



# **PROGETTO S10L5**

## **MALWARE ANALYSIS**

**Simone La Porta**  
**CS0424IT**

# Table of contents

**01**

## **Malware analysis**

- Librerie importate
- Sezioni

**02**

## **Codice Assembly**

- Identificazione costrutti noti
- Ipotesi funzionamento
- Analisi riga per riga

**03**

## **BONUS: iexplore.exe**

# 1

# Malware analysis

# Traccia

Con riferimento al file denominato Malware\_U3\_W2\_L5, rispondere ai seguenti quesiti:

- Librerie importate:
  - Elencare tutte le librerie importate dal file eseguibile.
  - Fornire una breve descrizione della funzione di ciascuna libreria.
- Sezioni del file eseguibile:
  - Identificare e descrivere le sezioni del file eseguibile.
  - Spiegare la funzione e l'importanza di ciascuna sezione.

# Librerie importate

Per identificare le librerie importate dal malware viene utilizzato CFF Explorer. Vengono identificate le seguenti librerie all'interno della sezione Import Directory:

- KERNEL32.dll
- WININET.dll

The screenshot shows the CFF Explorer interface with the title bar "CFF Explorer VIII - [Malware\_U3\_W2\_L5.exe]". On the left, there's a tree view of file headers: File, Dos Header, Nt Headers, File Header, Optional Header, Data Directories [x], Section Headers [x], and Import Directory. The Import Directory node is expanded. A red arrow points from the "szAnsi" row in the main table to the "Imports" column for the KERNEL32.dll row. The main table has columns: Module Name, Imports, OFTs, TimeStamp, ForwarderChain, Name RVA, and FTs (IAT). The KERNEL32.dll row shows 44 imports, OFTs value 00006518, TimeStamp 00000000, ForwarderChain 00000000, Name RVA 000065EC, and FTs (IAT) 00006000. The WININET.dll row shows 5 imports, OFTs value 000065CC, TimeStamp 00000000, ForwarderChain 00000000, Name RVA 00006664, and FTs (IAT) 000060B4.

Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

# Libreria KERNEL32

Contiene funzioni essenziali per il funzionamento del sistema operativo e delle applicazioni Windows.

Principali aree di gestione:

- **Memoria** (allocazione e liberazione della memoria)
  - Funzioni: VirtualAlloc, VirtualFree, HeapCreate, HeapAlloc.
- **Processi e Thread** (creazione e gestione di processi e thread)
  - Funzioni: CreateProcess, CreateThread, TerminateProcess, TerminateThread.
- **Input/output** (operazioni di lettura, scrittura e gestione di file e dispositivi)
  - Funzioni: ReadFile, WriteFile, CreateFile, CloseHandle.
- **File** (manipolazione dei file nel file system)
  - Funzioni: CopyFile, DeleteFile, MoveFile.
- **Tempi e date** (fornire e manipolare informazioni temporali)
  - Funzioni: GetSystemTime, SetSystemTime, GetTickCount.
- **Risorse** (caricare, ottenere indirizzi di funzioni e liberare librerie dinamiche)
  - Funzioni: LoadLibrary, GetProcAddress, FreeLibrary.

# Libreria KERNEL32

Malware_U3_W2_L5.exe						
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064E0	000064E4	000064E8	000064EC
szAnsi	(nFunctions)	Dword	Dword	Dword	Word	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep
00006940	00006940	027C	SetStdHandle
0000692E	0000692E	0156	GetStringTypeW
0000691C	0000691C	0153	GetStringTypeA
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA
000068E6	000068E6	01E4	MultiByteToWideChar
00006670	00006670	00CA	GetCommandLineA
00006682	00006682	0174	GetVersion
00006690	00006690	007D	ExitProcess
0000669E	0000669E	029E	TerminateProcess
000066B2	000066B2	00F7	GetCurrentProcess

La libreria KERNEL32, parte integrante del sistema operativo Windows, fornisce una vasta gamma di funzionalità essenziali per la gestione delle risorse di sistema e l'esecuzione dei programmi.

Tra le 44 funzioni che importa, alcune delle più significative sono:

- **Sleep**: sospende temporaneamente l'esecuzione di un thread per un periodo specificato. Nel frattempo, permette l'esecuzione di altri thread, la gestione efficiente delle risorse di calcolo e libera tempo di CPU per altri processi.
- **CloseHandle**: API utilizzata per chiudere un handle, ossia un riferimento o puntatore a un oggetto kernel. Funzione cruciale per la gestione delle risorse, poiché rilascia le risorse allocate a un handle una volta che non sono più necessarie, prevenendo perdite di memoria e migliorando la stabilità del sistema.

# Libreria KERNEL32

- **GetProcAddress**: funzione che permette di ottenere un puntatore a una funzione specifica all'interno di una DLL (Dynamic Link Library) caricata in memoria. Viene utilizzata per chiamare funzioni esportate da una DLL in modo dinamico durante l'esecuzione del programma, anziché durante la fase di compilazione. Questo è particolarmente utile per plugin, estensioni o per caricare moduli opzionali senza ricompilare l'intera applicazione.
- **VirtualAlloc**: utilizzata principalmente per riservare o allocare memoria virtuale per un processo. Questa funzione consente di riservare una porzione di memoria virtuale senza dover allocare immediatamente la memoria fisica corrispondente. È fondamentale per la gestione efficiente della memoria, poiché permette di controllare l'allocazione delle risorse in modo più granulare e flessibile.
- **VirtualFree**: complementare a VirtualAlloc, viene utilizzata per liberare la memoria precedentemente allocata. Essenziale per la gestione della memoria dinamica, poiché garantisce che la memoria non più necessaria venga restituita al sistema, riducendo così il rischio di frammentazione e migliorando l'efficienza complessiva del sistema.

# Libreria KERNEL32

Queste funzioni, insieme ad altre importate da KERNEL32.dll, sono essenziali per il funzionamento efficace ed efficiente delle applicazioni Windows. Forniscono strumenti cruciali per la gestione delle risorse, la comunicazione tra processi e l'allocazione dinamica della memoria.

Probabile uso nel contesto del malware:

- **Persistenza**
  - Utilizzo delle funzioni di gestione dei processi e dei thread per rimanere attivo nel sistema anche dopo il riavvio.
- **Evasione**
  - Sfruttamento delle funzioni di gestione della memoria per nascondere il malware o offuscare il suo codice.
- **Comunicazione**
  - Utilizzo delle operazioni di input/output e gestione dei file per esfiltrare dati o comunicare con server di comando e controllo.

# Libreria WININET

Contiene funzioni per l'implementazione di protocolli di rete come HTTP, FTP e NTP.

Principali aree di gestione:

- **Richieste di rete** (inviare richieste HTTP ai server web)
  - Funzioni: InternetOpen, InternetConnect, HttpOpenRequest, HttpSendRequest.
- **Operazioni di rete** (download/upload di file tramite FTP e lettura dati da internet)
  - Funzioni: FtpGetFile, FtpPutFile, InternetReadFile.

Probabile uso nel contesto del malware:

- **Download/upload di payload**
  - Utilizzo delle funzioni di rete per scaricare componenti aggiuntivi o caricare dati rubati.
- **Comunicazione con Server C2**
  - Le richieste HTTP sono usate per comunicare con server di comando e controllo per ricevere istruzioni o inviare dati.

# Libreria WININET

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
00006664	N/A	000064F0	000064F4	000064F8	000064F0	00006500
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

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

WININET.dll offre funzionalità legate alla rete Internet, permettendo agli utenti di accedere facilmente a protocoli internet come HTTP, FTP e HTTPS.

Di seguito una panoramica delle 5 funzioni importate:

- **InternetOpenUrlA**: aprire un URL specificato usando un handle di sessione Internet. Questo permette al malware di accedere a risorse su Internet.
- **InternetCloseHandle**: chiudere un handle utilizzato in una sessione Internet. Importante per la gestione delle risorse e per evitare perdite di memoria.
- **InternetReadFile**: leggere dati da un handle di un file Internet. È utilizzato per scaricare dati da Internet, che potrebbero includere aggiornamenti del malware o nuovi payload.

# Libreria WININET

- **InternetGetConnectedState**: verificare la connessione a Internet del sistema. Il malware può utilizzare questa funzione per decidere quando attivare o eseguire determinate azioni a seconda della disponibilità della connessione Internet.
- **InternetOpenA**: iniziare una sessione Internet che può essere utilizzata per accedere a risorse Internet. È il primo passo per stabilire una connessione Internet dal programma.

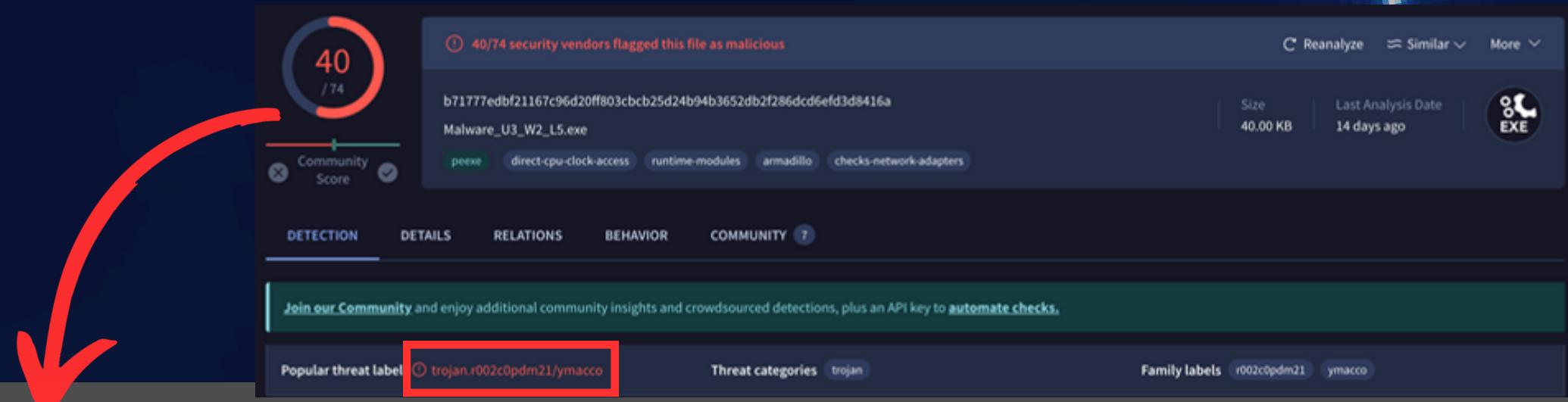
Ognuna di queste funzioni fornisce capacità fondamentali per un malware che vuole interagire con la rete, eseguire download/upload di dati, o semplicemente monitorare lo stato della connessione Internet del dispositivo infetto. Queste API sono strumenti comuni utilizzati nei software dannosi per realizzare varie attività malevole tramite la rete.

# Sezioni del malware

- Sezione **.text** contiene le istruzioni che la CPU eseguirà una volta avviato il software. Generalmente, questa è l'unica sezione di un file eseguibile che viene eseguita direttamente dalla CPU, mentre tutte le altre sezioni contengono dati o informazioni di supporto.
- Sezione **.rdata** include generalmente le informazioni sulle librerie e le funzioni importate ed esportate dall'eseguibile.
- Sezione **.data** contiene tipicamente i dati e le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi scope del programma.

Name	Virtual size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
<b>.text</b>	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
<b>.rdata</b>	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
<b>.data</b>	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

# Ricerca VirusTotal



Da ricerca su VirusTotal tramite HASH ricavato da CFF Explorer, il file eseguibile risulta un malware Trojan: tipico software dannoso che si maschera da applicazione legittima per ingannare gli utenti e indurli a installarlo.

Una volta attivato, può eseguire diverse attività malevoli, tra cui:

- Accesso remoto non autorizzato al sistema infetto.
- Furto di informazioni sensibili come credenziali, dati finanziari e personali.
- Installazione di altri software dannosi, come ransomware o spyware.
- Monitoraggio delle attività dell'utente, come la digitazione di tasti (keylogging).
- Alterazione delle impostazioni di sistema o file per compromettere la sicurezza o la funzionalità del dispositivo.

I trojan sono spesso distribuiti attraverso email phishing, download da siti web compromessi o allegati in programmi apparentemente innocui.

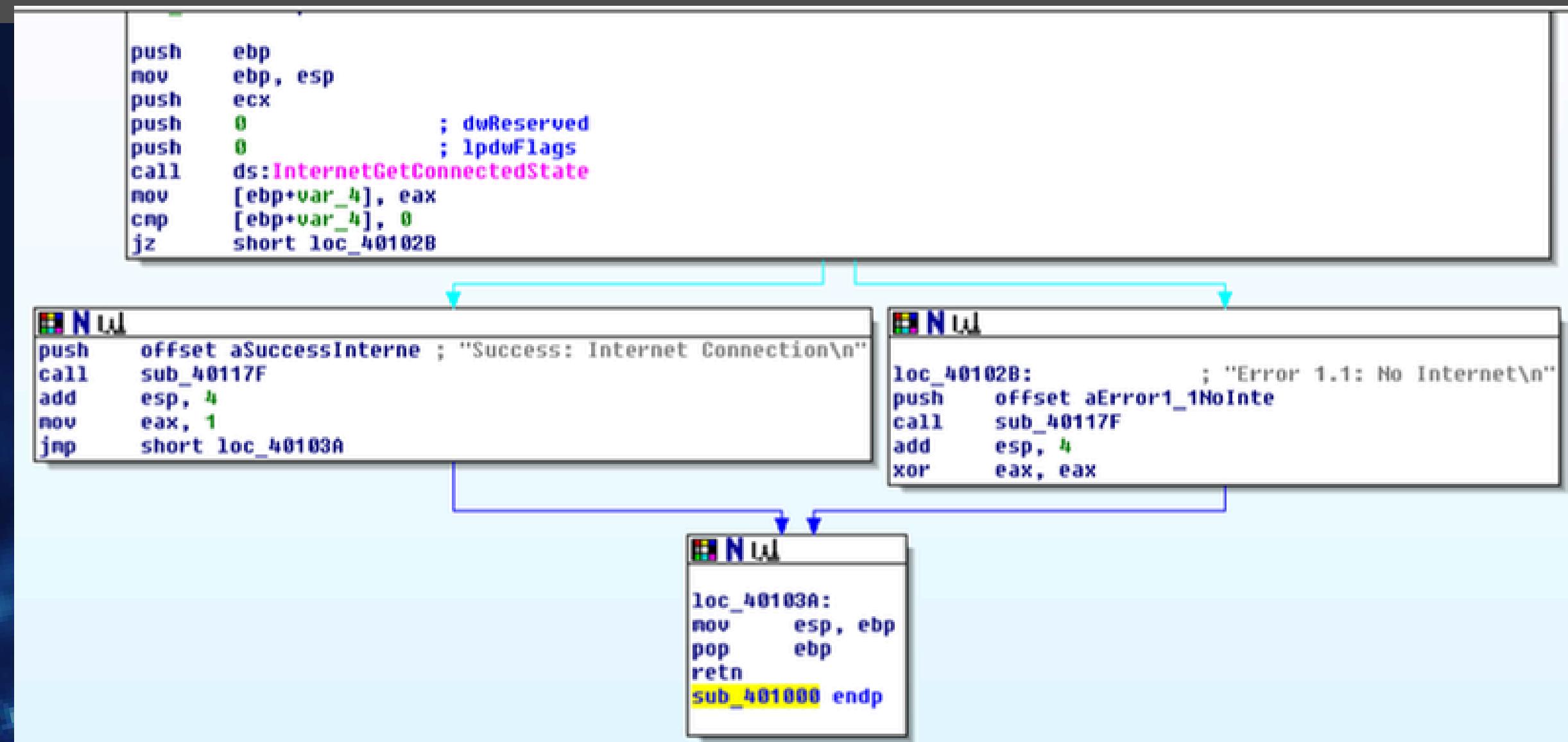
2

# Codice Assembly

# Traccia

Utilizzando la figura fornita, rispondere ai seguenti quesiti relativi al codice visualizzato.

- Identificazione dei costrutti noti.
- Formulare un'ipotesi sul comportamento e sulla funzionalità del codice analizzato.
- Creare una tabella che spieghi il significato di ciascuna riga di codice.



# Identificazione costrutti

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

Creazione dello stack frame.

```
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
```

Chiamata di funzione con passaggio di parametri sullo stack tramite *push*.

Confronto e salto condizionato: ciclo if.

```
NUL
```

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

```
NUL
```

```
loc_40102B:                      ; "Error 1.1: No Internet\n"  
push    offset aError1_NoInte  
call    sub_40117F  
add    esp, 4  
xor    eax, eax
```

Chiamata di funzione con passaggio di parametri sullo stack tramite *push*.

Eliminazione dello stack.

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

Chiamata di funzione con passaggio di parametri sullo stack tramite *push*.

# Identificazione costrutti

Chi scrive malware non si concentra sulla singola riga di codice, ma struttura il codice in base alle funzionalità che desidera implementare, utilizzando **costrutti**.

Riconoscere questi costrutti in Assembly è una competenza fondamentale nel campo del **reverse engineering**, o ingegneria inversa. Questo processo consiste nel ricostruire le funzionalità ad alto livello di un malware partendo dall'analisi del codice assembly, permettendo di comprendere il comportamento e gli obiettivi del software dannoso.

**push**  
**mov**      **ebp**      **ebp, esp**

## Creazione dello stack

Comandi utilizzati per creare lo stack frame, salvando il puntatore base precedente (ebp) e assegnando il puntatore stack (esp) al puntatore base (ebp). Viene creato uno stack di dimensione non specificata.

# Identificazione costrutti

```
push    ecx  
push    0          ; dwReserved  
push    0          ; lpdwFlags  
call    ds:InternetGetConnectedState
```

## Chiamata di funzione

Istruzioni per posizionare valori sullo stack che saranno usati come parametri per la chiamata a funzione InternetGetConnectedState, per verificare la connettività ad Internet.

```
cmp    [ebp+var_4], 0  
jz     short loc_40102B
```

## Confronto e salto condizionato

Formano una struttura condizionale (ciclo if) che controlla se il risultato di InternetGetConnectedState è zero (nessuna connessione) e, in caso, salta a un blocco di codice per gestire questa situazione.

# Identificazione costrutti

```
push    offset aSuccessInternet ; "Success: Internet Connection\n"
call    sub_40117F
```

## Chiamata di funzione

Chiamata di funzione con relativo passaggio di parametri.  
Gestione della memoria: *push offset aSuccessInternet* posiziona l'indirizzo di stringhe costanti sullo stack, che vengono poi passate alla subroutine di stampa.

```
loc_40102B:          ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
```

## Chiamata di funzione

Altra gestione della memoria: *push offset aError11NoInter* posiziona l'indirizzo di stringhe costanti sullo stack, che vengono poi passate alla subroutine di stampa.

# Identificazione costrutti

```
mov      esp, ebp  
pop      ebp  
retn
```

## Eliminazione dello stack

Qui avviene l'eliminazione dello stack: *mov esp, ebp* e *pop ebp* fanno parte dell'epilogo della funzione, che ripristina il frame precedente alla fine della funzione. *retn* termina la funzione e ritorna il controllo al chiamante.

# Ipotesi comportamento

L'obiettivo principale di questo programma è determinare lo stato della connessione Internet del sistema e loggare il risultato. Questo tipo di verifica può essere utilizzato in diverse applicazioni, incluso il monitoraggio di rete, la diagnostica di sistema, o come parte di un più ampio insieme di funzioni in software legittimi o anche in malware.

## Funzionamento ad alto livello

- Il programma inizia preparando l'ambiente di esecuzione creando un nuovo frame dello stack. Questo passaggio implica salvare i registri attuali per garantire che possano essere ripristinati una volta completata l'esecuzione della funzione.
- Il programma prepara e passa i parametri necessari alla funzione *InternetGetConnectedState*, che è una API di Windows utilizzata per determinare se il computer è connesso a Internet.
- Viene chiamata *InternetGetConnectedState*, che controlla lo stato della connessione Internet. Questa funzione ritorna un valore che indica se c'è una connessione attiva o meno.

# Ipotesi comportamento

## Funzionamento ad alto livello

- Il risultato della funzione (*InternetGetConnectedState*) viene memorizzato in una variabile locale.
- Il programma quindi confronta questo risultato con zero:
  - Se il risultato è zero, significa che non c'è connessione Internet e il programma procede a loggare un messaggio di errore ("Error 1.1: No Internet").
  - Se il risultato è diverso da zero, significa che c'è una connessione Internet attiva e il programma logga un messaggio di successo ("Success: Internet Connection").
- Infine, il programma ripristina lo stack e i registri al loro stato originale prima di *retn*, assicurando che il sistema possa continuare ad operare correttamente.

# Analisi riga per riga

<b>push ebp</b>	Salva il registro ebp (Base Pointer) corrente sullo stack. Questo è il primo passo nella creazione di un nuovo frame dello stack per la funzione chiamata.
<b>mov ebp, esp</b>	Imposta ebp (Base Pointer) al valore corrente di esp (Stack Pointer). Questo passo configura il nuovo frame dello stack, permettendo di riferirsi ai parametri e variabili locali della funzione in modo ordinato.
<b>push ecx</b>	Salva il registro ecx sullo stack. ecx è un registro general-purpose e potrebbe essere usato per conservare un valore temporaneo o preservare il suo valore per la funzione chiamata.
<b>push 0 ; dwReserved</b>	Carica il valore 0 sullo stack. Questo valore è passato come parametro dwReserved alla funzione InternetGetConnectedState.
<b>push 0 ;lpdwFlags</b>	Carica un altro valore 0 sullo stack. Questo valore è passato come parametro lpdwFlags alla funzione InternetGetConnectedState.
<b>call ds:InternetGetConnectedState</b>	Chiama la funzione InternetGetConnectedState, che verifica lo stato della connessione Internet del sistema. I parametri sono passati tramite lo stack.

# Analisi riga per riga

<b>mov [ebp+var_4], eax</b>	Copia il risultato della chiamata alla funzione InternetGetConnectedState dal registro eax alla variabile locale var_4.
<b>cmp [ebp+var_4], 0</b>	Confronta il valore della variabile locale var_4 con 0. Questo serve per verificare se la connessione a Internet è presente (var_4 sarà diverso da zero) o assente (var_4 sarà uguale a zero).
<b>jz short loc_40102B</b>	Se il confronto precedente risulta vero (cioè var_4 è zero), il controllo passa all'etichetta loc_40102B, indicando un'assenza di connessione Internet.
<b>push offset aSuccessInterne ; "Success: Internet Connection\n"</b>	Pusha l'offset del messaggio di successo (“Success: Internet Connection”) sullo stack. Questo messaggio sarà loggato/stampato se la connessione Internet è presente.
<b>call sub_40117F</b>	Chiama una subroutine (sub_40117F) per loggare/stampare il messaggio di successo.
<b>add esp, 4</b>	Incrementa il puntatore stack esp di 4 byte, liberando lo spazio utilizzato per il parametro del messaggio di successo.

# Analisi riga per riga

<b>mov eax, 1</b>	Imposta il registro eax a 1, indicando che la funzione ha avuto successo.
<b>jmp short loc_40103A</b>	Salto incondizionato di breve distanza ("short", -128/+127 byte) all'etichetta loc_40103A, che è alla fine della funzione, bypassando il codice di errore.
<b>loc_40102B:</b>	Etichetta per il blocco di codice che gestisce l'assenza di connessione Internet.
<b>push offset aError1_1NoInte ; "Error 1.1: No Internet\n"</b>	Pusha l'offset del messaggio di errore ("Error 1.1: No Internet") sullo stack. Questo messaggio sarà loggato/stampato se la connessione Internet è assente.
<b>call sub_40117F</b>	Chiama una subroutine (sub_40117F) per loggare/stampare il messaggio di errore.
<b>add esp, 4</b>	Incrementa il puntatore stack esp di 4 byte, liberando lo spazio utilizzato per il parametro del messaggio di errore.
<b>xor eax, eax</b>	Imposta il registro eax a 0 tramite la funzione logica XOR, indicando che la funzione non ha avuto successo (errore).

# Analisi riga per riga

<b>loc_40103A:</b>	Etichetta per la fine della funzione.
<b>mov esp, ebp</b>	Ripristina il puntatore stack esp al valore del puntatore base ebp, smantellando il frame dello stack creato all'inizio della funzione.
<b>pop ebp</b>	Ripristina il registro ebp con il valore salvato all'inizio della funzione.
<b>retn</b>	Ritorna dalla funzione, ripristinando il puntatore di istruzione al valore precedente salvato nello stack.
<b>sub_401000 endp</b>	Marca la fine della subroutine.

# **3 BONUS**

# Traccia

Un giovane dipendente neo assunto ha segnalato la presenza di un file sospetto denominato **iexplore.exe** situato nella directory C:\Programmi\Internet Explorer.

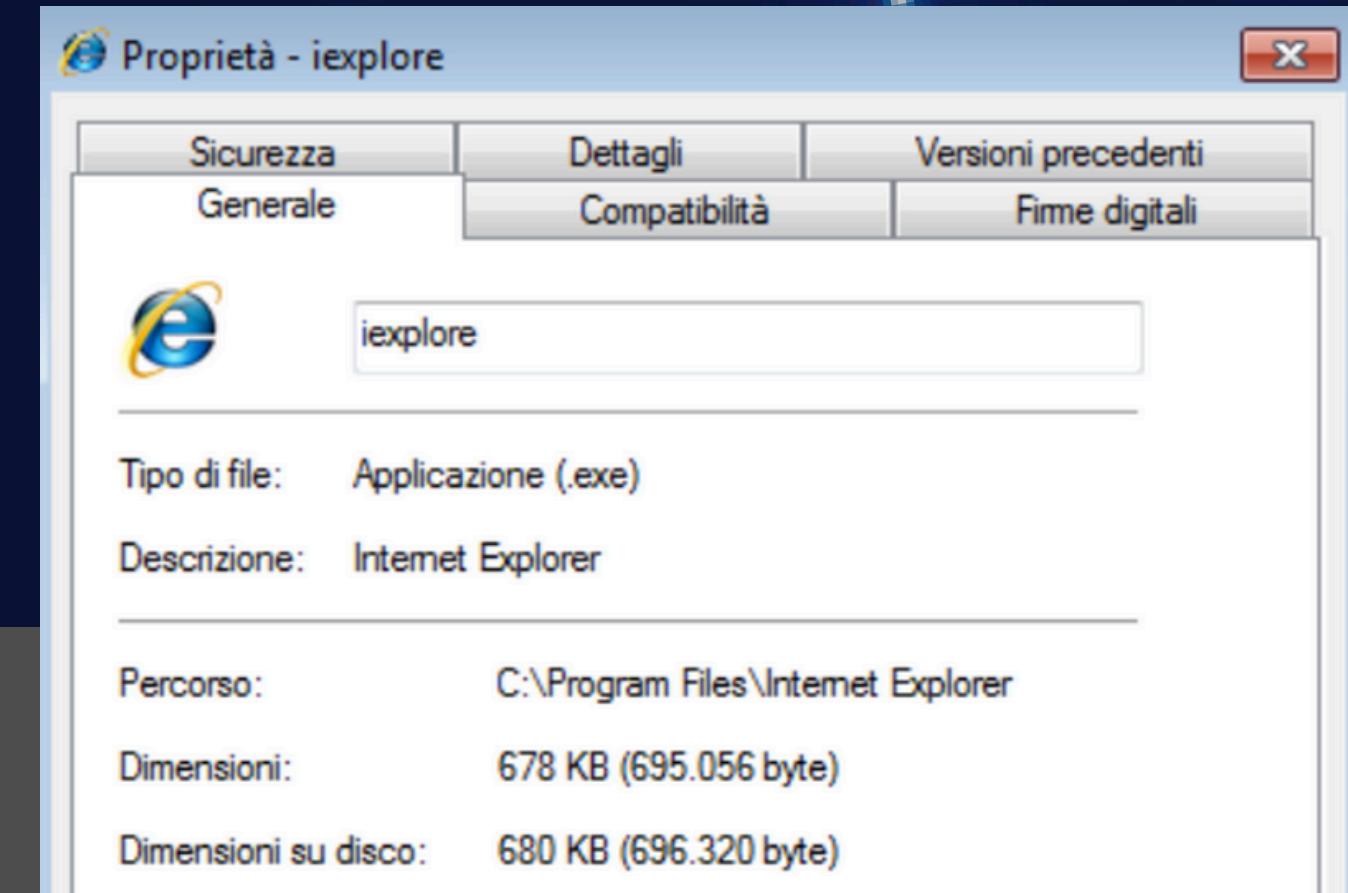
Sebbene il superiore abbia rassicurato il dipendente, è stato richiesto un supporto al Security Operations Center (SOC) per confermare che il file non sia maligno.

Come membro senior del SOC, il tuo compito è di condurre un'analisi dettagliata per convincere il dipendente della non pericolosità del file senza l'uso di disassemblatori, debug o strumenti simili.

# Analisi statica basica

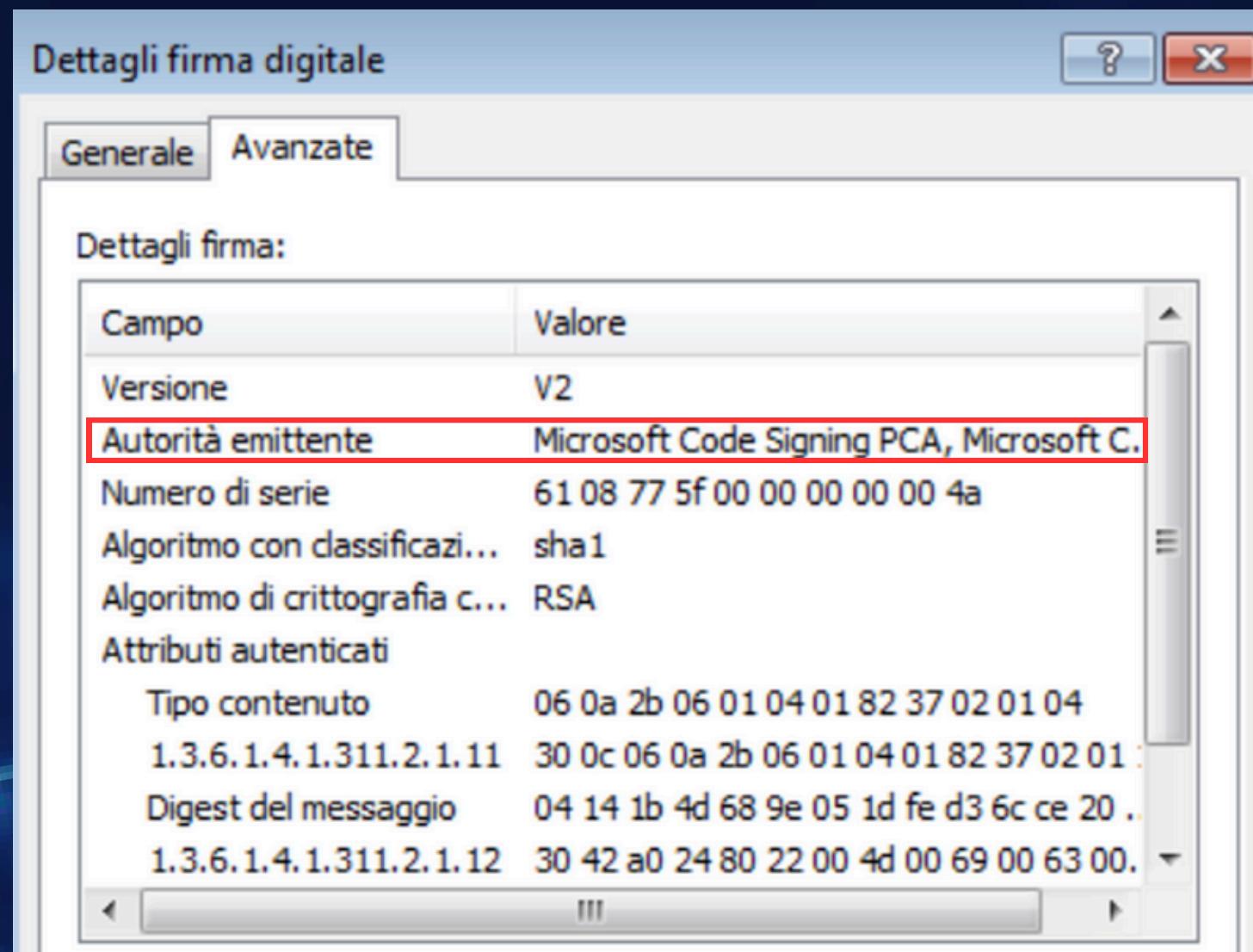
Metodi per l'analisi statica del file:

- Controllo proprietà del file:
  - Verificare che la dimensione del file corrisponda alle dimensioni tipiche del file di Internet Explorer distribuito da Microsoft.
  - Controllare le proprietà del file per assicurarsi che la versione corrisponda a una versione ufficiale di Internet Explorer.
  - Verificare che le date di creazione/modifica siano coerenti con una normale installazione di Internet Explorer.



# Analisi statica basica

- Verifica della firma digitale:
  - Utilizzare lo strumento integrato di Windows per controllare la firma digitale del file. Un file iexplore.exe autentico dovrebbe essere firmato da Microsoft Corporation.
  - Esaminare i dettagli della firma per assicurarsi che sia valida e non contraffatta.



# Analisi statica basica

- Controllo del hash del file:
  - Utilizzare strumenti come CertUtil o Get-FileHash di PowerShell per calcolare l'hash SHA-256 del file.
  - Confrontare l'hash calcolato con l'hash ufficiale del file iexplore.exe distribuito da Microsoft, disponibile nei database di sicurezza o tramite il sito ufficiale di Microsoft.

```
PS C:\Windows\system32> certutil -hashfile "C:\Programmi\Internet Explorer\iexplore.exe" SHA256
Hash SHA256 del file C:\Programmi\Internet Explorer\iexplore.exe:
cf a8 88 e7 1c 65 a8 80 7c d7 19 a1 9c 21 1d 1a 5d cc 04 b3 6d 2e be 2d 94 bf 17 97 1e c2 26 90
CertUtil: - Esecuzione comando hashfile riuscita.
```



The screenshot shows the VirusTotal analysis interface. At the top, there is a search bar with the hash value: `cfa888e71c65a8807cd719a19c211d1a5dcc04b36d2ebe2d94bf17971ec22690`. Below the search bar, a large green checkmark icon indicates a clean scan. The main card displays the following information:

- 1 / 73**: One file distributed by Microsoft.
- cfa888e71c65a8807cd719a19c211d1a5dcc04b36d2ebe2d94bf17971ec22690**: Hash value.
- IEXPLORE.EXE**: File name.
- Community Score**: A progress bar showing a high score.
- Size**: 678.77 KB.
- Last Analysis Date**: 2 months ago.
- File Type**: EXE.
- Tags**: peexe, direct-cpu-clock-access, assembly, trusted, overlay, via-tor, known-distributor, runtime-modules, 64bits, signed.

# Analisi dinamica basica

- Monitoraggio del comportamento del file
  - Utilizzare strumenti come Process Monitor (ProcMon) di Sysinternals per osservare l'attività del file iexplore.exe.
  - Monitorare la creazione di file, modifiche al registro di sistema e altre attività che il file esegue al momento dell'apertura. Un comportamento normale dovrebbe includere attività legate alla navigazione web e accesso a risorse di sistema standard.

Time ...	Process Name	PID	Operation	Path	Result	Detail
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 3.584, Len...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 0, Length: 5...
16:28:...	iexplore.exe	4980	SetEndOfFileIn...	C:\Users\user\AppData\Local\Micro...	SUCCESS	EndOfFile: 4.096
16:28:...	iexplore.exe	4980	SetAllocationIn...	C:\Users\user\AppData\Local\Micro...	SUCCESS	AllocationSize: 4.096
16:28:...	iexplore.exe	4980	UnlockFileSingle	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 2.147.483.5...
16:28:...	iexplore.exe	4980	QueryStandardI...	C:\Users\user\AppData\Local\Tem...	SUCCESS	AllocationSize: 16...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Tem...	SUCCESS	Offset: 12.288, Len...
16:28:...	iexplore.exe	4980	LockFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Exclusive: True, Of...
16:28:...	iexplore.exe	4980	QueryStandardI...	C:\Users\user\AppData\Local\Micro...	SUCCESS	AllocationSize: 4.0...
16:28:...	iexplore.exe	4980	QueryBasicInfor...	C:\Users\user\AppData\Local\Micro...	SUCCESS	Creation Time: 02/0...
16:28:...	iexplore.exe	4980	SetEndOfFileIn...	C:\Users\user\AppData\Local\Micro...	SUCCESS	EndOfFile: 4.608
16:28:...	iexplore.exe	4980	SetAllocationIn...	C:\Users\user\AppData\Local\Micro...	SUCCESS	AllocationSize: 4.608
16:28:...	iexplore.exe	4980	ReadFile	C:\Users\user\AppData\Local\Tem...	SUCCESS	Offset: 12.288, Len...
16:28:...	iexplore.exe	4980	ReadFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 3.072, Len...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 4.608, Len...
16:28:...	iexplore.exe	4980	ReadFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 4.096, Len...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 512, Length...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 2.560, Len...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 4.096, Len...
16:28:...	iexplore.exe	4980	WriteFile	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 0, Length: 5...
16:28:...	iexplore.exe	4980	SetEndOfFileIn...	C:\Users\user\AppData\Local\Micro...	SUCCESS	EndOfFile: 5.120
16:28:...	iexplore.exe	4980	SetAllocationIn...	C:\Users\user\AppData\Local\Micro...	SUCCESS	AllocationSize: 5.120
16:28:...	iexplore.exe	4980	UnlockFileSingle	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 2.147.483.5...
16:28:...	iexplore.exe	4980	UnlockFileSingle	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 2.147.483.5...
16:28:...	iexplore.exe	4980	UnlockFileSingle	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 2.147.483.5...
16:28:...	iexplore.exe	4980	UnlockFileSingle	C:\Users\user\AppData\Local\Micro...	SUCCESS	Offset: 2.147.483.5...
16:28:...	iexplore.exe	4980	CloseFile	C:\Users\user\AppData\Local\Tem...	SUCCESS	Offset: 2.147.483.5...

Nessun  
comportamento/processo  
anomalo.



# Analisi dinamica basica

- Analisi del network traffic
  - Utilizzare strumenti come Wireshark o ApateDNS per monitorare il traffico di rete generato dal file iexplore.exe.
  - Un file iexplore.exe legittimo dovrebbe comunicare solo con server noti e legittimi legati a Microsoft e ai servizi di Internet Explorer.

ApateDNS

Capture Window DNS Hex View

Time	Domain Requested	DNS Returned
16:33:44	6a10189edef67029a1abb9ad0a9ae69b.clo.footprintdns.com	FOUND
16:33:45	c89a75e61ac2a4e92a1ebff85a63732.clo.footprintdns.com	FOUND
16:33:46	aa332fd82be6a06029cee63df0af56e6.clo.footprintdns.com	FOUND
16:33:47	teredo.ipv6.microsoft.com	FOUND

Nessun comportamento di rete anomalo filtrando i pacchetti che hanno come sorgente il nostro IP e come protocollo HTTP/HTTPS.

No.	Time	Source	Destination	Protocol	Length	Info
531	0.250212	10.0.2.15	20.191.45.158	TLSv1	188	Client Hello
533	0.292864	20.191.45.158	10.0.2.15	SSLv3	61	Alert (Level: Fatal, Description: Handshake Failure)
539	0.294642	20.191.45.158	10.0.2.15	TLSv1	61	Alert (Level: Fatal, Description: Protocol Version)
552	0.339534	10.0.2.15	20.191.45.158	SSLv3	112	Client Hello
557	0.380848	20.191.45.158	10.0.2.15	SSLv3	61	Alert (Level: Fatal, Description: Handshake Failure)
561	0.381709	10.0.2.15	20.191.45.158	TLSv1	188	Client Hello
567	0.422790	20.191.45.158	10.0.2.15	TLSv1	61	Alert (Level: Fatal, Description: Protocol Version)
579	0.469319	10.0.2.15	20.191.45.158	TLSv1	188	Client Hello
583	0.512990	20.191.45.158	10.0.2.15	TLSv1	61	Alert (Level: Fatal, Description: Protocol Version)
591	0.556191	10.0.2.15	20.191.45.158	SSLv3	112	Client Hello
593	0.595985	20.191.45.158	10.0.2.15	SSLv3	61	Alert (Level: Fatal, Description: Handshake Failure)
607	9.728813	10.0.2.15	104.104.52.80	HTTP	2203	GET /fd/ls/1?IG=C2485B0064574137AE75837388032088&TYPE=Event.ClientInst&DATA=\$50%70%22Text%22%3A..
610	9.816476	104.104.52.80	10.0.2.15	HTTP	289	HTTP/1.1 200 OK



# Conclusioni

È stato analizzato in dettaglio il file sospetto e si può concludere con certezza che si tratta di un file eseguibile legittimo e non malevolo.

In particolare:

- Si sono verificate le date di creazione/modifica del file e sono consistenti rispetto alla data di installazione del sistema operativo.
- Si è controllata la firma digitale (autentica Microsoft) e l'hash SHA256 del file, confrontandolo con l'originale online.
- Si è controllato il funzionamento del file eseguibile in maniera dinamica, analizzando il suo comportamento tramite Process Monitor e Wireshark, senza riscontrare alcun tipo di comportamento malevolo o comunicazione verso canali illegittimi.

Si può quindi concludere che il file sia legittimo e possa essere avviato in sicurezza.





**Thank you!**

