

S10 L5

ANALISI STATICA E DINAMICA: UN APPROCCIO PRATICO

ELEONORA VIOLA

TRACCIA:

Con riferimento al file **Malware_U3_W2_L5** presente all'interno della cartella «**Esercizio_Pratico_U3_W2_L5**» sul desktop della macchina virtuale dedicata per l'analisi dei malware, rispondere ai seguenti quesiti:

Esercizio Traccia e requisiti

1. Quali librerie vengono importate dal file eseguibile ? Fare anche una descrizione
2. Quali sono le sezioni di cui si compone il file eseguibile del malware? Fare anche una descrizione

Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:

3. Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti)
4. Ipotizzare il comportamento della funzionalità implementata
5. Fare una tabella per spiegare il significato delle singole righe di codice

Analisi sui malware

STATICA

DINAMICA

Basica

Analisi del codice del malware, le sue strutture e le sue funzionalità attraverso disassemblatori, decompilatori, strumenti di firma digitale, analizzatori di file binari. Ispezione del codice sorgente, analisi delle librerie e delle dipendenze, estrazione di stringhe, analisi delle API utilizzate. Identificazione delle funzionalità di base del malware per ottenere una comprensione preliminare delle sue capacità

Avanzata

Analisi del codice del malware, le sue strutture e le sue funzionalità attraverso disassemblatori avanzati (come IDAPro), deobfuscatori, strumenti per l'analisi del flusso di controllo. Analisi approfondita delle tecniche di offuscamento, decodifica delle routine crittografiche, studio dettagliato del comportamento del codice. Identificazione del funzionamento interno del malware e le sue componenti critiche.

Basica

Esecuzione del malware in un ambiente controllato per osservare il suo comportamento in tempo reale attraverso sandboxes, strumenti di monitoraggio dei processi e del traffico di rete. Osservazione delle modifiche al file system, monitoraggio delle chiamate di rete, analisi del comportamento del processo in memoria, registrazione delle attività del malware.

Avanzata

Esecuzione del malware in un ambiente controllato per osservare il suo comportamento in tempo reale attraverso debugger avanzati (come OllyDbg), strumenti per la reverse engineering del traffico di rete, monitoraggio dettagliato delle API di sistema. Debugging passo-passo del malware, analisi del comportamento dinamico sotto varie condizioni, studio delle tecniche di evasione e anche vulnerabilità nel malware stesso

OPERAZIONI PRELIMINARI

Prima di intraprendere qualsiasi operazione di analisi del malware, è essenziale configurare correttamente un ambiente di test.

Alcune best practices da seguire per garantire un ambiente sicuro includono:

1. **Configurazione delle schede di rete:** assicurarsi che la macchina utilizzata per l'analisi non abbia accesso diretto a Internet e, preferibilmente, nemmeno ad altre macchine sulla rete. Durante l'analisi statica, è consigliabile disabilitare completamente le interfacce di rete e abilitare solo un'interfaccia di rete interna per l'analisi dinamica. Questa impostazione consente di monitorare il traffico generato potenzialmente dal malware.
2. **Gestione dei dispositivi USB:** evitare di abilitare o disabilitare il controller USB durante l'analisi. Il malware potrebbe sfruttare i dispositivi USB per propagarsi sulla macchina fisica.
3. **Cartelle condivise:** la stessa precauzione vale per le cartelle condivise tra la macchina di analisi e il laboratorio virtuale. Evitare di condividere cartelle tra l'host e la macchina virtuale, poiché il malware potrebbe sfruttarle per propagarsi al di fuori del laboratorio, causando danni alla macchina e alle altre macchine sulla rete domestica.
4. **Creazione di snapshot:** prima di iniziare qualsiasi analisi, è consigliabile creare uno snapshot della macchina virtuale nel suo stato iniziale. Questo permette di ripristinarla facilmente in caso di necessità o danni causati dall'analisi del malware.

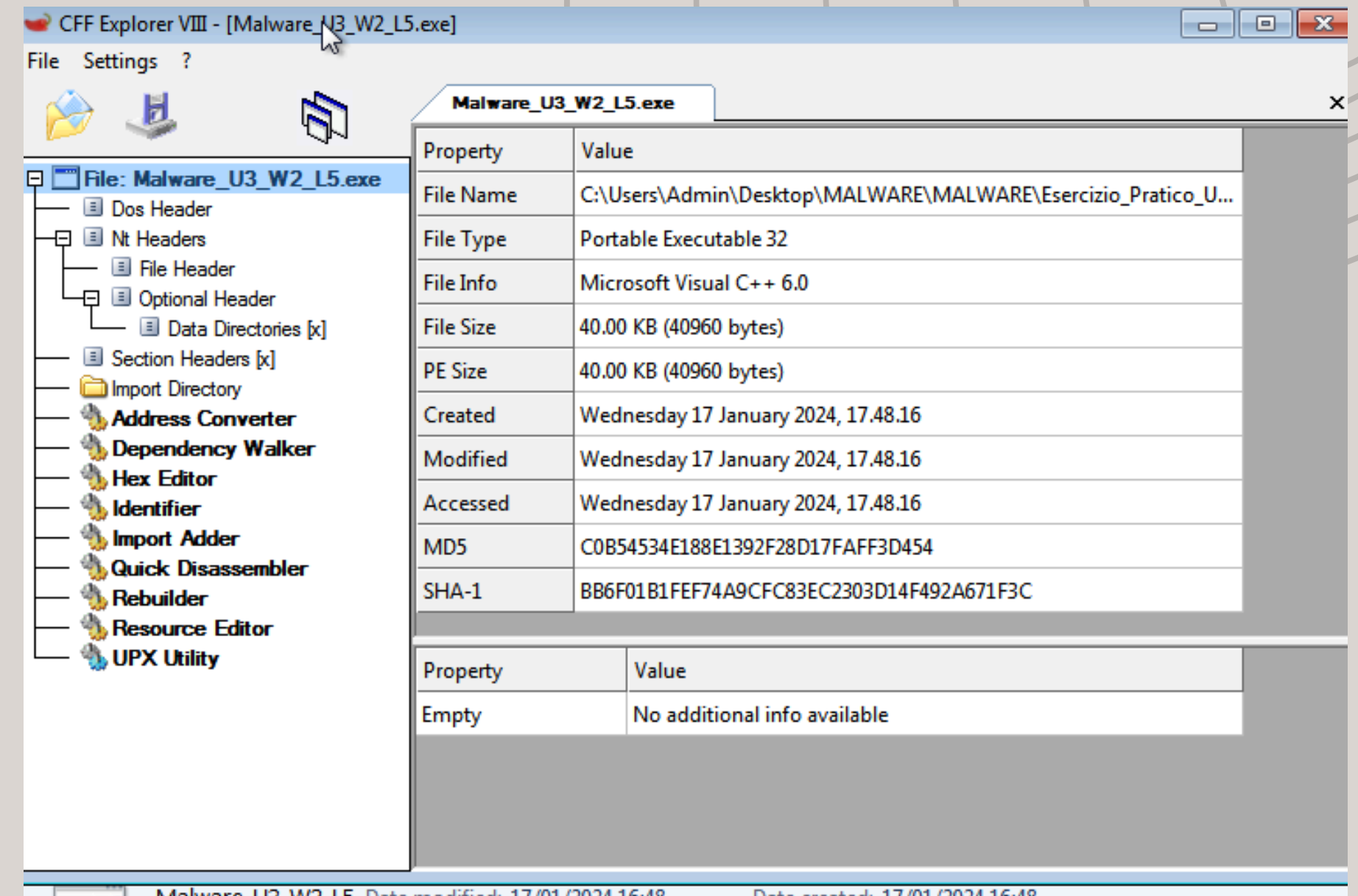
TASK 1 – CFF

Iniziamo con aprire il file malware in questione con CFF explorer e grazie ad esso, possiamo effettuare un'analisi statica del malware.

Qui si possono osservare varie caratteristiche, tra cui la data di creazione e modifica, utili per comprendere il contesto temporale del file.

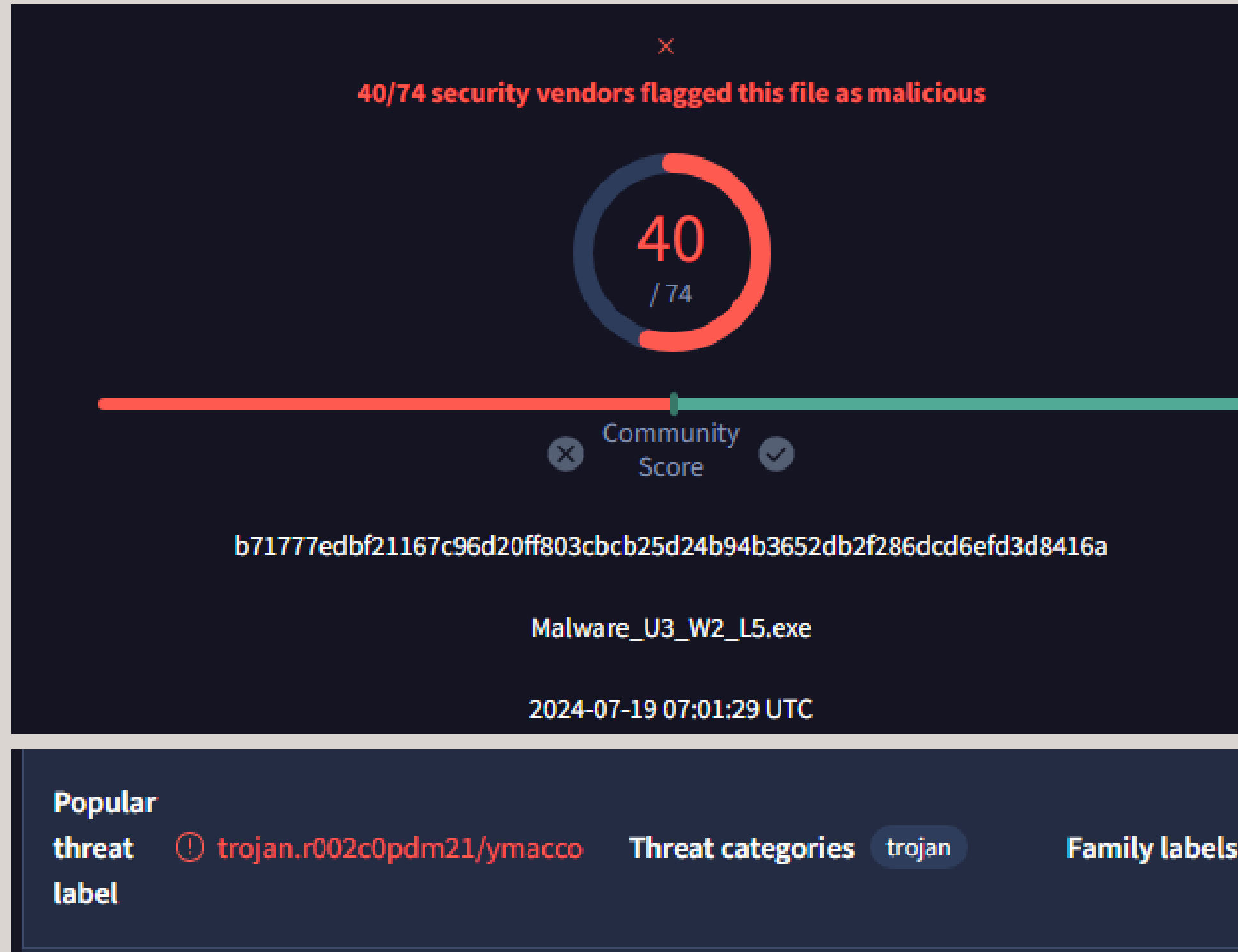
Inoltre, le informazioni sul tipo di file e sulle dimensioni possono offrire indicazioni sul comportamento e la provenienza del malware.

In particolare, impieghiamo gli hash MD5 e SHA-1 per identificare univocamente il file e confrontarlo con altre minacce conosciute.



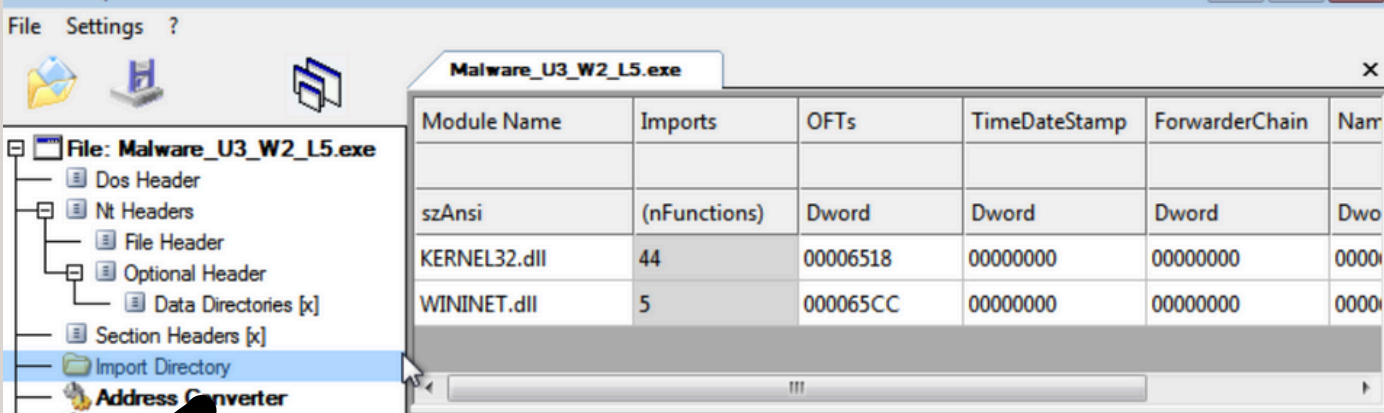
TASK 1 – VirusTotal

Per fare ciò andiamo su VirusTotal dove possiamo caricare il file ed effettivamente notiamo che è già stato identificato come Trojan:

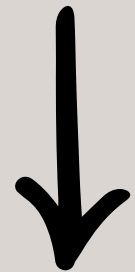


Task 1 – librerie

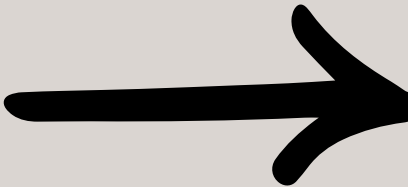
Per verificare le librerie e le funzioni importate, ci spostiamo su "import directory" nel menù a sinistra. Il pannello che apparirà a destra ci fornirà informazioni sulle librerie importate dall'eseguibile, mentre per ciascuna di esse, il pannello inferiore mostrerà l'elenco delle funzioni richieste all'interno della libreria selezionata.



| TimeDateStamp | ForwarderChain | Name RVA | FTs (IAT) |
|---------------|----------------|----------|-----------|
| | | | |
| Dword | Dword | Dword | Dword |
| 00000000 | 00000000 | 000065EC | 00006000 |
| 00000000 | 00000000 | 00006664 | 000060B4 |



| OFTs | FTs (IAT) | Hint | Name |
|----------|-----------|------|-------------------------|
| | | | |
| Dword | Dword | Word | szAnsi |
| 00006796 | 00006796 | 0115 | GetFileType |
| 000067A4 | 000067A4 | 0150 | GetStartupInfoA |
| 000067B6 | 000067B6 | 0126 | GetModuleHandleA |
| 000067CA | 000067CA | 0109 | GetEnvironmentVariableA |
| 000067E4 | 000067E4 | 0175 | GetVersionExA |
| 000067F4 | 000067F4 | 019D | HeapDestroy |
| 00006802 | 00006802 | 019B | HeapCreate |
| 00006810 | 00006810 | 02BF | VirtualFree |
| 0000681E | 0000681E | 019F | HeapFree |
| 0000682A | 0000682A | 022F | RtlUnwind |
| 00006836 | 00006836 | 02DF | WriteFile |
| 00006842 | 00006842 | 0199 | HeapAlloc |
| 0000684E | 0000684E | 00BF | GetCPInfo |
| 0000685A | 0000685A | 00B9 | GetACP |
| 00006864 | 00006864 | 0131 | GetOEMCP |
| 00006870 | 00006870 | 02BB | VirtualAlloc |
| 00006880 | 00006880 | 01A2 | HeapReAlloc |
| 0000688E | 0000688E | 013E | GetProcAddress |
| 000068A0 | 000068A0 | 01C2 | LoadLibraryA |
| 000068B0 | 000068B0 | 011A | GetLastError |



| | | | | | |
|-------------|-----------|----------|---------------------------|----------|----------|
| WININET.dll | 5 | 000065CC | 00000000 | 00000000 | 00006664 |
| 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 | | |

Task 1

Possiamo definire rapidamente le librerie (note anche come moduli) come insiemi di funzioni. Quando un programma necessita di una funzione, richiama una libreria in cui essa è definita. Gli eseguibili possono importare librerie e funzioni in tre modi differenti:

- **Importazione statica:** l'eseguibile copia l'intero contenuto della libreria nel proprio codice. Questo approccio aumenta la dimensione del file e, per l'analista, rende più difficile distinguere tra il codice della libreria e quello dell'eseguibile durante l'analisi statica avanzata.
- **Importazione a tempo di esecuzione (runtime):** l'eseguibile richiama la libreria solo quando necessita di una specifica funzione. Questo metodo è ampiamente utilizzato dai malware per ridurre la loro invasività e rilevabilità. Funzioni come **LoadLibrary** e **GetProcAddress**, fornite dal sistema operativo, vengono utilizzate per richiamare la libreria quando necessario.
- **Importazione dinamica:** è il metodo più comune e interessante per gli analisti di sicurezza. Le librerie importate dinamicamente vengono caricate dal sistema operativo all'avvio dell'eseguibile. La funzione richiesta viene chiamata ed eseguita all'interno della libreria solo quando necessaria.

Task 1

Nel caso del malware che stiamo analizzando, abbiamo a che fare con due librerie diverse:

Kernel32.dll: è una delle principali librerie di sistema di Windows e fornisce funzioni fondamentali per la gestione della memoria, dei processi e dei thread, dell'I/O (input/output) e di altre operazioni di sistema basilari.

È essenziale per il funzionamento delle applicazioni su Windows.

Funzioni comuni in KERNEL32.dll includono:

- creazione e gestione dei processi (CreateProcess);
- gestione della memoria (VirtualAlloc, VirtualFree);
- gestione dei file (CreateFile, ReadFile, WriteFile);
- operazioni su thread (CreateThread, ExitThread).

I malware spesso utilizzano KERNEL32.dll per eseguire operazioni fondamentali come creare nuovi processi o thread, manipolare file, allocare memoria, e molto altro.

L'uso intensivo di questa libreria è comune in quasi tutti i tipi di software, inclusi i malware

Wininet.dll: è una libreria che fornisce funzioni per l'accesso a Internet e per l'implementazione di protocolli come HTTP e FTP.

Questa libreria permette alle applicazioni di interagire con le risorse di rete e di effettuare operazioni come il download e l'upload di file via Internet.

Funzioni comuni in WININET.dll includono:

- connessione a server HTTP e FTP (InternetOpen, InternetConnect)
- richieste HTTP (HttpOpenRequest, HttpSendRequest)
- gestione di sessioni Internet (InternetOpenUrl, InternetCloseHandle)

I malware che necessitano di comunicare con server remoti per scaricare ulteriori payload, esfiltrare dati, o ricevere comandi spesso utilizzano WININET.dll. Questa libreria permette al malware di connettersi a Internet, scaricare file, inviare informazioni, e altro, senza dover implementare direttamente i complessi protocolli di rete.

TASK 2 – sezioni

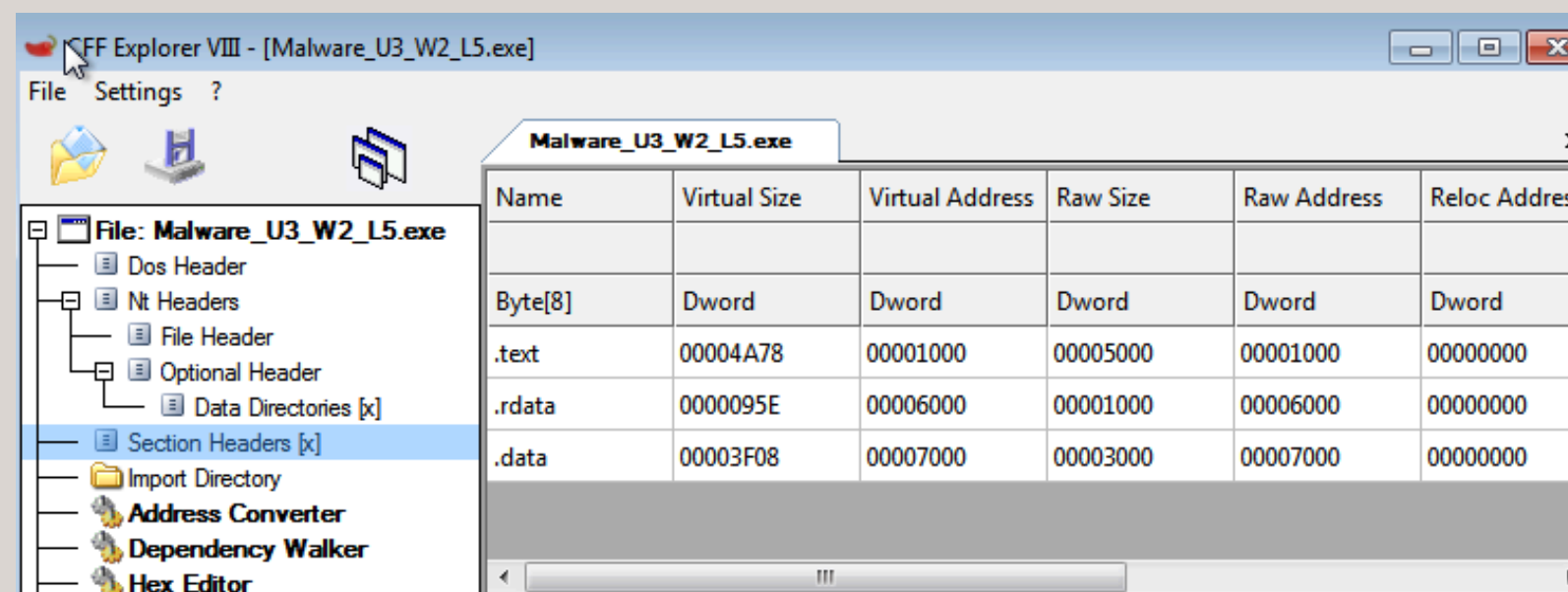
Conoscere le finalità di ogni sezione è un'informazione preziosa per condurre analisi, dunque per effettuare il controllo delle sezioni che compongono il malware in analisi, ci spostiamo nel menù "Section Header" presente sul pannello sinistro di CFF Explorer.

Per quanto riguarda le sezioni del file **eseguibile** PE (Portable Executable) e i loro attributi, possiamo identificare tre principali sezioni:
.text, **.rdata** e **.data**.

1. La **sezione .text** contiene il codice eseguibile del programma. Questo è il segmento in cui risiede il codice macchina che viene eseguito dalla CPU. Qualsiasi analisi o disassemblaggio del codice malevolo parte da questa sezione.
2. La **sezione .rdata** contiene dati di sola lettura. Spesso include le tabelle di importazione e le stringhe costanti che indicano quali funzioni di sistema o di altre librerie vengono utilizzate dal malware.
3. La **sezione .data** contiene dati inizializzati utilizzati dal programma, come variabili globali e statiche, accessibili in lettura e scrittura. Qui si trovano i dati modificabili che il malware utilizza durante la sua esecuzione.

È fondamentale fare notare che, sebbene queste siano le sezioni standard presenti in numerosi file eseguibili, le stesse I malware sofisticati possono manipolare o occultare tali sezioni, oppure possono persino creare altre nuove, usando nomi truffaldini o insoliti per evitare la scansione statica. Questo potrebbe generare confusione tra gli analisti o i ricercatori.

I software di sicurezza automatica operano cercando di individuare il comportamento malevolo basandosi sulla loro programmazione.
riguardo alla formattazione tipica di un file eseguibile



| Linenumbers | Relocations N... | Linenumbers ... | Characteristics |
|-------------|------------------|-----------------|-----------------|
| | | | |
| Dword | Word | Word | Dword |
| 00000000 | 0000 | 0000 | 60000020 |
| 00000000 | 0000 | 0000 | 40000040 |
| 00000000 | 0000 | 0000 | C0000040 |

TASK 3 – Assembly

L'analisi statica avanzata richiede la conoscenza di un linguaggio specifico chiamato Assembly. Il linguaggio Assembly è unico per ogni architettura di un PC e varia tra diverse architetture.

Durante l'analisi statica, l'analista di sicurezza utilizza strumenti chiamati "Disassembler", progettati per tradurre le istruzioni binarie eseguite dalla CPU in un formato più leggibile dall'uomo, ovvero il linguaggio Assembly.

La comprensione di Assembly permette di "leggere" le istruzioni eseguite dalla CPU in una forma comprensibile. Come già accennato, Assembly è un linguaggio che dipende dall'architettura del computer.

Le architetture più comuni includono x86, x64, ARM, MIPS e PowerPC.

È importante sapere che i processori possono essere a 32 o 64 bit, che rappresentano la quantità massima di informazioni che la CPU può gestire per ogni operazione.

Per il nostro progetto, ci concentreremo su Assembly per il set di istruzioni x86, cioè per processori a 32 bit.

La base del linguaggio Assembly sono le istruzioni, che consistono di due parti:

- **codice mnemonico:** una parola che identifica l'istruzione da eseguire.
- **operandi:** una o più entità che rappresentano le variabili o la memoria oggetto dell'istruzione.

Esistono tre tipi di operandi:

- **un valore:** come un numero, generalmente scritto in formato esadecimale (es. 0xYY dove YY è la versione esadecimale del numero decimale).
- **registri:** memorie rapide messe a disposizione della CPU.
- **indirizzo di memoria:** che contiene un valore di interesse.

Un registro è una piccola memoria ad accesso rapido (limitata nelle dimensioni ma con velocità di accesso superiore rispetto ad altre memorie come quelle secondarie) che consente di salvare temporaneamente una variabile necessaria alla CPU.

Task 3 - costrutti

```
push    ebp
mov     ebp, esp
push    ecx
push    0          ; dwReserved
push    0          ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

```
loc_40102B:          ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
add     esp, 4
xor     eax, eax
```

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```


Riquadro **rosso**: inizio della funzione e contesto

Nel linguaggio assembly x86, le istruzioni "**push ebp**" e "**mov ebp, esp**" vengono utilizzate all'inizio di una funzione per creare un contesto specifico. Questo contesto include la creazione di uno stack frame che ospita variabili locali e parametri della funzione. In particolare, "**push ebp**" salva il valore attuale del registro base sullo stack, mentre "**mov ebp, esp**" assegna al registro base lo stesso valore del puntatore dello stack (ESP). Questa operazione consente di accedere facilmente alle variabili locali e ai parametri della funzione utilizzando il registro base come punto di riferimento nello stack.

Riquadro **blu**: confronto e salto

Questo segmento di codice assembly x86 esegue un confronto (cmp) tra il valore memorizzato all'indirizzo di memoria [ebp+var_4] e il valore 0. Il risultato di questo confronto non viene utilizzato direttamente; piuttosto, un'istruzione di salto condizionato (jz) viene impiegata per saltare a una specifica etichetta (**loc_40102B**) solo se il risultato del confronto è zero. Se il confronto non risulta zero, l'esecuzione prosegue in modo lineare.

Riquadro **rosa**: ripristino dello stack frame

Nel linguaggio assembly x86, le istruzioni "**mov esp, ebp**" e "**pop ebp**" vengono usate alla fine di una funzione per riportare lo stack al suo stato originale, prima dell'esecuzione della funzione. La prima istruzione ripristina il puntatore dello stack (ESP) al valore precedente, che era stato salvato nel registro base (EBP) all'inizio della funzione. La seconda istruzione preleva il valore corrente dallo stack e lo carica nel registro base (EBP), ripristinando così il valore originale del registro base. In sostanza, queste istruzioni permettono di "pulire" lo stack frame creato per la funzione, garantendo che lo stack ritorni allo stato precedente all'esecuzione della funzione stessa.

Funzionalità

Questo codice Assembly verifica la disponibilità di una connessione Internet e fornisce un feedback all'utente in base all'esito.

Utilizza l'API `InternetGetConnectedState` per determinare lo stato della connessione.

Se una connessione è presente, visualizza un messaggio di successo e ritorna 1.

Se non c'è connessione, mostra un messaggio di errore e ritorna 0.

Il programma gestisce il flusso tramite istruzioni di confronto e salti condizionali.

In presenza di una connessione Internet, il programma potrebbe eseguire ulteriori operazioni, mentre in assenza di connessione, potrebbe terminare o tentare altri metodi di connessione.

Task 5

| Indirizzo | Istruzione | Descrizione |
|-----------|--------------------------------------|---|
| 0x401000 | push ebp | Salva il valore corrente del puntatore base (ebp) sullo stack per preservare il contesto del chiamante. |
| 0x401001 | mov ebp, esp | Imposta ebp al valore corrente del puntatore dello stack (esp), creando un nuovo stack frame. |
| 0x401003 | push ecx | Salva il registro ecx sullo stack per preservare il suo valore. |
| 0x401004 | call ds:InternetGetConnectedState | Chiama la funzione di sistema InternetGetConnectedState per verificare lo stato della connessione Internet. |
| 0x401006 | mov [ebp+var_4], eax | Salva il valore di ritorno della funzione InternetGetConnectedState nella variabile locale var_4 |
| 0x40100B | cmp [ebp+var_4], 0 | Confronta il valore della variabile locale var_4 con 0 per determinare se la connessione Internet è attiva. |
| 0x40100E | jz short loc_401028 | Se il risultato del confronto è zero (nessuna connessione), salta all'etichetta loc_401028. |

| Indirizzo | Istruzione | Descrizione |
|-----------|-----------------------------|--|
| 0x401014 | push offset aSuccessInterne | Spinge l'indirizzo del messaggio di successo ("Success: Internet Connection") sullo stack per passarlo alla funzione sub_40117F. |
| 0x401019 | call sub_40117F | Chiama una funzione (probabilmente per stampare il messaggio di successo). |
| 0x40101E | add esp, 4 | Ripulisce lo stack rimuovendo il parametro passato alla funzione sub_40117F |
| 0x401021 | mov eax, 1 | Imposta il valore di ritorno eax a 1, indicando successo. |
| 0x401023 | jmp short loc_40103A | Salta all'etichetta loc_40103A per chiudere lo stack frame e ritornare. |
| 0x401028 | loc_401028 | Etichetta per il codice eseguito in caso di connessione Internet non attiva. |
| 0x401028 | push offset aError1_1NoInte | Spinge l'indirizzo del messaggio di errore ("Error 1.1: No Internet") sullo stack per passarlo alla funzione sub_40117F |

| Indirizzo | Istruzione | Descrizione |
|-----------|-----------------|---|
| 0x40102D | call sub_40117F | Chiama una funzione (probabilmente per stampare il messaggio di errore) |
| 0x401032 | add esp, 4 | Ripulisce lo stack rimuovendo il parametro passato alla funzione sub_40117F. |
| 0x401035 | xor eax, eax | Imposta eax a 0, indicando fallimento |
| 0x40103A | loc_40103A | Etichetta per il codice eseguito per chiudere lo stack frame e ritornare. |
| 0x40103A | mov esp, ebp | Ripristina il puntatore dello stack (esp) al valore del puntatore base (ebp), preparando la chiusura dello stack frame. |
| 0x40103C | pop ebp | Ripristina il puntatore base (ebp) al suo valore precedente, chiudendo lo stack frame corrente |
| 0x40103D | ret | Ritorna al chiamante, utilizzando l'indirizzo salvato sullo stack. |

BONUS

Un giovane dipendente neo assunto segnala al reparto tecnico la presenza di un programma sospetto.

Il suo superiore gli dice di stare tranquillo ma lui non è soddisfatto e chiede supporto al SOC.

Il file "sospetto" è **iexplore.exe** contenuto nella cartella **C:\Programmi\Internet Explorer** (no, non ridete ragazzi)

Come membro senior del SOC ti è richiesto di convincere il dipendente che il file non è maligno.

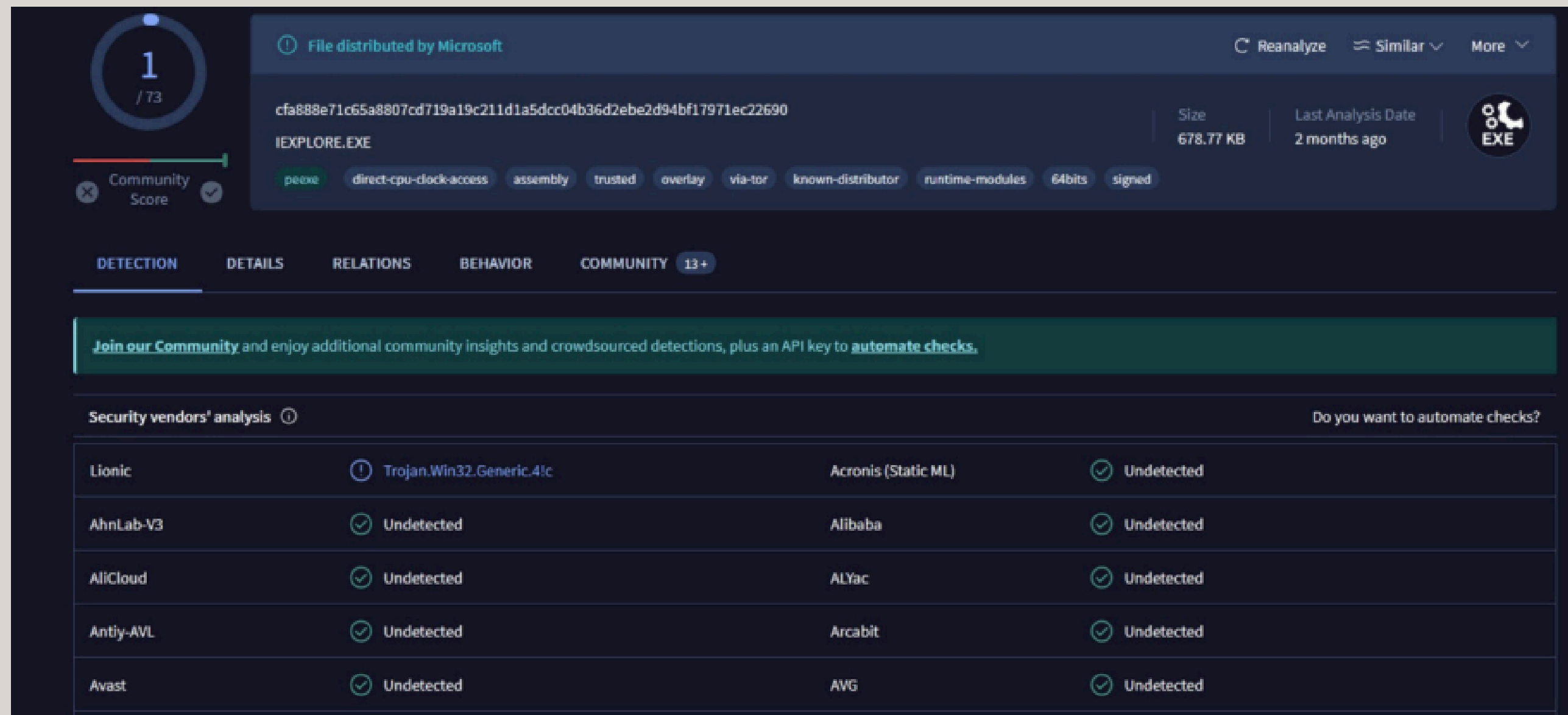
Esercizio Traccia e requisiti

Possono essere usati gli strumenti di analisi statica basica e/o analisi dinamica basica visti a lezione.

No disassembly no debug o similari VirusTotal non basta, ovviamente
Non basta dire iexplorer è Microsoft quindi è buono, punto.

BONUS

In primo luogo carichiamo il file su VirusTotal per una prima scansione, anche se non è sufficiente ma ci fornisce già un'idea sul file e la sua integrità. Da questa scansione non sembrerebbe essere un file malevolo.



The screenshot shows the VirusTotal interface for a file named IEXPLORE.EXE. The file is identified as being distributed by Microsoft. The scan shows no detections from any of the listed security vendors. The file is 678.77 KB and was last analyzed 2 months ago. The file type is EXE. The file hash is cfa888e71c65a8807cd719a19c211d1a5dcc04b36d2ebe2d94bf17971ec22690. The file is 64bits and signed. The file is not a known distributor, runtime module, or via-tor. The file is not a direct-cpu-dock-access or assembly. The file is trusted and overlay. The file is not a peexe. The file is not a direct-cpu-dock-access or assembly. The file is trusted and overlay. The file is not a peexe.

File distributed by Microsoft

Reanalyze Similar More

cfa888e71c65a8807cd719a19c211d1a5dcc04b36d2ebe2d94bf17971ec22690

Size: 678.77 KB Last Analysis Date: 2 months ago

EXE

EXPLORE.EXE

peexe direct-cpu-dock-access assembly trusted overlay via-tor known-distributor runtime-modules 64bits signed

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 13+

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis ⓘ Do you want to automate checks?

| | | | |
|-----------|----------------------------|---------------------|--------------|
| Lionic | ⓘ Trojan.Win32.Generic.4!c | Acronis (Static ML) | ✓ Undetected |
| AhnLab-V3 | ✓ Undetected | Alibaba | ✓ Undetected |
| AliCloud | ✓ Undetected | ALYac | ✓ Undetected |
| Antiy-AVL | ✓ Undetected | Arcabit | ✓ Undetected |
| Avast | ✓ Undetected | AVG | ✓ Undetected |

BONUS

In seguito possiamo procedere all'analisi statica del file IEXPLORER.exe con CFF EXPLORER.

| iexplore.exe | |
|--------------|---|
| Property | Value |
| File Name | C:\Program Files\Internet Explorer\iexplore.exe |
| File Type | Portable Executable 64 |
| File Info | Microsoft Visual C++ 8.0 (DLL) |
| File Size | 678.77 KB (695056 bytes) |
| PE Size | 672.00 KB (688128 bytes) |
| Created | Sunday 21 November 2010, 05.24.43 |
| Modified | Sunday 21 November 2010, 05.24.43 |
| Accessed | Sunday 21 November 2010, 05.24.43 |
| MD5 | 86257731DDB311FBC283534CC0091634 |
| SHA-1 | 2AA859F008FAFBAEFB578019ED0D65CD0933981C |

| Property | Value |
|------------------|---|
| CompanyName | Microsoft Corporation |
| FileDescription | Internet Explorer |
| FileVersion | 8.00.7601.17514 (win7sp1_rtm.101119-1850) |
| InternalName | iexplore |
| LegalCopyright | © Microsoft Corporation. All rights reserved. |
| OriginalFilename | IEXPLORE.EXE |
| ProductName | Windows® Internet Explorer |

| Module Name | Imports | OFTs | TimeStamp | ForwarderChain | Name RVA | FTs (IAT) |
|--------------|--------------|----------|-----------|----------------|----------|-----------|
| | | | | | | |
| szAnsi | (nFunctions) | Dword | Dword | Dword | Dword | Dword |
| ADVAPI32.dll | 13 | 0000F6B8 | FFFFFFFF | FFFFFFFF | 0000F6A8 | 00009000 |
| KERNEL32.dll | 56 | 0000F728 | FFFFFFFF | FFFFFFFF | 0000F698 | 00009070 |
| USER32.dll | 9 | 0000F8F0 | FFFFFFFF | FFFFFFFF | 0000F68C | 00009238 |
| msvcrt.dll | 29 | 0000F940 | FFFFFFFF | FFFFFFFF | 0000F680 | 00009288 |
| ntdll.dll | 3 | 0000FA30 | FFFFFFFF | FFFFFFFF | 0000F674 | 00009378 |
| SHLWAPI.dll | 23 | 0000FA50 | FFFFFFFF | FFFFFFFF | 0000F668 | 00009398 |
| SHELL32.dll | 7 | 0000FB10 | FFFFFFFF | FFFFFFFF | 0000F65C | 00009458 |
| ole32.dll | 5 | 0000FB50 | FFFFFFFF | FFFFFFFF | 0000F650 | 00009498 |
| iertutil.dll | 14 | 0000FB80 | FFFFFFFF | FFFFFFFF | 0000F640 | 000094C8 |
| urlmon.dll | 3 | 0000FBF8 | FFFFFFFF | FFFFFFFF | 0000F634 | 00009540 |

Le librerie in uso non sono sospette ma a seguito di una veloce ricerca sono le Dynamic Link Libraries (DLL) di Windows. Inoltre dalle generalità possiamo vedere e confrontare che l'hash è quello ufficiale di Microsoft.

CONCLUSION

In conclusione a seguito di queste analisi il file iexplorer.exe sembra essere autentico, probabilmente è una versione di Internet Explorer poiché:

1. Le DLL importate da iexplore.exe includono funzioni fondamentali del sistema operativo (KERNEL32.dll, ntdll.dll), gestione dell'interfaccia utente (USER32.dll) e funzioni avanzate di sicurezza e registro (ADVAPI32.dll).
2. L'importazione di DLL specifiche per Internet Explorer (iertutil.dll, urlmon.dll) e per la shell di Windows (SHLWAPI.dll, SHELL32.dll) suggerisce che iexplore.exe fa uso di componenti essenziali per operazioni di rete e gestione del file system.
3. L'importazione di msvcrt.dll indica l'uso di funzioni standard del C, comunemente impiegate in molte applicazioni Windows.

Purtroppo avendo avuto problemi alla VM non ho potuto fare altre analisi con RegShot o Procmon ma in un ambiente funzionante sarebbe ideale anche passare per quei tool.

S10-L5

Thank You

ELEONORA VIOLA