell'eseguibile nonché dalla scansione di virustotal che ce lo riporta come trojan, sembrerebbe legittimo pensare che il malware vada a importare dinamicamente queste due librerie probabilmente per stabilire una connessione non autorizzata a qualche server remoto per l'apertura di una backdoor.

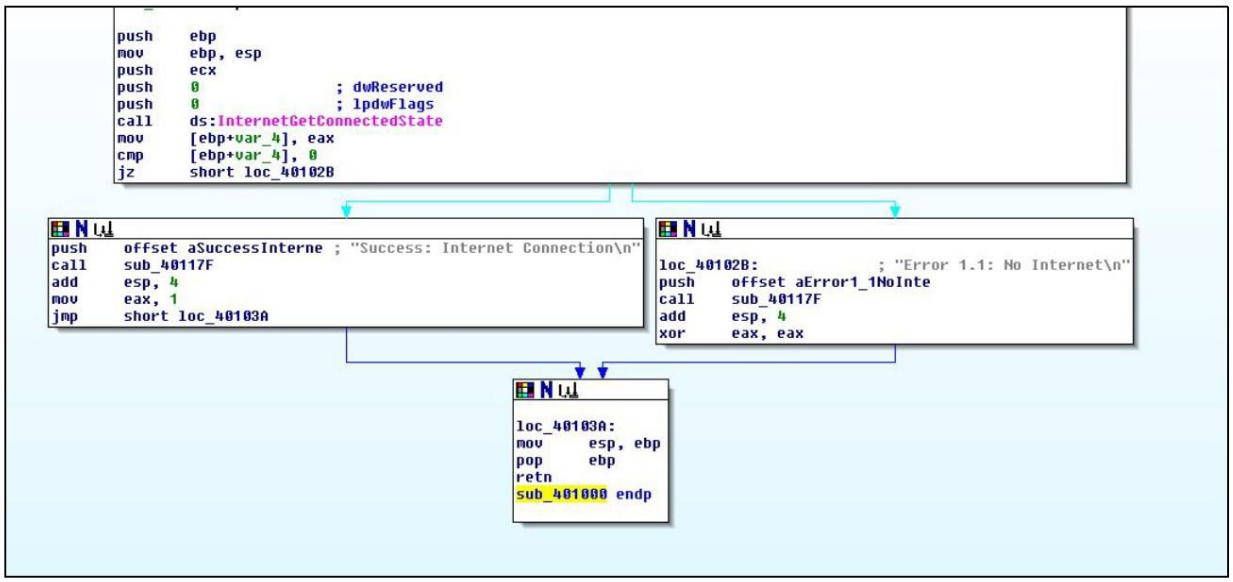
2. SEZIONI DEL FILE MALWARE

Dall'analisi di CFFexplorer prima e ExeinfoPE poi, abbiamo constatato che l'eseguibile si compone di tre **SEZIONI** principali:

- .text: che contiene le istruzioni da far eseguire alla CPU all'avvio del file, ovvero contiene tutte le righe di codice
- .rdata: che contiene le informazioni sulle librerie e le funzioni importate ed esportate (kernel32 e wininet)
- .data: che contiene le variabili globali del codice, che devono essere disponibili per l'accesso da qualsiasi funzione interna

Il file quindi contiene quasi tutte le comuni sezioni rinvenibili nell'header di un eseguibile PE.

ASSEMBLY



3. COSTRUTTI NOTI

In riferimento alla figura in alto possiamo identificare:

```
push    ebp
mov     ebp, esp
push    ecx
```

Questa tripletta iniziale non è che la **creazione dello stack**. Ritroviamo infatti gli stack pointer EBP (Extendend Base Pointer), ESP (Extended Stack Pointer) e ECX che sono tutte voci di registro general purpose per inizializzare una funzione, ovvero ogni chiamata di funzione crea uno stack e queste sono le righe iniziali riferite per l'appunto alla creazione del suddetto stack. In particolare “push ebp” inserisce un il pointer alla base dello stack, “mov ebp, esp” inserisce un pointer in cima allo stack appena creato e “push ecx” inserisce un registro ecx nello stack.

```
push    0               ; dwReserved
push    0               ; lpdwFlags
```

Questi due “push 0” creano uno spazio vuoto nello stack riservato ai valori “dwreserved” e “lpdwflags” di cui è ignoto l’utilizzo. Il secondo probabilmente è un puntatore che riceve info su un set di flag.

```
call    ds:InternetGetConnectedState
```

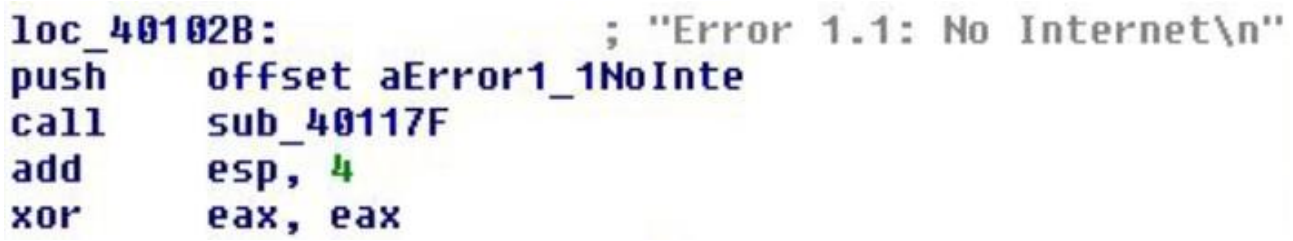
Questa è una chiamata di funzione, e in particolare una chiamata alla libreria wininet.dll per verificare se è disponibile una connessione ad internet.

Le righe soprastanti includono un “mov” che copia il valore del registro eax dentro un altro valore [ebp+var_4], e un “cmp” insieme a un “jz” identificabili come un **costrutto if**. In particolare “cmp” compara il valore sorgente 0 al valore destinazione contenuto nel registro [ebp+var_4] e in base al risultato “jz” salta alla locazione di memoria specificata. In altre parole cmp effettua un’operazione aritmetica simile alla sottrazione (sub) senza modificare gli operandi, ma modificando le flag ZF (Zero Flag) e CF (Carry Flag).

Per chiarire: se nel <cmp destinazione, sorgente> la sorgente è uguale alla destinazione si avrà una sottrazione tra numeri uguali e il risultato sarà 0, dunque ZF sarà uguale a 1.

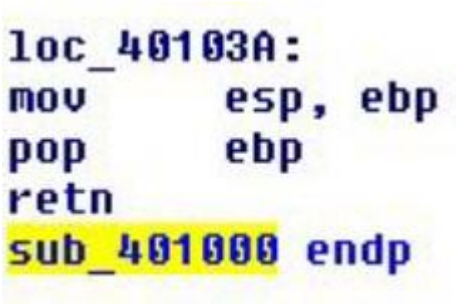
In questo caso l’istruzione “jz” controlla se ZF è vera quindi se ZF=1, e salta alla locazione di memoria indicata se questa condizione è verificata.

Questo costrutto if segna l’inizio di un **ciclo**.



Se ZF=1, il salto di jz ci porta qui dove notiamo un’istruzione push sullo stack di un errore (print di errore di connessione), una chiamata “call” a una funzione sconosciuta e un’operazione “add” di somma di un valore 4 al registro esp, per poi chiudere con un <xor eax, eax> che inizializza a 0 il registro eax, quindi pulisce il valore per ricominciare il ciclo.

Se ZF=0, il codice continua qui con un “push” di un messaggio print di successo di connessione, poi anche qui viene effettuata un’operazione “add” del valore 4 al registro esp, l’istruzione “mov” del valore 1 al registro eax e infine il “jmp” a un’altra locazione di memoria.



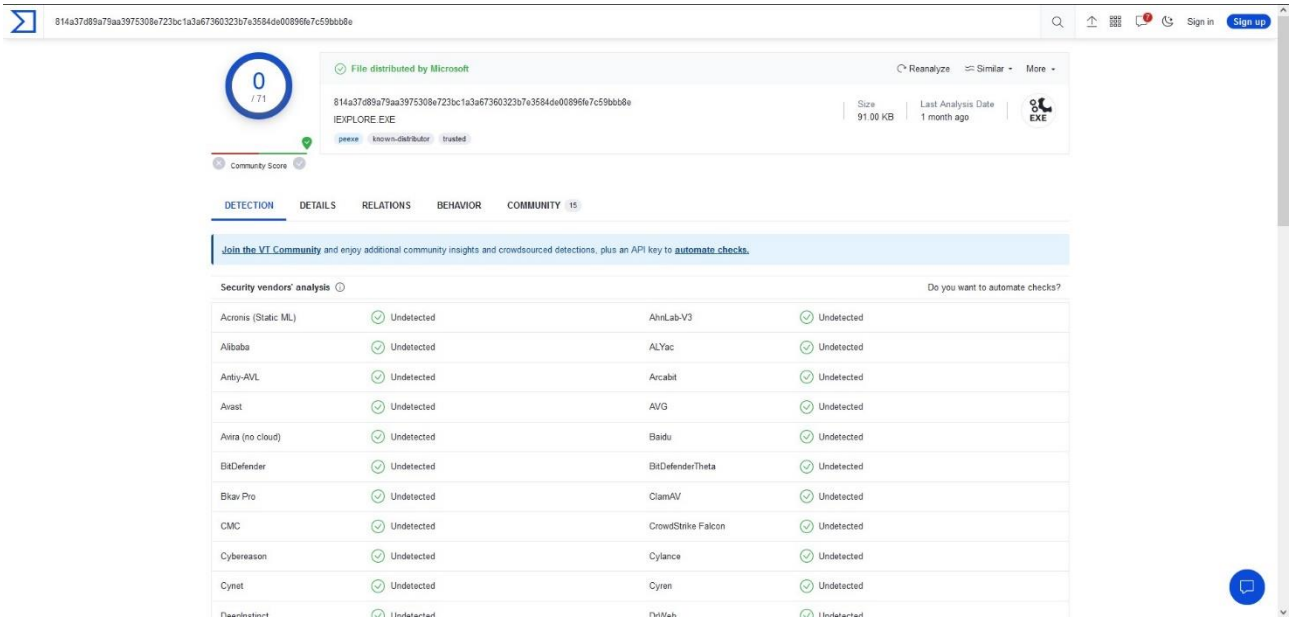
L'ultimo jmp conduce a questa locazione di memoria che chiude il ciclo: il "mov" va a spostare il l'ebp sullo stack e infine il "pop" lo elimina. "retn" è un ritorno alla funzione chiamante e "endp" termina la funzione chiamata.

4. IPOTESI COMPORTAMENTO

In conclusione questo codice assembly cerca di ottenere una verifica di connessione avvenuta. È in sostanza una funzione della libreria wininet.dll chiamata da un'altra funzione principale (chiamante) che va a fare un controllo su determinati valori inseriti in delle variabili per capire se esiste ed è possibile stabilire una connessione ad internet, ovvero capire se la macchina è online o meno. Nel caso di alcuni codici malevoli, questa è una prerogativa per riuscire ad ottenere connessioni non autorizzate a server remoti o scaricare file indesiderati o pericolosi.

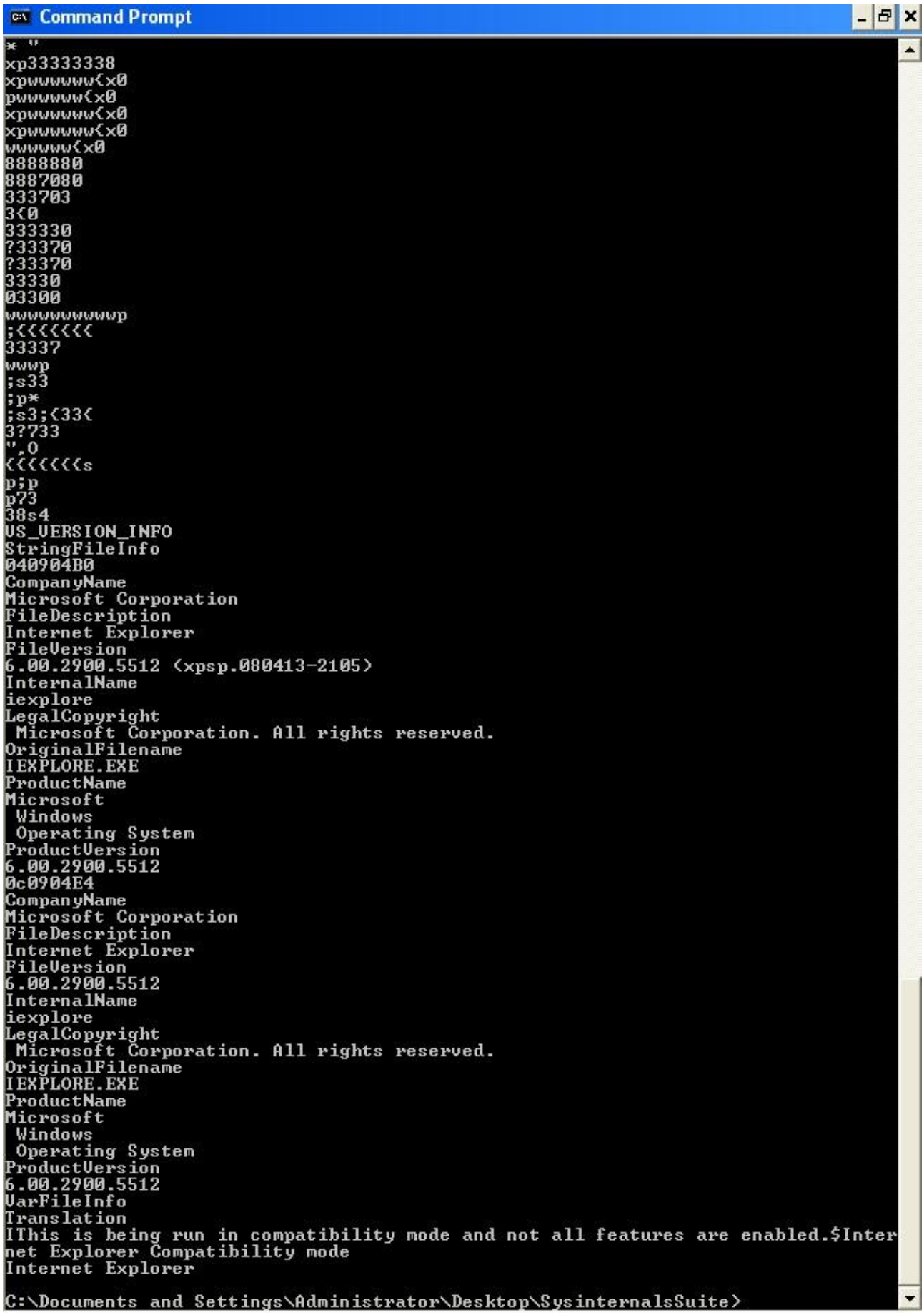
BONUS

Per tranquillizzare il nostro collega, facciamo delle verifiche sul file IEXPLORER.exe, l'eseguibile del browser predefinito di windows. Non potendoci affidare a virustotal per caricare il file in mancanza di connessione sulla macchina, possiamo però estrarre l'hash con md5deep e andarlo a cercare esternamente:

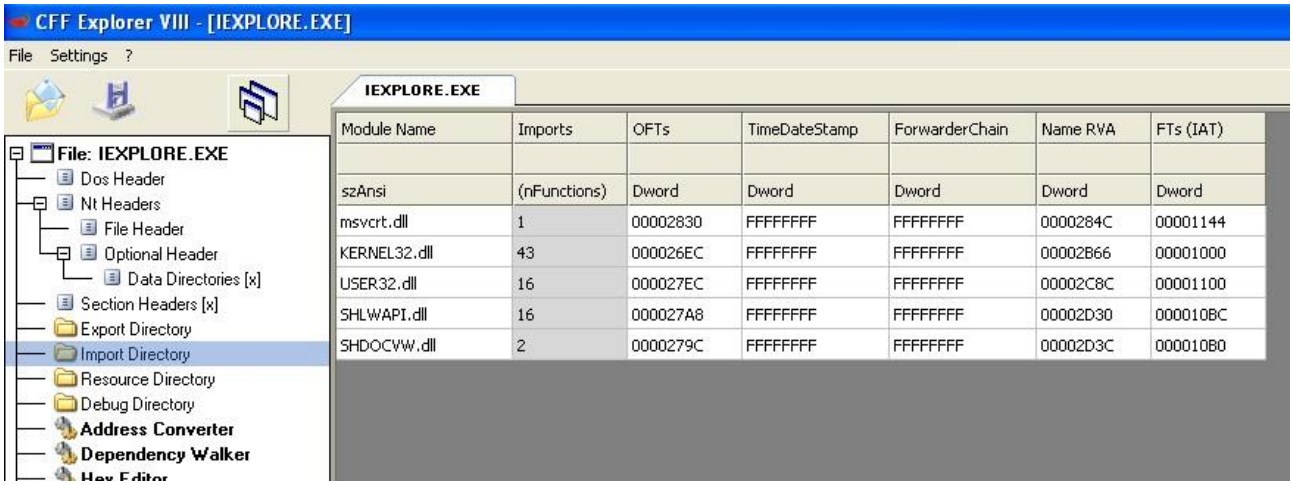


Abbiamo già una conferma di quello che sapevamo. L'hash è sicuro, il file è distribuito da microsoft, ma andiamo a fare ulteriori analisi.

L'analisi delle stringhe non riporta nulla di anomalo, se non altre conferme riguardo il vendor microsoft. Ma queste stringhe potrebbero essere state inserite manualmente, dunque a scanso di equivoci proseguiamo.


Command Prompt
* "
xp33333338
xpwwwwww{x0
pwwwwww{x0
xpwwwwww{x0
xpwwwwww{x0
wwwwww{x0
8888880
8887080
333703
3{0
333330
?33370
?33370
33330
03300
wwwwwwwwwp
;{{{{{{{
33337
wwwp
;s33
;p*
;s3;{33{
3?733
",O
{{{{{{{s
p;p
p73
38s4
VS_VERSION_INFO
StringFileInfo
040904B0
CompanyName
Microsoft Corporation
FileDescription
Internet Explorer
FileVersion
6.00.2900.5512 (xpsp.080413-2105)
InternalName
iexplore
LegalCopyright
Microsoft Corporation. All rights reserved.
OriginalFilename
IEXPLORE.EXE
ProductName
Microsoft
Windows
Operating System
ProductVersion
6.00.2900.5512
0c0904E4
CompanyName
Microsoft Corporation
FileDescription
Internet Explorer
FileVersion
6.00.2900.5512
InternalName
iexplore
LegalCopyright
Microsoft Corporation. All rights reserved.
OriginalFilename
IEXPLORE.EXE
ProductName
Microsoft
Windows
Operating System
ProductVersion
6.00.2900.5512
VarFileInfo
Translation
IThis is being run in compatibility mode and not all features are enabled.$Inter
net Explorer Compatibility mode
Internet Explorer
C:\Documents and Settings\Administrator\Desktop\SysinternalsSuite>

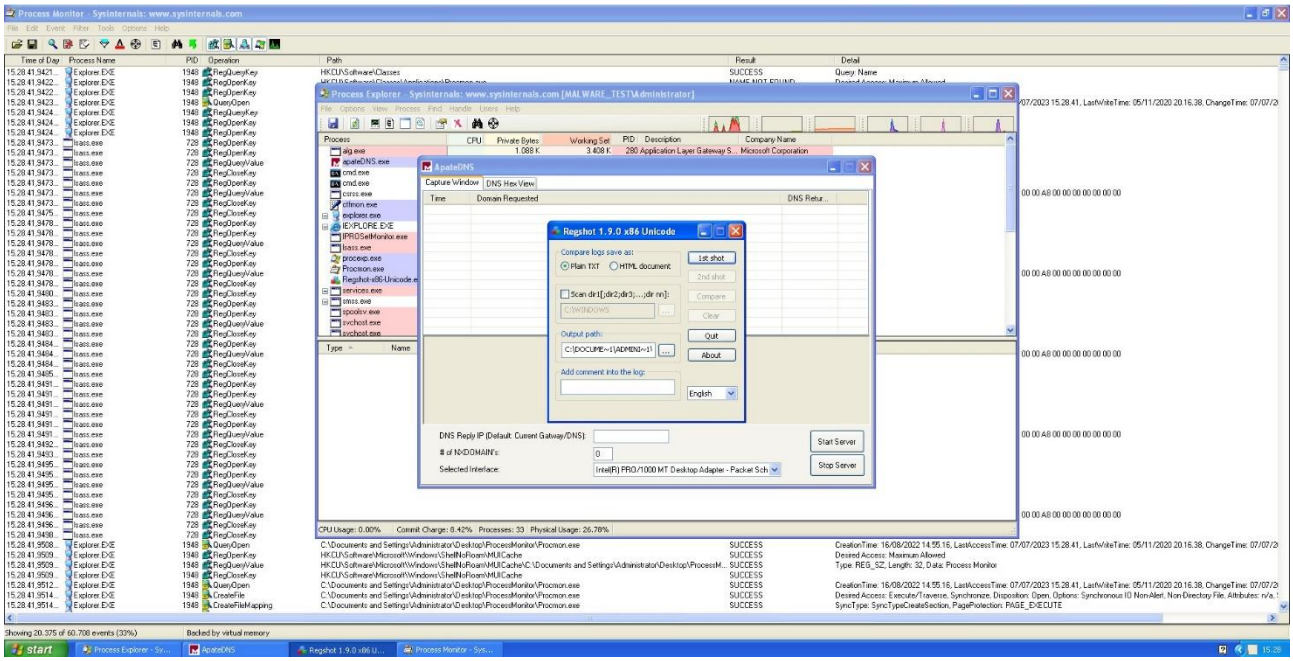
CFFexplorer è ugualmente poco significativo, ci mostra solo le sezioni e le funzioni importate/esportate, peraltro normali per il tipo di eseguibile. In particolare shdocvw è una libreria per operazioni di networking, così come shlwapi che contiene funzioni per i percorsi dell'URL.



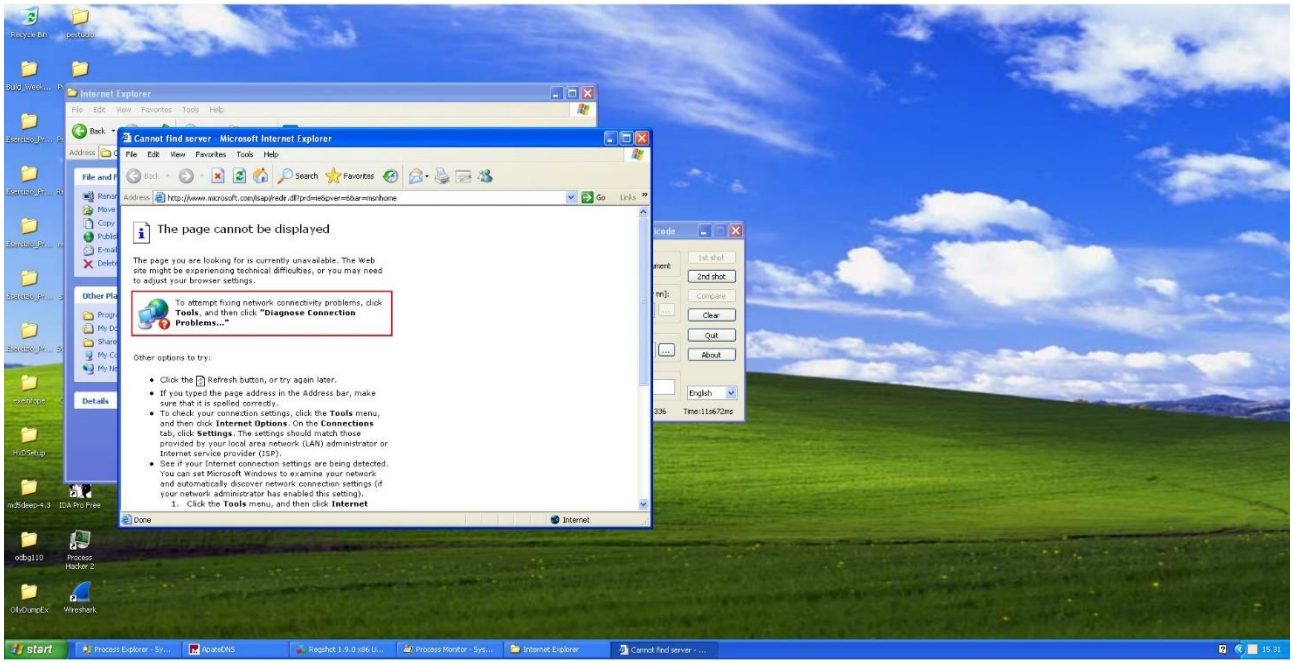
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
msvcrt.dll	1	00002830	FFFFFFFF	FFFFFFFF	0000284C	00001144
KERNEL32.dll	43	000026EC	FFFFFFFF	FFFFFFFF	00002B66	00001000
USER32.dll	16	000027EC	FFFFFFFF	FFFFFFFF	00002C8C	00001100
SHLWAPI.dll	16	000027A8	FFFFFFFF	FFFFFFFF	00002D30	000010BC
SHDOCVW.dll	2	0000279C	FFFFFFFF	FFFFFFFF	00002D3C	000010B0

Non andremo ad usare exeinfope per ovvi motivi. Per toglierci definitivamente ogni dubbio, passiamo a un livello di studio più approfondito con l'analisi dinamica basica.

Possiamo provare un approccio drastico aprendo tutti i tool a nostra disposizione, quindi affidandoci a process explorer, apatedns, regshot e procmon (ed eventualmente wireshark).



Eseguiamo tutto, avviamo l'eseguibile, aspettiamo e vediamo cosa succede:



Explorer come ci aspettavamo non può raggiungere alcuna pagina internet, essendo disabilitata la scheda di rete. Analizziamo dunque i risultati dei tool: