Floriana Feminò

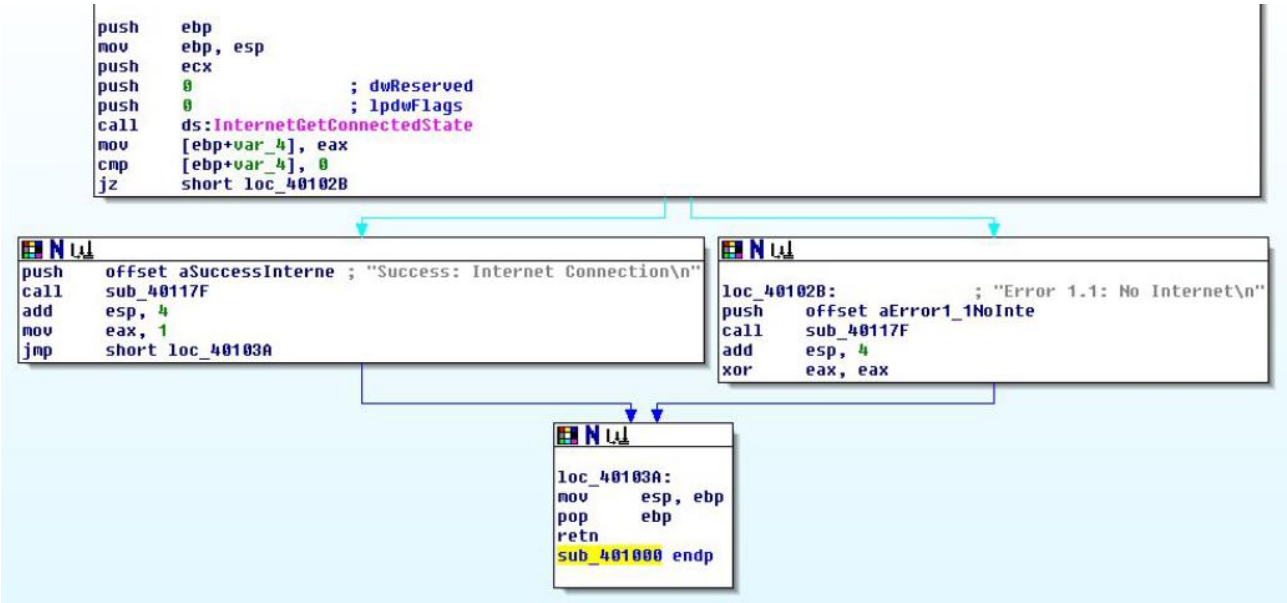
13/01/2023

MALWARE ANALYSIS

Analisi del malware *Malware_U3_W2_L5.exe*

Tasks:

1. **Identificazione delle librerie importate dal file eseguibile**
2. **Identificazione delle sezioni di cui si compone il file eseguibile**
3. **Identificazione dei costrutti noti a partire dalle istruzioni implementate in linguaggio Assembly**
4. **Ipotesi sul comportamento delle funzionalità implementate**
5. **Illustrazione del significato delle singole righe di codice Assembly**

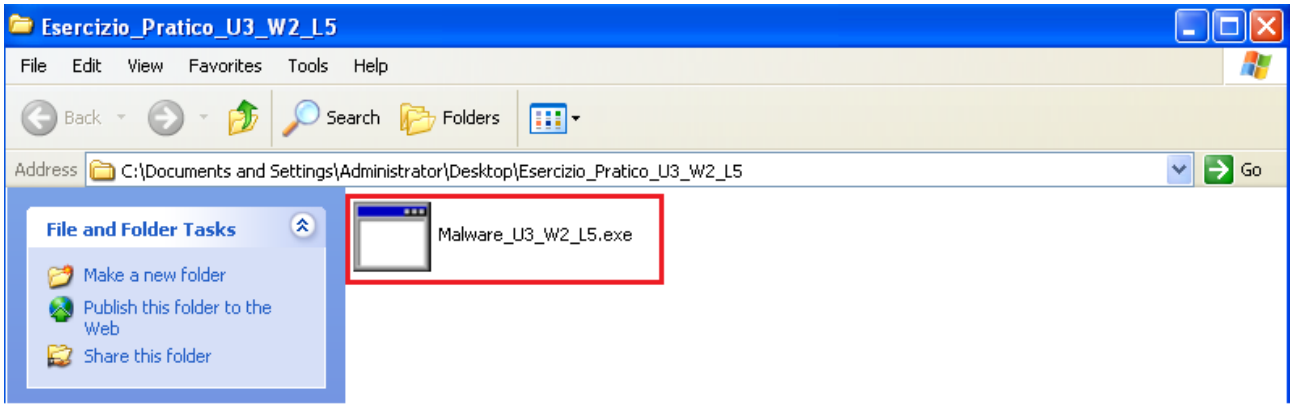


---

1. Identificazione delle librerie importate dal file eseguibile

Le attività odierne sono incentrate sull'analisi del file eseguibile di test **Malware_U3_W2_L5.exe**; lo rintracciamo all'interno del path

C:\Documents and Settings\Administrator\Desktop\Esercizio_Pratico_U3_W2_L5 della VM "Malware Analysis_Final" con sistema operativo Windows XP SP3.



Analisi Statica Basica

Le operazioni oggetto di focus odierno rientrano nel settore della **Malware Analysis** (= analisi dei malware).

L'**analisi dei malware** è l'insieme di competenze e tecniche che permettono ad un'analista della sicurezza informatica di **indagare accuratamente un malware per studiarne il comportamento**, al fine di rimuoverlo correttamente dal sistema; tali competenze sono fondamentali per i membri tecnici del CSIRT durante la risposta agli incidenti di sicurezza.

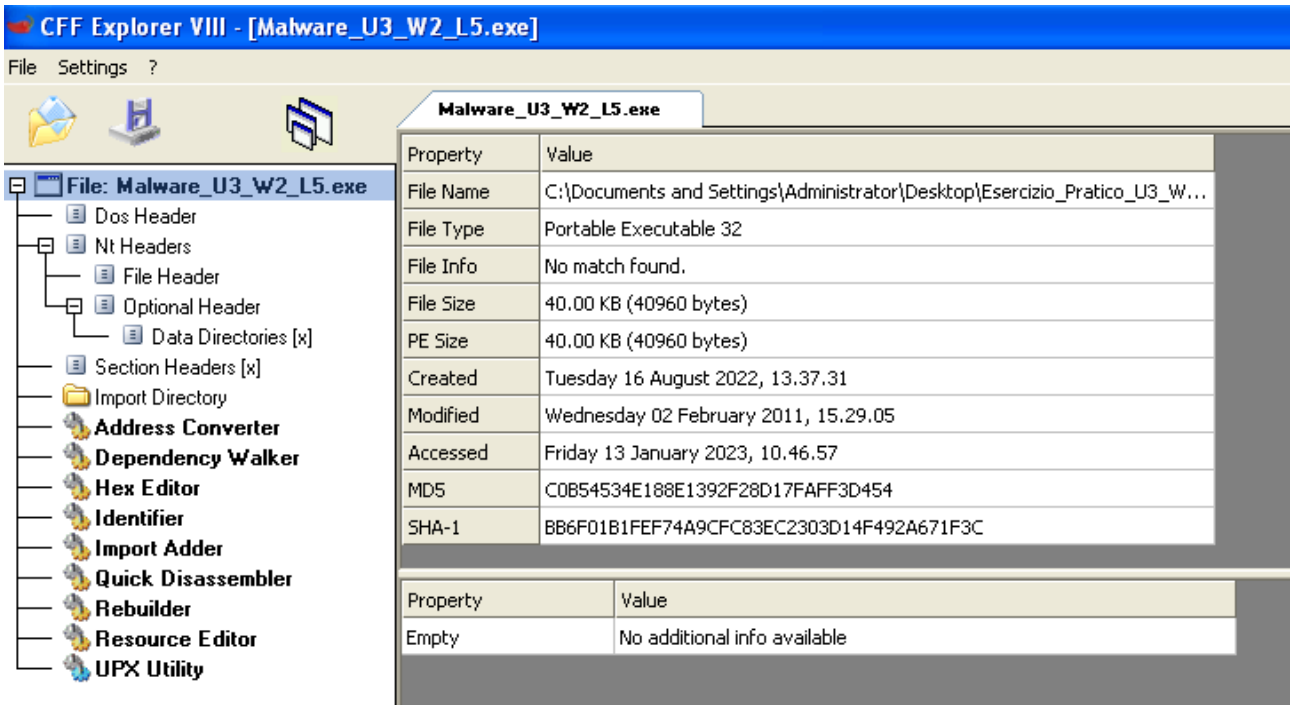
Nello specifico, in questa prima sezione del test svolgeremo alcune attività di **Analisi Statica *Basica***: questo tipo di analisi consiste nell'esaminare un file **eseguibile** (è il caso dei file con estensione **.exe** in ambiente Windows) senza tener conto delle istruzioni che lo compongono. Lo scopo di questa analisi è confermare la natura malevola di un file e fornire informazioni generiche circa le sue funzionalità. È la più semplice da mettere in pratica, tuttavia può rivelarsi inefficiente in presenza di malware complessi.

Per avviare l'analisi utilizzeremo il tool **CFF Explorer**, che si occupa di esaminare l'**header del formato PE** (= ***P**ortable **E**xecutable*). Infatti, il sistema operativo Windows utilizza questo formato per la maggior parte dei file eseguibili; al suo interno sono contenute le informazioni necessarie al sistema operativo per capire come gestire il codice del file. Nello specifico, all'interno dell'header del formato PE troviamo

- Un elenco delle **librerie** importate e delle funzioni richieste da un eseguibile
- Eventuali funzioni esportate (= messe a disposizione di altri programmi) dall'eseguibile
- **Sezioni** di cui si compone il software

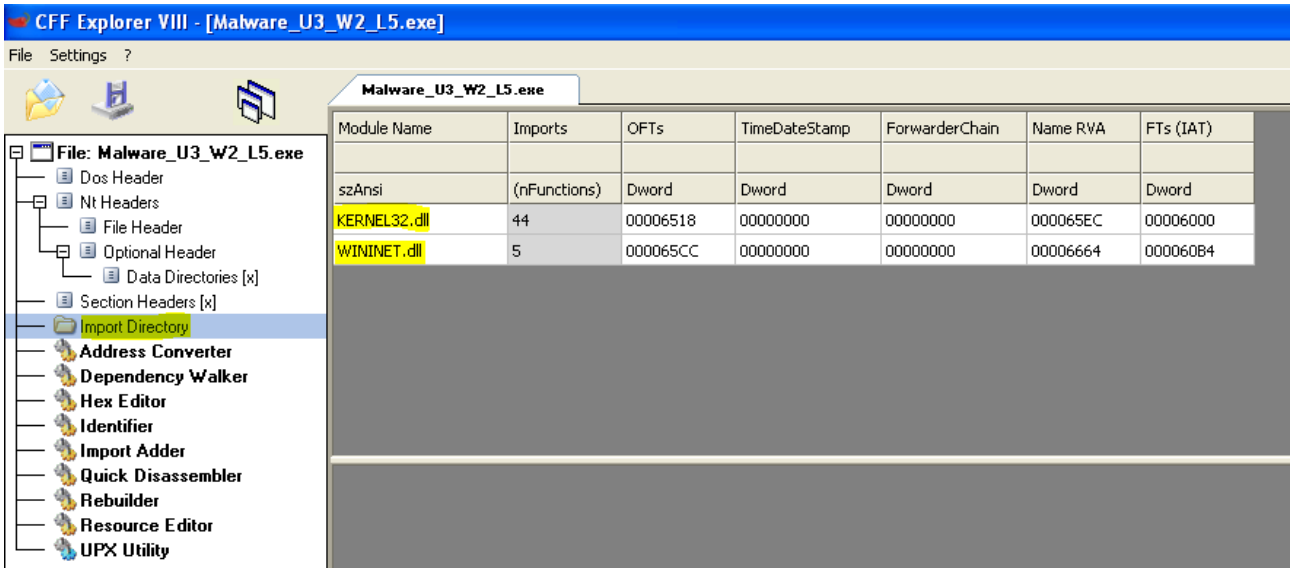
Una libreria è un insieme di funzioni. Per poter funzionare, un Malware richiama funzioni contenute in una o più librerie di Windows: è facile intuire, quindi, che controllare quali sono le librerie e le funzioni importate è fondamentale per identificare lo scopo del malware.

Avviamo il tool e apriamo al suo interno l'eseguibile di nostro interesse **Malware_U3_W2_L5.exe**



Il tool restituisce, nella schermata iniziale, alcune informazioni di riepilogo tra cui le dimensioni, la data di creazione ed il suo hash nei formati MD5 e SHA-1, che si rivelano utili ai fini di una successiva ricerca su un database online come quello di VirusTotal, per ottenere più informazioni circa il comportamento del malware in oggetto.

Spostandoci nella sezione "**Import Directory**", possiamo identificare le librerie importate dal malware:



KERNEL32.dll – libreria piuttosto comune che contiene le funzioni principali per interagire con il sistema operativo, es. manipolazione del file, gestione della memoria

WININET.dll – libreria che contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP (network time protocol)

Inoltre, selezionando individualmente una libreria, è possibile rintracciare al suo interno le funzioni richieste dal malware. L'elenco di **funzioni richieste dalla libreria KERNEL32.dll** è il seguente:

OFTs	FTs (IAT)	Hint	Name	OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi	Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep	000067A4	000067A4	0150	GetStartupInfoA
00006940	00006940	027C	SetStdHandle	000067B6	000067B6	0126	GetModuleHandleA
0000692E	0000692E	0156	GetStringTypeW	000067CA	000067CA	0109	GetEnvironmentVariableA
0000691C	0000691C	0153	GetStringTypeA	000067E4	000067E4	0175	GetVersionExA
0000690C	0000690C	01C0	LCMapStringW	000067F4	000067F4	019D	HeapDestroy
000068FC	000068FC	01BF	LCMapStringA	00006802	00006802	019B	HeapCreate
000068E6	000068E6	01E4	MultiByteToWideChar	00006810	00006810	02BF	VirtualFree
00006670	00006670	00CA	GetCommandLineA	0000681E	0000681E	019F	HeapFree
00006682	00006682	0174	GetVersion	0000682A	0000682A	022F	RtlUnwind
00006690	00006690	007D	ExitProcess	00006836	00006836	02DF	WriteFile
0000669E	0000669E	029E	TerminateProcess	00006842	00006842	0199	HeapAlloc
000066B2	000066B2	00F7	GetCurrentProcess	0000684E	0000684E	00BF	GetCPInfo
000066C6	000066C6	02AD	UnhandledExceptionFilter	0000685A	0000685A	00B9	GetACP
000066E2	000066E2	0124	GetModuleFileNameA	00006864	00006864	0131	GetOEMCP
000066F8	000066F8	00B2	FreeEnvironmentStringsA	00006870	00006870	02BB	VirtualAlloc
00006712	00006712	00B3	FreeEnvironmentStringsW	00006880	00006880	01A2	HeapReAlloc
0000672C	0000672C	02D2	WideCharToMultiByte	0000688E	0000688E	013E	GetProcAddress
00006742	00006742	0106	GetEnvironmentStrings	000068A0	000068A0	01C2	LoadLibraryA
0000675A	0000675A	0108	GetEnvironmentStringsW	000068B0	000068B0	011A	GetLastError
00006774	00006774	026D	SetHandleCount	000068C0	000068C0	00AA	FlushFileBuffers
00006786	00006786	0152	GetStdHandle	000068D4	000068D4	026A	SetFilePointer
00006796	00006796	0115	GetFileType	00006950	00006950	001B	CloseHandle

Notiamo che tra le funzioni richieste sono presenti **LoadLibraryA** e **GetProcAddress**: ciò implica che la modalità di importazione è **a tempo di esecuzione (runtime)**. L'eseguibile, dunque, richiama la libreria solamente quando ha necessità di utilizzare una determinata funzione, ossia durante l'esecuzione del file. Questo comportamento è ampiamente analizzato dai malware, che chiamano una determinata funzione all'occorrenza **per risultare meno invasivi e rilevabili** possibile.

Ne consegue che non è possibile identificare con precisione tramite la sola analisi statica il dettaglio delle operazioni svolte e la finalità ultima del malware. Per avere più dettagli ed osservare da vicino il comportamento del file, si rende dunque necessaria un'analisi dinamica (= un'analisi del malware effettuata a partire dalla sua esecuzione in ambiente protetto).

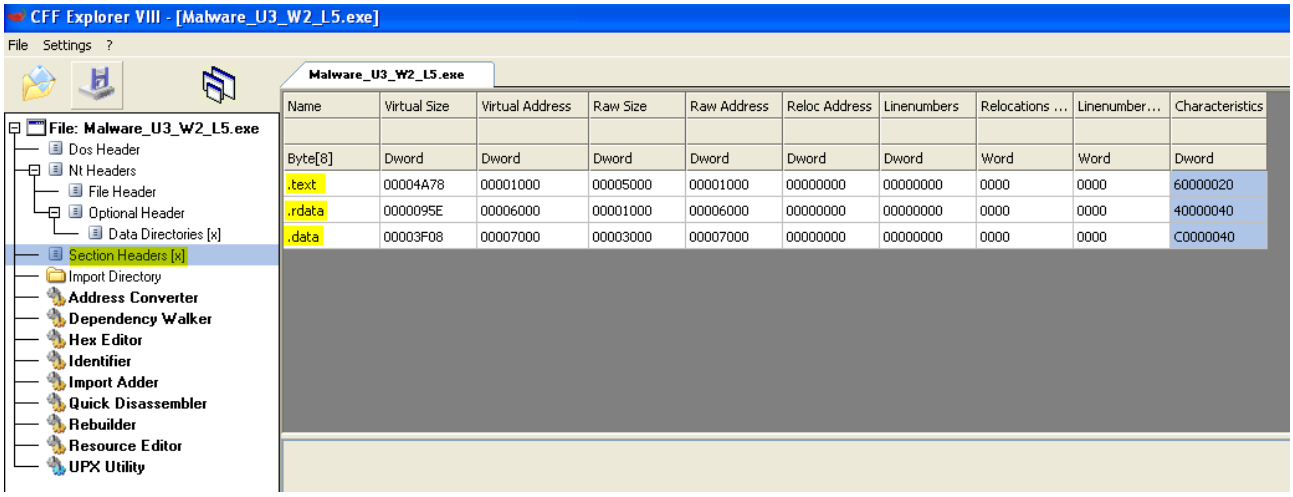
L'elenco di **funzioni richieste dalla libreria WININET.dll** è il seguente:

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

Tra le funzioni richieste, notiamo la presenza di **InternetGetConnectedState**, che ha lo scopo di verificare se una macchina ha accesso ad Internet: da ciò si può ipotizzare che il malware svolga alcune operazioni tramite una connessione Internet.

2. Identificazione delle sezioni di cui si compone il file eseguibile

Spostandoci all'interno della sezione "**Section Headers**", possiamo identificare le seguenti sezioni:



.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

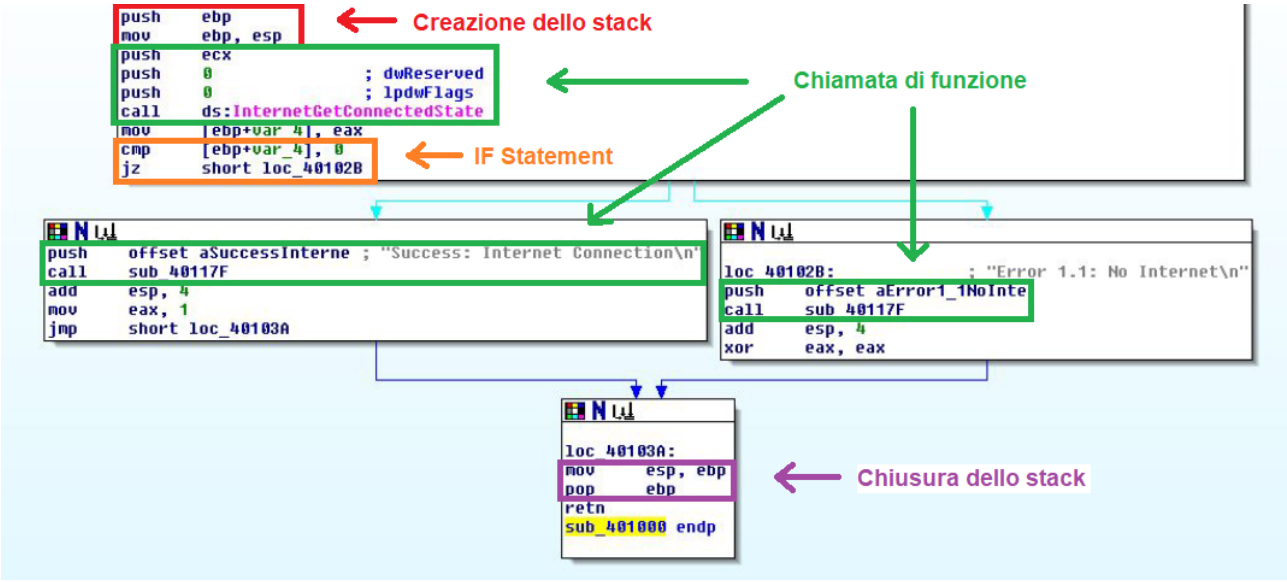
.data – contiene dati/**variabili globali** del programma eseguibile, che devono essere quindi disponibili da qualsiasi parte del programma (una variabile è globale quando non è definita all'interno di una funzione, ma è globalmente dichiarata ed è quindi accessibile da qualsiasi funzione dell'eseguibile)

Nella schermata è possibile rintracciare anche altre informazioni circa le sezioni, come la **Virtual size** (che fa riferimento alla memoria RAM), ossia la dimensione dello spazio allocato per la sezione **durante il processo di caricamento** dell'eseguibile in memoria) e la **Raw size** (dimensione dello spazio occupato dalla sezione quando è sul disco).

Alcuni malware più avanzati oscurano il nome delle sezioni oppure le configurano con un nome nonsense; ciò costituisce un'ulteriore prova di inefficacia prestazionale della sola analisi statica in presenza di malware sofisticati.

3. Identificazione dei costrutti noti a partire dalle istruzioni implementate in linguaggio Assembly

All'interno delle istruzioni in linguaggio Assembly, è possibile identificare i costrutti rappresentati nella seguente immagine:



4. Ipotesi sul comportamento delle funzionalità implementate

Le istruzioni in linguaggio assembly in esame ci mostrano che il programma si occupa dapprima di creare uno stack per le variabili locali: notiamo la presenza dei due puntatori **EBP** (Extended Base Pointer) ed **ESP** (Extended Stack Pointer) che puntano rispettivamente alla base ed alla cima dello stack.

Successivamente, il programma si occupa di verificare, attraverso la chiamata della funzione **InternetGetConnectedState**, se la macchina vittima ha accesso ad Internet: da ciò si può ipotizzare che il malware si serva della connettività ad Internet per eseguire alcune operazioni di cui non abbiamo dettagliata evidenza a meno di un'esecuzione del software.

Effettuata la verifica posta in esame dall'if statement, si può presumere che il programma stamperà a schermo – ad esempio tramite una finestra di dialogo – un feedback sullo stato della connettività della macchina sia in caso che la verifica abbia esito positivo, sia in caso di esito negativo (chiamata di funzione ***call sub_40117F*** presente in entrambi i blocchi di codice relativi alle condizioni if/else).

Se è presente una connessione Internet, il programma andrà avanti a svolgere alcune operazioni tra cui l'aumento del valore dello stack (*add esp, 4*) e successivamente effettuerà un salto non condizionale all'indirizzo di memoria 40103A, in cui viene effettuata la chiusura dello stack (specificamente tramite l'istruzione ***pop ebp***, che andrà a rimuovere il "piatto" dalla pila) ed il ritorno alla funzione chiamante (istruzione ***retn***).

In caso di mancata connessione ad Internet, viene reinizializzato a 0 il valore del registro eax (*xor eax, eax*).

Si può concludere che, affinché il malware possa sfruttare a pieno le sue funzionalità, abbia bisogno di una connessione ad Internet per svolgere alcune operazioni, ad esempio

- Invio di file verso web server remoti di proprietà dell'attaccante
- Connessione verso uno o più domini infetti
- Creazione di una backdoor per creare una comunicazione persistente tra l'indirizzo IP dell'attaccante e la macchina vittima

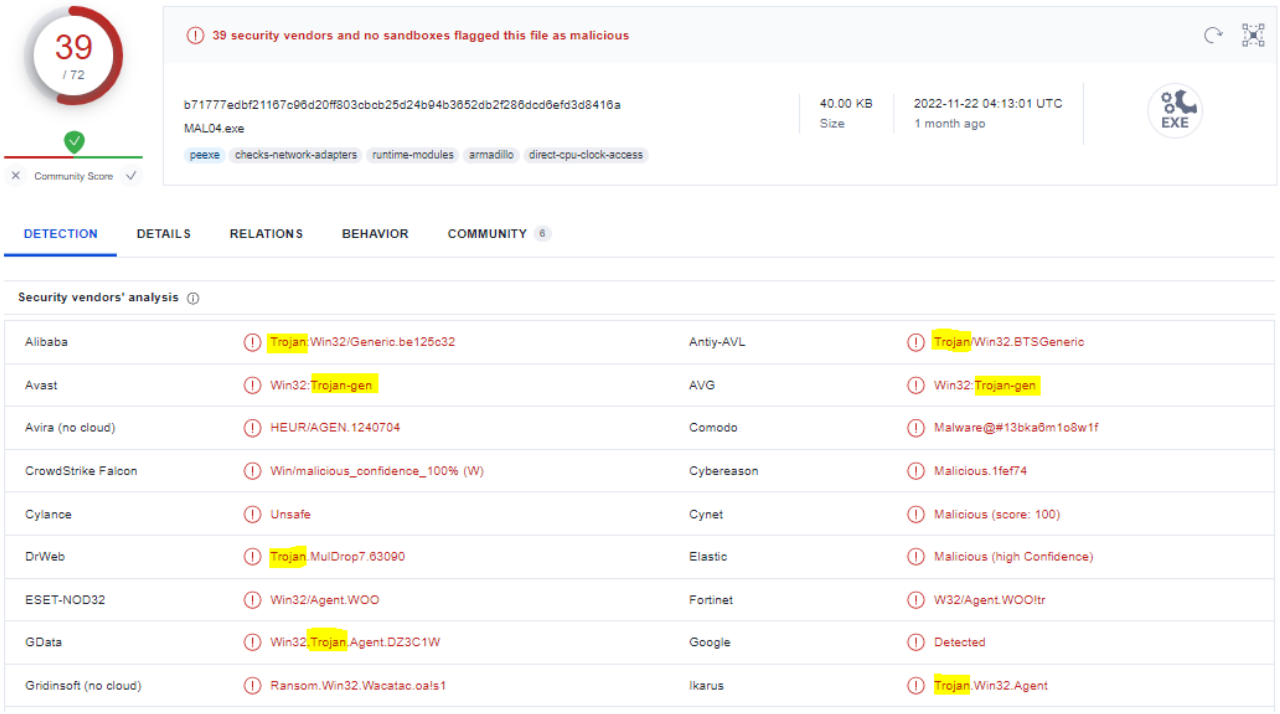
Si può inoltre riassumere il funzionamento del malware, in base alle informazioni in nostro possesso, nel seguente **pseudocodice**:

```
State = InternetGetConnectedState (p1, 0, 0)

if (state ! = 0) printf ("Active connection");
    return 0;
else printf("No connection");
   int a;
   .
   .
   .
   return 0;
```

Ulteriori verifiche

Per ottenere più informazioni circa il malware appena esaminato, possiamo sfruttare gli hash ricavati dall'analisi in CFF Explorer per eseguire una ricerca sul database **VirusTotal**, che ci restituisce i seguenti risultati:



Come si vede, il file ha ottenuto uno score di 39/72 e viene identificato da molti security vendors come software malevolo. Nello specifico, più di un vendor identifica l'eseguibile in esame come malware di tipo Trojan.

Fonte:
https://www.virustotal.com/gui/file/b71777edbf21167c96d20ff803cbcb25d24b94b3652db2f286dcd6efd3d8416a/detection

5. Illustrazione del significato delle singole righe di codice Assembly

Push ebp	Viene "pushato" il registro Extended Base Pointer sulla cima dello stack
Mov ebp, esp	Viene assegnato il valore del registro dell'Extended Stack pointer al registro dell'Extended Base Pointer
Push ecx	Tramite l'istruzione push, viene posto il valore inserito nel registro "ecx" in cima allo stack

Push 0 ; dwReversed	Viene pushato il parametro 0 di una variabile in cima allo stack
Push 0 ; lpdwFlags	Viene pushato il parametro 0 di una variabile in cima allo stack
Call ds: InternetGetConnectedState	Viene chiamata la funzione "InternetGetConnectedState" che verifica lo stato di connettività del sistema locale
Mov [ebp+var_4], eax	Viene copiato il valore contenuto nel registro "eax" nel registro [ebp+var_4]
Cmp [ebp+var_4], 0	Con questa istruzione viene effettuata una sottrazione tra il parametro contenuto nel registro [ebp+4_var] e 0 andando a modificare la Zero Flag e la Carry Flag (CF) del registro
Jz short loc_40102B	Jump Zero: con questa istruzione di jump viene controllata la zero flag ottenuta dalla precedente istruzione "cmp". Se questa risulterà uguale a 1 verrà effettuato uno "short jump" all'indirizzo di memoria 40102B, altrimenti verranno eseguite normalmente le successive righe di codice
Push offset aSuccessInterne ; “Success: Internet Connection\n”	Viene inserita la stringa "aSuccessInterne" in un registro in cima allo stack
Call sub_40117F	Viene chiamata la funzione all’indirizzo di memoria 40117F
Add esp, 4	Viene sommato il valore 4 a quello contenuto nel registro ESP
Mov eax, 1	Viene sostituito il valore contenuto all'interno del registro "eax" con il valore 1
Jmp short loc_40103A	Viene effettuato un salto all'indirizzo di memoria 40103A
Loc_40102B: ; “Error 1.1: No Internet\n”	Indica la locazione di memoria 40102B
Push offset aError1_1NoInte	Viene “pushato” l’offset aError1_NoInte in cima allo stack
Call sub_40117F	Viene chiamata la funzione all’indirizzo di memoria 40117F
Add esp, 4	Viene sommato il valore 4 a quello contenuto nel registro ESP
Xor eax, eax	Viene usata l’istruzione XOR per inizializzare a 0 il registro EAX
Loc_40103A:	Indica la locazione di memoria 40103A
Mov ebp, esp	Viene copiato il contenuto del registro EBP nel registro ESP
Pop ebp	Viene rimosso il contenuto del registro EBP dalla dallo stack
retn	Organizza il ritorno al programma chiamante al termine di una procedura
Sub_401000 endp	Indica la fine della procedura all’indirizzo di memoria 401000