

# **Malware Analysis**

Quasi tutto quello che c'è da sapere

Luca Gugole

12 gennaio 2022

# Indice

<b>1</b>	<b>Introduzione ai malware e ai tipi di analisi</b>	<b>2</b>
1.1	Tipologie di malware . . . . .	2
1.2	Cyber Kill Chain . . . . .	3
1.3	Analisi dei malware . . . . .	4
1.4	Comportamenti dei malware . . . . .	5
1.5	Preparazione dell'ambiente . . . . .	5
1.6	Reperire i malware . . . . .	6
<b>2</b>	<b>Analisi statica di base</b>	<b>7</b>
2.1	Identificazione del file . . . . .	7
2.2	Calcolo dell'hash . . . . .	8
2.3	Librerie e funzioni importate . . . . .	9
2.4	Analisi delle stringhe . . . . .	10
2.5	Data di compilazione . . . . .	11
2.6	Strumenti per l'analisi statica di base . . . . .	11
<b>3</b>	<b>Analisi dinamica di base</b>	<b>17</b>
3.1	Modifica del file system . . . . .	17
3.2	Modifica dei registri . . . . .	17
3.3	Attività di rete del malware . . . . .	18
3.4	Esecuzione di ulteriore codice . . . . .	20
3.5	System Integrity Monitoring . . . . .	21
3.6	Behavioural Monitoring . . . . .	23
3.7	Altri strumenti utili per l'analisi dinamica di base . . . . .	25
3.7.1	Simulare la presenza di una rete . . . . .	25
3.8	Come effettuare un'analisi dinamica di base completa . . . . .	26
3.8.1	Analisi del traffico di rete . . . . .	26
<b>4</b>	<b>Analisi statica avanzata</b>	<b>27</b>
4.1	Reverse Engineering . . . . .	27
4.1.1	General Stack Frame . . . . .	27
4.1.2	Riconoscere le funzioni in Assembly . . . . .	27
<b>5</b>	<b>Analisi dinamica avanzata</b>	<b>31</b>
5.1	Debugging . . . . .	31
<b>6</b>	<b>Analisi di script e documenti</b>	<b>32</b>
6.1	Analisi di documenti PDF . . . . .	32
6.1.1	Struttura di un PDF . . . . .	32
6.1.2	Analisi di un PDF . . . . .	32
6.2	Analisi di codice Javascript . . . . .	32
<b>7</b>	<b>Tecniche di offuscamento</b>	<b>33</b>
7.1	Modifica delle stringhe . . . . .	33
7.2	Impacchettamento . . . . .	33
7.2.1	Come spaccettare un malware . . . . .	33
<b>8</b>	<b>Esempi pratici di Malware Analysis</b>	<b>34</b>

<b>9</b>	<b>Introduzione alla Malware Analysis per sistemi Unix</b>	<b>35</b>
9.1	Sistemi Linux . . . . .	35
9.2	Sistemi Mac . . . . .	35
<b>A</b>	<b>Sistema operativo Windows</b>	<b>36</b>
A.1	Librerie ed API in Windows . . . . .	36
A.2	DLL . . . . .	36
A.2.1	Tipi di dati . . . . .	36
A.2.2	Kernel32.dll . . . . .	36
A.2.3	Advapi32.dll . . . . .	36
A.2.4	User32.dll . . . . .	36
A.2.5	Gdi32.dll . . . . .	36
A.2.6	Ntdll.dll . . . . .	36
A.2.7	WSock32.dll . . . . .	36
A.2.8	Ws2_32.dll . . . . .	36
A.2.9	Wininet.dll . . . . .	37
A.3	Formato Portable Executable . . . . .	37
A.3.1	DOS Header . . . . .	37
A.3.2	PE Header . . . . .	37
A.3.3	File Header . . . . .	37
A.3.4	Optional Header . . . . .	37
A.3.5	Section Header . . . . .	38
A.3.6	Risorse . . . . .	38
A.4	Registro di sistema . . . . .	38
A.4.1	HKEY_CLASSES_ROOT . . . . .	38
A.4.2	HKEY_CURRENT_USER . . . . .	38
A.4.3	HKEY_LOCAL_MACHINE . . . . .	38
A.4.4	HKEY_USERS . . . . .	38
A.4.5	HKEY_CURRENT_CONFIG . . . . .	38
<b>B</b>	<b>Linguaggio Assembly</b>	<b>39</b>
B.1	Richiamo sull'architettura x86 . . . . .	39
B.2	Endianess . . . . .	39
B.3	Registri della CPU . . . . .	40
B.4	Istruzioni . . . . .	42
B.4.1	Istruzioni condizionali o roba simile . . . . .	44
B.4.2	Gestione dello stack . . . . .	45
B.5	Codice C vs. codice Assembly . . . . .	45
B.5.1	Call Convention: stdcall . . . . .	45
B.5.2	Strutture condizionali IF e IF-ELSE . . . . .	45
B.5.3	Struttura ciclica FOR . . . . .	45
B.5.4	Struttura ciclica WHILE . . . . .	45

# Capitolo 1

## Introduzione ai malware e ai tipi di analisi

La parola **malware** nasce dalla contrazione tra le parole *Malicious* e *Software* e può essere definito come un software o firmware che compie processi non autorizzati che hanno un diverso impatto sulla confidenzialità, integrità o disponibilità di un sistema informativo. Solo nell'ultimo decennio c'è stato un incremento dell'87% di infezioni causate da malware. Oggi il 93% degli attacchi è causato da ransomware.

I malware possono essere utilizzati essenzialmente da tre grandi categorie:

- **Insiders:** dipendenti o ex dipendenti che, volendo vendicarsi della propria azienda utilizzano i loro privilegi di amministratore o le loro conoscenze dei sistemi aziendali per compiere attacchi.
- **Cyber criminali:** sono criminali che spesso utilizzano i malware per creare un guadagno.
- **Stati-nazione:** stati che utilizzano dei malware con lo scopo di arrecare danni ad altri stati e alle loro strutture strategiche, oppure cercando di destabilizzarli.

Le modalità per installare i malware all'interno di una macchina di solito si basano quasi esclusivamente sull'ingegneria sociale. ....

### 1.1 Tipologie di malware

I malware possono essere suddivisi in diverse tipologie:

#### Virus

Oggi in disuso, il virus è un malware che ha la capacità di replicarsi sulla macchina su cui viene installato. Per essere eseguita necessita però dell'azione dell'utente che utilizza la macchina. Gli antivirus sono generalmente in grado di identificare i virus. Si possono distinguere in tre tipologie: le *macro* che arrivano sui computer delle vittime tramite un allegato, i *virus polimorfici* che sono caratterizzati dal fatto che hanno comportamenti diversi in base a quale sistema operativo possiede la vittima, ed i *companion* che si mascherano da programmi legittimi.

#### Worm

Sono simili ai virus ma non hanno bisogno di un'azione dell'utente per essere eseguiti. Inoltre hanno la capacità di diffondersi su vari dispositivi attraverso la rete. Il loro scopo è generalmente quello di installare delle backdoor.

#### Logic Bomb

Non molto diffuso, la logic bomb è progettata per compiere dei danni solo quando un determinato evento si verifica. Spesso utilizzato dagli Insider all'interno delle aziende.

## Trojans

Significa "cavallo di Troia" e si presenta come un software sicuro e legittimo quando in realtà viene progettato per compiere azioni malevoli quali operazioni bancarie fraudolente o creare delle backdoor. Ultimamente i trojans vengono utilizzati come vettore per scaricare altri malware sul computer della vittima.

## Rootkits

Sono una tipologia molto pericolosa poichè il loro scopo è dare accesso ai privilegi di amministratore all'attaccante. I rootkit si installano tra il sistema operativo e l'hardware fisico (ad esempio nel kernel), per questo sono difficili da individuare e qualora venissero individuati spesso non possono essere rimossi se non letteralmente distruggendo il disco. Vengono utilizzati dagli hacker per diversi scopi tra cui quello di assicurarsi di mantenere il loro controllo sulle macchine infette.

## Dropper/Downloader

I dropper e i downloader sono dei malware il cui scopo è quello di portare sulla macchina attaccata altre tipologie di malware scaricandoli dalla rete. I dropper includono il malware stesso all'interno dell'eseguibile come una risorsa, oppure sono contenuti in allegati malevoli.

## Key logger

Molto utilizzata dagli ingegneri sociali, è un malware che ha lo scopo di raccogliere tutti i caratteri digitati sulla tastiera. Tipicamente i caratteri registrati vengono salvati in un file di testo all'interno del file system e inviati all'attaccante tramite la rete.

## Ransomware

Malware il cui scopo è compromettere la disponibilità dei dati sulla macchina delle vittime. Ciò avviene con la cifratura dei dati personali presenti nelle cartelle e l'unico modo per decifrarli è pagare un riscatto (tipicamente in Bitcoin). Non è mai assicurato il rilascio della chiave di decifratura dopo aver pagato il riscatto, per questo è sempre consigliato di non pagare mai. Il malware più famoso è stato WannaCry.

## Bot

Infettano le macchine per fare in modo di creare una rete di computer infettati dal bot, le cosiddette *botnet*, e spingerle a fare attacchi di Denial of Service (DoS e DDos) e interrompere il servizio di target specifici. Rimangono in attesa del comando dell'attaccante per essere attivati. Esempi famosi sono Mirai, che colpiva i dispositivi IoT, e Satori.

## Cripto Miner

Tipologia di malware che infetta un dispositivo per installarci sopra un software che mina criptovalute al posto dell'attaccante. Oggi questi malware sembrano essere sempre meno diffusi.

## 1.2 Cyber Kill Chain

Articola le fasi di un attacco informatico. È importante per capire come è avvenuto l'attacco e quali vulnerabilità ha sfruttato. Si compone di 7 fasi.

1. **Reconnaissance:** obiettivo è quello di conoscere il target e raccogliere più informazioni possibili. Ci sono due tecniche principali:
  - Attiva: si raccolgono informazioni interagendo con il target. Alcuni strumenti utilizzati sono, ad esempio, *nmap*, portscanning, vulnerability scanners
  - Passiva: si raccolgono informazioni senza interagire direttamente con il target ma si utilizzano dei servizi esterni. Ad esempio lanciando *whois* o *nslookup*.

2. **Weaponization:** in questa fase si sceglie un modo per attaccare e si crea il mezzo. È in questo momento che si crea il malware, chiamato anche "payload". Alcuni strumenti di questa fase sono, ad esempio, Metasploit, Exploit DB o Social Engineering Toolkit.
3. **Delivery:** in questa fase si adotta un metodo per consegnare il payload malevolo sulla macchina della vittima. Ciò può avvenire ricorrendo alle tecniche di ingegneria sociale, quindi cercando di inviare il malware tramite email, link infetti, chiavette USB e molto altro.
4. **Exploitation:** fase di esecuzione del malware sulla macchina della vittima. L'esecuzione avviene sfruttando delle vulnerabilità del sistema utilizzato.
5. **Installation:** in questa fase l'obiettivo è quello di mantenere il controllo della macchina infetta, raggiungendo così persistenza. Alcune tecniche consistono nel sostituire i file DLL di Windows con dei file infetti oppure modificando alcune chiavi di registro.
6. **Command & Control (C2):** questa è la fase in cui l'attaccante comunica con il computer della vittima da remoto stabilendo un canale di comando e controllo.
7. **Actions on objectives:** fase dell'azione malevola progettata fin dall'inizio. L'attaccante potrebbe cifrare tutti i file del computer oppure raccogliere dati personali della vittima.

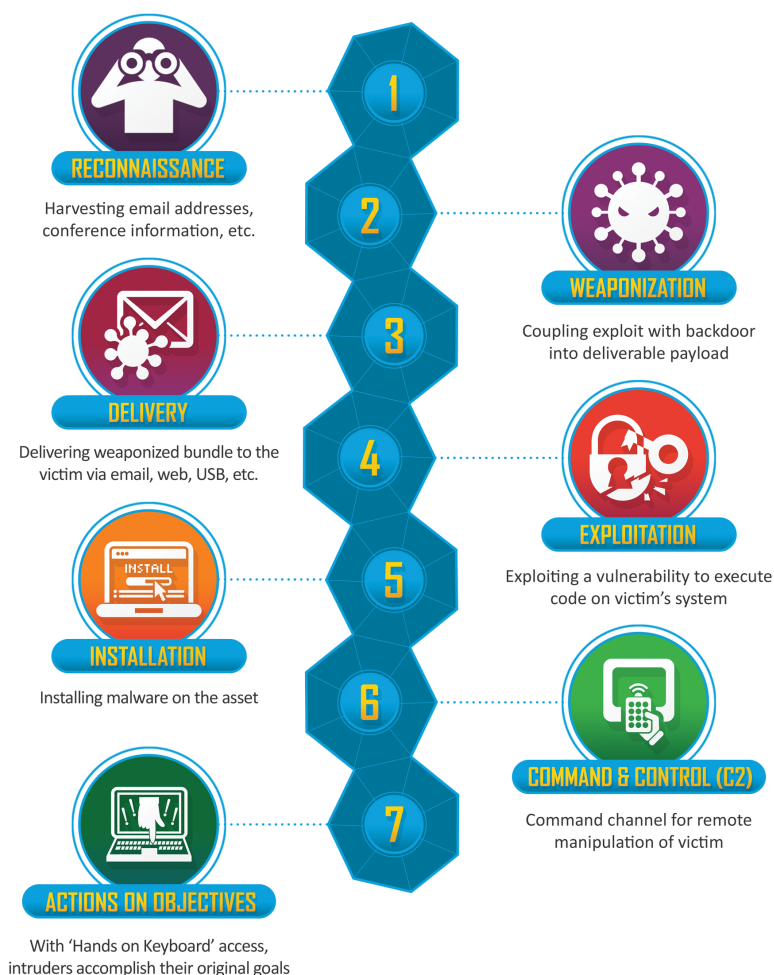


Figura 1.1: Schema che mostra i vari passi della Cyber Kill Chain.

## 1.3 Analisi dei malware

L'analisi di un malware è un processo che consiste principalmente in due fasi: l'*analisi statica* e l'*analisi dinamica*. Con l'analisi statica si cerca di analizzare il malware senza eseguirlo, mentre con l'analisi

dinamica lo si esegue per vederne il comportamento. Per capire cosa fa un malware spesso si ripete l'analisi più volte, sia statica che dinamica. I due tipi di analisi possono essere a loro volta suddivisi in *analisi di base* e *analisi avanzata*. Nelle analisi di base si va ad analizzare l'eseguibile senza addentrarsi nel codice in cui è stato scritto, mentre nelle analisi avanzate si eseguono operazioni come il reverse engineering o il debugging del codice malevolo. Per riepilogare i tipi di analisi da eseguire e che saranno illustrati nei prossimi capitoli sono:

1. Analisi statica di base
2. Analisi statica avanzata
3. Analisi dinamica di base
4. Analisi dinamica avanzata

## 1.4 Comportamenti dei malware

I malware sfruttano molte tecniche per cercare di mascherare il loro lavoro, non solo alla vittima ma anche ai malware analyst. Alcune di queste tecniche sono il cercare di capire se l'eseguibile sta girando su una macchina reale o su una macchina virtuale e per scoprirlo analizzano inanzitutto la quantità di risorse dedicate al sistema in termini di CPU, RAM e quantità di disco, inoltre analizzano i programmi installati sul dispositivo, per esempio, se sono presenti strumenti come Regshot oppure ProcessMonitor sanno che stanno girando sulla macchina di un malware analyst mentre se individuano programmi come Office oppure Adobe Reader sanno che la vittima è un normale utente. Tecniche più raffinate sono l'analizzare le date di creazione, di modifica e di accesso dei file, da cui si evince che, se sono troppo recenti, la macchina potrebbe essere stata appena creata con lo scopo di individuare e analizzare il malware in questione.

Molti malware prima di connettersi alla rete, controllano se la connessione è disponibile. Il malware analyst può tentare di far credere al malware che la connessione sia disponibile utilizzando strumenti come FakeNet e InetSim.

## 1.5 Preparazione dell'ambiente

Per eseguire l'analisi di un malware è importante preparare un ambiente in cui poterlo eseguire in modo sicuro, isolato e senza rischi per l'analizzatore. Esistono i cosiddetti malware lab con macchine dedicate e con una rete completamente isolata, ma questo approccio è utilizzato da aziende e grandi organizzazioni.

Altrimenti, è possibile utilizzare delle macchine virtuali che hanno il vantaggio di isolare sia il sistema operativo sia la rete. Con le macchine virtuali si possono inoltre riportare a istantanee precedenti ed eliminare gli effetti di un malware. La preparazione dell'ambiente virtuale procede come segue:

### Scelta del software di virtualizzazione

Scegliere il software che può fornirci la creazione di macchine virtuali. I software più famosi sono VMware, Hyper-V e VirtualBox. Il primo è un software commerciale e quindi a pagamento (ha una gestione migliore delle risorse per esperti), mentre gli altri due sono free. VirtualBox è quello che verrà utilizzato da qui in avanti all'interno di questo testo.

### Creazione di una nuova macchina virtuale

Per creare una macchina virtuale, in VirtualBox, basta cliccare su Nuova e impostare le caratteristiche che desideriamo, quali capacità del disco, memoria RAM dedicata, core del processore dedicati e stato della rete. In particolare è consigliato impostare come requisiti minimi le seguenti specifiche:

- Almeno 1 GB di memoria RAM;
- Almeno 2 core del processore;

Meglio se le specifiche assegnate si avvicinano il più possibile a quelle di una macchina reale, poiché esistono delle categorie di malware che, per rendere più difficile il lavoro del malware analyst, cercano di capire se si trovano su macchine reali o virtuali e cambiare il loro comportamento.

## Installazione del sistema operativo

Prima di procedere all'installazione del sistema operativo sulla macchina, bisogna scaricare l'immagine ISO per l'installazione. Se si tratta di distribuzioni Linux o comunque sistemi open source, le immagini sono scaricabili gratuitamente, mentre per altri sistemi proprietari come Windows (il sistema che si userà in questo testo) è possibile reperire delle versioni per sviluppatore attive per un tempo limitato. Microsoft dà la possibilità di scaricare una macchina virtuale completa all'indirizzo: <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms>.

Si può decidere se installare gli aggiornamenti o meno del sistema operativo; questo per fare in modo di avere una particolare vulnerabilità risolta nel tempo da un particolare aggiornamento.

Su Windows bisogna disabilitare Windows Defender, sia in real time protection che in cloud protection. Questo perchè anche i tool utilizzati possono essere visti come malevoli.

## Installazione dei software per l'analisi dei malware

L'installazione degli strumenti per l'analisi dei malware può avvenire manualmente, installando i software desiderati. ecc.... su github script python che installa tutti i tool.

## Configurazione della rete

Configurare l'opzione di rete Host-Only (scheda solo host), per impedire al malware di diffondersi attraverso la rete e infettare dispositivi al di fuori della macchina virtuale. Con quest'opzione la rete sarà confinata solamente all'interno della macchina.

## Creazione di un istantanea

Prima di iniziare la prima analisi, è necessario catturare un'istantanea della macchina per poter ripristinare la macchina ad ogni nuova analisi.

## 1.6 Reperire i malware

Per esercitarsi è possibile scaricare alcuni esempi di malware da diversi siti:

- Hybrid Analysis
- ANY.RUN
- VirusBay
- VirusShare
- VirusSign
- <https://zeltser.com/malware-sample-sources/>



## Capitolo 2

# Analisi statica di base

Nell'analisi statica di base il file malevolo viene analizzato senza essere eseguito. Si differenzia dall'analisi statica avanzata per il fatto che non si vanno a visualizzare le istruzioni del codice malevolo ma si visualizza solo la sua signature. In questo tipo di analisi si vanno, ad esempio, a verificare le stringhe presenti nel file PE. Viene utilizzata per capire la tipologia di file che stiamo analizzando, anche per evitare che alcuni malware si nascondano sotto altre tipologie di file.

Con l'analisi statica di base si va a vedere se un file è effettivamente un malware, se è impacchettato oppure no, quali librerie importa, quali servizi di sistema utilizza e se modifica o no chiavi di registro.

Il tipico processo svolto durante l'analisi statica di base è l'analisi delle stringhe.

### 2.1 Identificazione del file

Determinare l'effettiva estensione del file è una delle parti principali di questa analisi. Gli hacker tentano infatti di dare una doppia estensione ai file per cercare di mascherare la vera estensione .exe facendo credere che siano ad esempio dei file PDF, documenti di testo o qualsiasi altra cosa appaia innocua.

Alcune volte gli attaccanti mascherano i malware da archivi autoestraenti: con un programma apposito (ad esempio con WinRAR) creano un archivio che contiene il malware e scrivono uno script da eseguire all'apertura dell'archivio, in questo modo quando l'utente aprirà il file che sembrerà in tutto e per tutto un archivio compresso, lo script tirerà fuori il malware e lo eseguirà.

L'identificazione di un eseguibile di Windows avviene attraverso l'analisi della signature. In particolare un eseguibile di Windows avrà sempre i primi due bytes dedicati alle lettere MZ in codifica ASCII, mentre un PDF inizierà sempre con %PDF-. Un archivio Zip inizia con i bytes dedicati alle lettere PK. Questa caratteristica appartiene al DOS Header di ogni file PE, in particolare si tratta del campo e\_magic. Conoscendo questi particolari un programma, come quelli che saranno descritti in questo capitolo, possono facilmente identificare la tipologia di file.

#### File

da scrivere.

#### ExeInfo PE

da scrivere.

#### PEiD

PEiD è uno strumento simile a ExeInfo PE e da informazioni su un file PE analizzato.

Per prima cosa, aprire PEiD come amministratore e cliccare sul tasto "..." (tre puntini) per scegliere il file. Una volta scelto il file, la finestra principale di PEiD ci mostra alcune informazioni come...

La figura 2.1 mostra l'analisi di un keylogger.

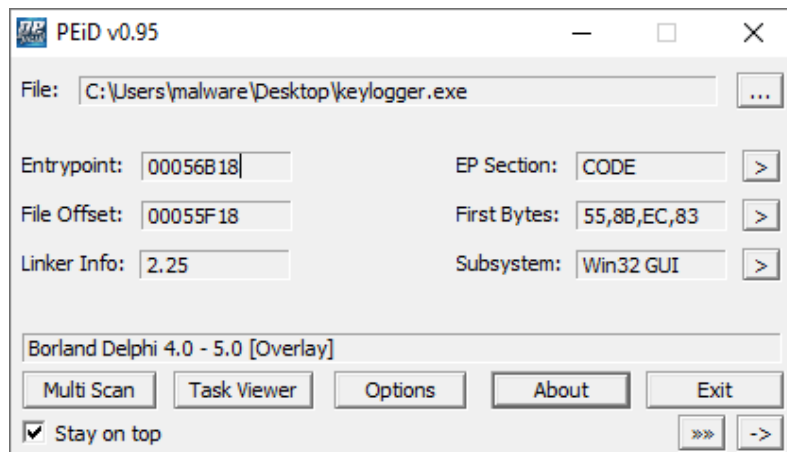


Figura 2.1: Schermata iniziale di PEiD.

### Plugin: Krypto ANALyzer

Se si dispone del plugin Krypto ANALyzer (o KANAL) è possibile ottenere in una nuova finestra informazioni riguardanti vari algoritmi di codifica e cifratura utilizzati all'interno del malware. Un esempio è mostrato dalla figura 2.2 che ci suggerisce l'utilizzo di ... per il keylogger analizzato sopra.

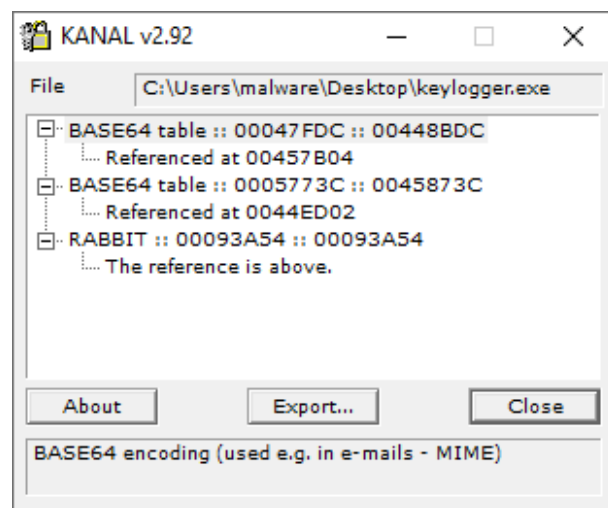


Figura 2.2: Plugin KANAL di PEiD.

### TriD

da scrivere. Utilizzando i pattern e non le signature non dice esattamente qual è la tipologia di file ma dà una probabilità sulla natura del file.

## 2.2 Calcolo dell'hash

È possibile generare un hash del malware utilizzando diversi algoritmi come MD5, SHA-1, SHA-256, ecc.... Avere l'hash di un file malevolo è utile per farlo analizzare da servizi che cercano di individuarlo nei loro database. Il più famoso è VirusTotal: un servizio online che offre la possibilità di far analizzare un malware dai più conosciuti antivirus in commercio e fornisce i risultati delle analisi e le statistiche. Fornendo a VirusTotal l'hash del codice malevolo è in grado di dire se lo stesso file è già stato individuato in passato come sospetto. VirusTotal è disponibile all'indirizzo: <https://virustotal.com>.

...immagine virustotal

Alcuni strumenti utili per il calcolo dell'hash di un generico file sono descritti di seguito.

HashMyFiles

HashCalc

ComputeHash

## 2.3 Librerie e funzioni importate

Durante l'analisi statica di base, è importante prestare molta attenzione alle librerie Windows, o DLL, importate dal malware. Le librerie più comunemente importate in un eseguibile Windows sono riassunte nella tabella sottostante:

Nome della libreria	Utilizzo da parte del sistema operativo	Utilizzo da parte dei malware
Kernel32.dll	utilizzata da tutti i programmi Windows	Permette al malware di interagire con il file system e la memoria RAM.
Advapi32.dll	manipolare i registri	Utilizzata dai malware per raggiungere persistenza.
User32.dll	gestire l'interfaccia utente	
Gdi32.dll		
Ntdll.dll		
WSock32.dll e Ws2_32.dll	crea connessioni di rete	
Wininet.dll	implementare protocolli di rete HTTP/Https	

Le funzioni più comunemente utilizzate dai malware dipendono dalle operazioni da eseguire.

### Operazioni sulla memoria

VirtualAlloc

VirtualProtect

VirtualFree

### Operazioni sui file

CreateFile

ReadFile

WriteFile

DeleteFile

### Operazioni sui registri

RegCreateKey

RegDeleteKey

RegGetValue

RegSetValueEx

RegSetKeyValue

### Operazioni di rete

connect, accept, send, recv, listen (utilizzata nei malware vettori), gethostbyaddr, gethostbyname, InternetConnect, InternetReadFile, InternetWriteFile

## Altre operazioni

LoadLibrary GetProcAddress IsDebuggerPresent WriteProcessMemory CreateRemoteThread

altre: shellExecuteA significa che il malware lancia comandi itoa: converte indirizzi ip in stringa (??)

## 2.4 Analisi delle stringhe

L'analisi delle stringhe consiste nel trovare le possibili stringhe presenti nell'eseguibile compilato e cercare di dare loro dei significati per comprendere se ci troviamo davanti ad un malware e quali sono le sue funzionalità.

Le stringhe che si vanno a cercare sono generalmente di due tipologie in base alla codifica: stringhe *ASCII* o stringhe *Unicode*. Nel caso delle stringhe *ASCII* ogni carattere della stringa viene codificato con un byte mentre nel caso dell'*Unicode* ogni carattere ha riservato 2 byte. In entrambe le codifiche ogni stringa viene terminata con il carattere nullo, rappresentato dallo 0.

esempio figura codifica

Gli hacker possono cercare di rendere difficile l'analisi delle stringhe tentando di impacchettare l'eseguibile. L'impacchettamento sarà descritto più avanti.

### Stringhe di rete

All'interno del malware possono trovarsi stringhe che in un certo qual modo indicano un'attività di rete da parte del malware. Queste stringhe possono essere indirizzi IP, nomi di host o indirizzi URL. Qui di seguito sono indicati alcuni esempi di stringhe di rete:

...

### Nomi di file

Si possono trovare stringhe di nomi di file che possono riguardare file creati dal malware stesso oppure file che il malware va a ricercare all'interno del file system. Molto spesso si trovano stringhe con il nome stesso del malware e ciò indica che il malware tenta di creare copie di se stesso.

### Stringhe di librerie

Si può riconoscere una chiamata a librerie se sono presenti stringhe come: WriteFile, SetRegValue, ecc....

### Stringhe di codifica

La presenza di stringhe del tipo: "inserire stringa" indica l'utilizzo di codifica in Base64. La codifica in base64 infatti utilizza solo un determinato insieme di caratteri che comprende i numeri da 0 a 9, i caratteri da A a Z sia maiuscoli che minuscoli, il simbolo + e /. Una particolarità di questa codifica è che trasforma ogni tre byte in quattro e se la lunghezza finale non è divisibile per quattro si aggiungono i simboli di uguale =.

Per capire quindi se la codifica in base64 è stata utilizzata, dovremmo trovare la stringa A—z0—9+// oppure stringhe che finiscono con uno o più simboli di uguale.

### XORSearch

### Chiavi di registro

Chiavi di registro come stringhe indicano che il malware accede, modifica, crea o cancella chiavi di registro. Ciò viene fatto di solito quando l'eseguibile malevolo vuole raggiungere una certa persistenza sulla macchina.

### Altre stringhe

Altre stringhe che si possono trovare sono stringhe che indicano funzionalità del malware come:

...

## Strings

## 2.5 Data di compilazione

Coloro che scrivono malware tendono spesso a nascondere la vera data di compilazione che può risultare molto indietro nel tempo oppure addirittura nel futuro. È molto utile associare la data di compilazione con altre date all'interno dell'eseguibile come i timestamp associati alle risorse: una grossa differenza in termini di anni può indicare una manomissione della data di compilazione e far crescere il sospetto che il file analizzato sia un file malevolo.

## 2.6 Strumenti per l'analisi statica di base

L'intera analisi statica di base si basa essenzialmente sull'analisi dei cosiddetti file PE di Windows. Il PE file (si veda l'Appendice A per saperne di più sui PE file) ci può dare molte informazioni sulle librerie importate, lo spazio di memoria allocato per l'eseguibile e molto altro.

Prima di analizzare come è composto un PE file è importante capire che l'importazione delle librerie utilizzate da un eseguibile può avvenire in tre modi:

- **Statica:** tipica dei sistemi Unix, quando si compila un file, le librerie richiamate dal file stesso vengono incluse nell'eseguibile e non è quindi necessario risolvere nessun riferimento.
- **Dinamica:** le librerie utilizzate dal file vengono caricate in memoria nel momento in cui l'eseguibile viene caricato in memoria. I nomi e i riferimenti delle librerie da caricare vengono forniti dal campo Import Address Table.
- **Runtime:** viene utilizzata principalmente da chi scrive malware. Le librerie vengono caricate nello spazio di memoria allocato dall'esecuzione del malware nel momento in cui vengono chiamate dal malware stesso. Le librerie **LoadLibrary** e **GetProcAddress** vengono spesso utilizzate per caricare altre librerie, ed è un forte indicatore del fatto che il malware è stato impacchettato.

Ogni file PE è composto da diversi header tra cui DOS Header, PE Header, File Header e Optional Header che sono essenzialmente delle strutture con all'interno diversi campi utili a identificare meglio l'eseguibile analizzato. In particolare, all'interno dell'Optional Header, il campo Magic ci dice se il file è un eseguibile per architetture a 32 bit o a 64 bit, il campo AddressOfEntryPoint indica il puntatore da cui iniziare ad eseguire il codice. I malware possono però effettuare altri controlli prima di questo puntatore quindi non sempre è realmente indicativo dell'inizio delle istruzioni eseguibili. Un altro campo DataDirectory, che contiene il riferimento a diverse tabelle tra cui:

- **Tabella Export Directory:** importante se il file eseguibile è una libreria, cioè un DLL. Le funzioni in questa tabella vengono rese disponibili esternamente a chi esegue la libreria.
- **Tabella Import Directory:** specifica tutte le librerie importate dal malware. Le librerie da importare sono specificate nel Import Address Table e l'importazione avviene con riferimento dinamico.
- **Tabella Resource Directory:** contiene le risorse utilizzate dal file, come le icone.
- **Import Address Table:** specifica tutte le librerie che il malware deve importare in memoria. Si tratta di una lista di funzioni per ogni file DLL specificato. Ci possono essere due modi per caricare le librerie contenute nella tabella: il primo consiste da parte del loader nel sostituire il nome contenuto nella tabella con l'effettivo indirizzo della libreria per indicare al malware dove recuperarle; il secondo metodo consiste nel utilizzare un valore ordinale delle funzioni all'interno di una DLL al posto del nome delle funzioni. figura...

Un indicatore che ci suggerisce che ci troviamo di fronte ad un malware impacchettato è la presenza di sezioni con nomi diversi da quelli tipici di un PE file e con permessi sia di esecuzione che di scrittura.

Per analizzare un file PE, e completare tutte i tipi di analisi descritti in questo capitolo, si possono utilizzare gli strumenti descritti qui di seguito.

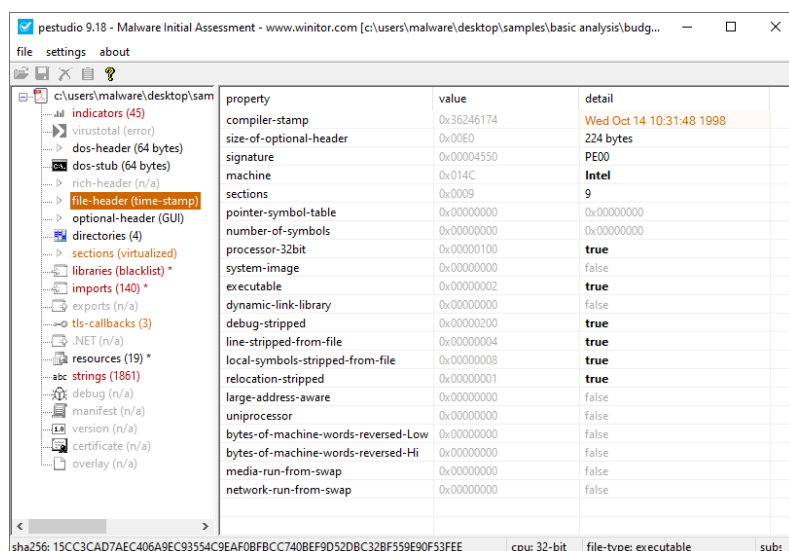
I malware utilizzano le risorse per immagazzinare codice malevolo o file di configurazione. I trojan, che si mascherano da programmi legittimi, nascondono ad esempio dei file PDF malevoli all'interno delle risorse che verranno eseguiti una volta lanciato il trojan.

## PEStudio

È un software pensato appositamente per fare malware analysis, infatti PEStudio tenta già di dare delle indicazioni sui campi analizzati per dire se sono malevoli o no. È possibile scaricare una versione PEStudio dal sito "<https://www.winitor.com/features>" dove sono disponibili sia una versione gratuita che a pagamento.

Innanzitutto avviare PEStudio come amministratore, dopodiché importare in PEStudio il file che si vuole analizzare trascinandolo sulla finestra del programma oppure cliccando su **file > open file** e selezionando dalle cartelle il file da analizzare.

Possiamo guardare il contenuto del file header cliccando sulla sezione a sinistra **file header**. Guardando l'esempio in figura 2.3 notiamo informazioni come la data di compilazione in cui una data di compilazione molto vecchia (in questo caso 2014) potrebbe essere un tentativo di depistaggio da parte dell'attaccante per nascondere il malware. Altre informazioni utili sono **machine** che indica l'architettura del file (in questo caso x86), il numero di sezioni viene dato da **section**. Tutte le restanti informazioni riguardano il campo delle caratteristiche.



property	value	detail
compiler-stamp	0x36246174	Wed Oct 14 10:31:48 1998
size-of-optional-header	0x00E0	224 bytes
signature	0x00004550	PE00
machine	0x014C	Intel
sections	0x0009	9
pointer-symbol-table	0x00000000	0x00000000
number-of-symbols	0x00000000	0x00000000
processor-32bit	0x00000100	true
system-image	0x00000000	false
executable	0x00000002	true
dynamic-link-library	0x00000000	false
debug-stripped	0x00000200	true
line-stripped-from-file	0x00000004	true
local-symbols-stripped-from-file	0x00000008	true
relocation-stripped	0x00000001	true
large-address-aware	0x00000000	false
uniprocessor	0x00000000	false
bytes-of-machine-words-reversed-Low	0x00000000	false
bytes-of-machine-words-reversed-Hi	0x00000000	false
media-run-from-swap	0x00000000	false
network-run-from-swap	0x00000000	false

sha256: 15C3CAD7AEC406A9EC93554C9EAF0BFBC740BEF9D52DBC32BF559E90F53FEE    cpu: 32-bit    file-type: executable    sub:

Figura 2.3: Schermata che mostra le informazioni estrapolate dal File Header del file caricato in PEStudio.

Cliccando invece su **optional header** si possono vedere le informazioni riguardanti l'Optional Header del file PE. La figura 2.4 mostra il campo magic che contiene PE, l'entry point che ci indica il punto della memoria dove iniziare ad eseguire il codice (in questo caso section:.text indica che il codice parte dalla sezione .text), la dimensione del codice è indicata dal campo size-of-code e l'image base, indicato da image-base indica il punto di caricamento del PE file per il Loader.

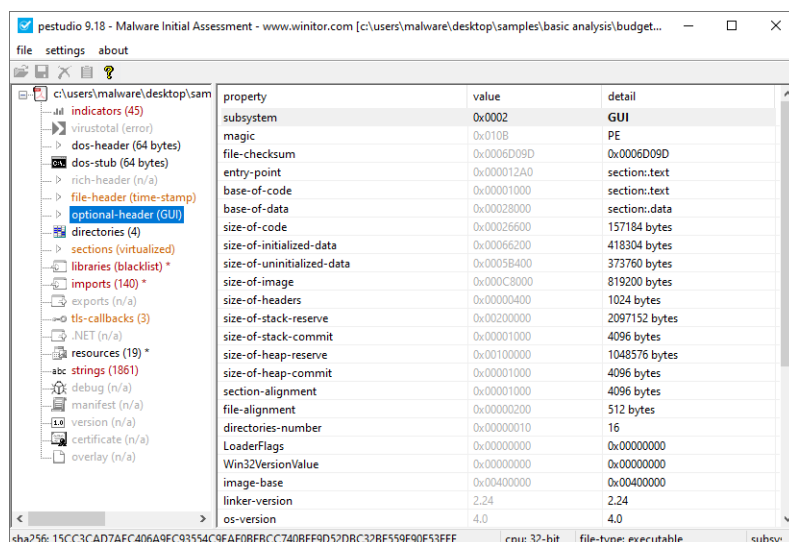


Figura 2.4: Schermata che mostra le informazioni contenute nell'Optional Header del file PE caricato in PESTudio.

Una funzionalità importante è la visualizzazione delle sezioni che è possibile cliccando su **sections** alla sinistra del programma come mostrato dalla figura 2.5. Qui è importante guardare se sono presenti sezioni con nomi inusuali e che hanno permessi sia di esecuzione che di scrittura. Il campo name ci indica il nome della sezione e i campi executable, readable e writeable i vari permessi.

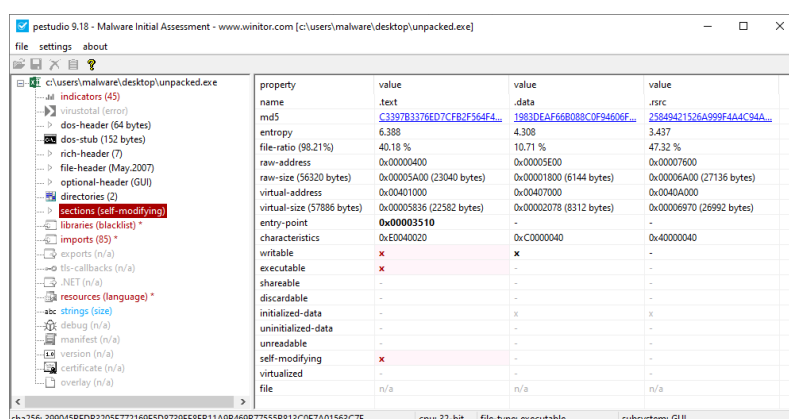


Figura 2.5: Informazioni riguardanti le sezioni di un file PE in PESTudio.

La sezione **libraries** ci mostra le librerie importate dal PE file malevolo. Nel malware analizzato in figura 2.6 notiamo che il file PE importa librerie come wininet.dll e ws2\_32.dll che indicano un'attività di rete da parte del malware e sono state riconosciute da PESTudio come sospette e quindi segnate da una X in rosso.

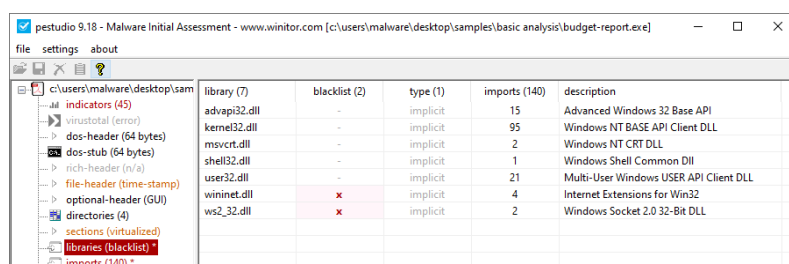


Figura 2.6: Analisi delle librerie contenute in un file PE attraverso PESTudio

La sezione **imports** mostra tutte le funzioni importate dal PE file. Anche in questo caso PESTudio ci segna quelle potenzialmente sospette. Dalla figura 2.7 possiamo notare comunque la presenza di librerie che possono indicare un'azione malevola da parte dell'eseguibile, in particolare: ... elenco librerie

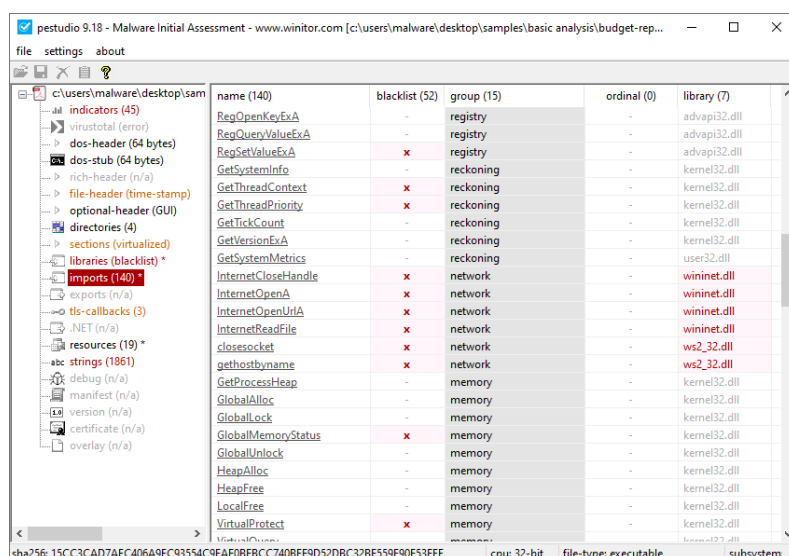


Figura 2.7: Analisi delle librerie importate all'interno di un file PE attraverso PESTudio.

Nel campo risorse PE Studio ci fornisce la tipologia del contenuto della risorsa, la signature, la lingua che di solito viene specificata dal programmatore del malware oppure viene impostata in automatico dal compilatore del malware. Nelle risorse di un file PE, un attaccante può includere una risorsa di tipo eseguibile che spesso può essere il vero malware nascosto come in figura 2.8.

type (1)	name	location (1)	signature (1)	size (16384 bytes)	file-ratio (44.44%)	hash (md5)	entropy	language (1)	first-bytes-hex
BIN	101	0x00004060	executable (cpu: 64-bit)	16384	44.44 %	6A95C2F88E0...	0.852	English-US	4D 5A 90 00 0...

Figura 2.8: Esempio di risorsa malevola contenuta in un file PE analizzata con PESTudio. In questo caso si può notare che come risorsa è stato inserito un eseguibile.

Molto importante è la sezione **strings** che ci mostra le stringhe trovate dall'analisi del file PE. La figura 2.9 mostra le come vengono rappresentate le stringhe e come PESTudio suggerisce già i possibili utilizzi delle stringhe. Analizzando con PESTudio un esempio di keylogger potremmo trovare all'interno le seguenti stringhe:

... elenco stringhe ... descrizione delle stringhe.



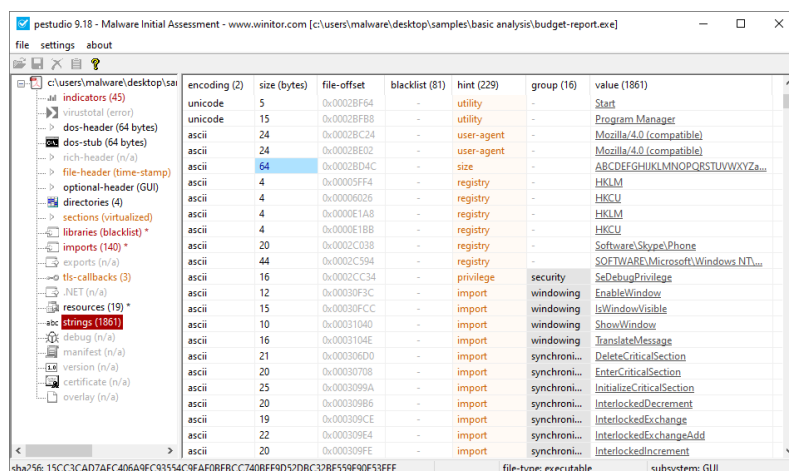


Figura 2.9: Elenco delle stringhe trovate all'interno di un file PE con l'analisi delle stringhe in PESTudio.

La funzione Sleep è spesso un segnale d'allarme poichè viene utilizzata dai malware per aggirare software antivirus o anti-malware.

Altre sezioni meno importanti per l'analisi sono **directories**, che indica le data directories.

Un problema che può sorgere nell'analisi di un malware è il fatto che il malware può essere impacchettato. Per analizzare un malware impacchettato bisogna prima spaccettarlo. Per fare ciò si rimanda al capitolo sull'offuscamento.

## PEView

Un altro software per l'analisi di un file PE è PEView. A differenza di PESTudio ...

Per analizzare un file con PEView la prima cosa da fare è avviare il programma con i privilegi di amministratore e verrà subito chiesto di selezionare dalle cartelle il file malevolo da visualizzare. Una volta fatto ciò PEView mostra i dati contenuti nel file PE come in figura 2.10.

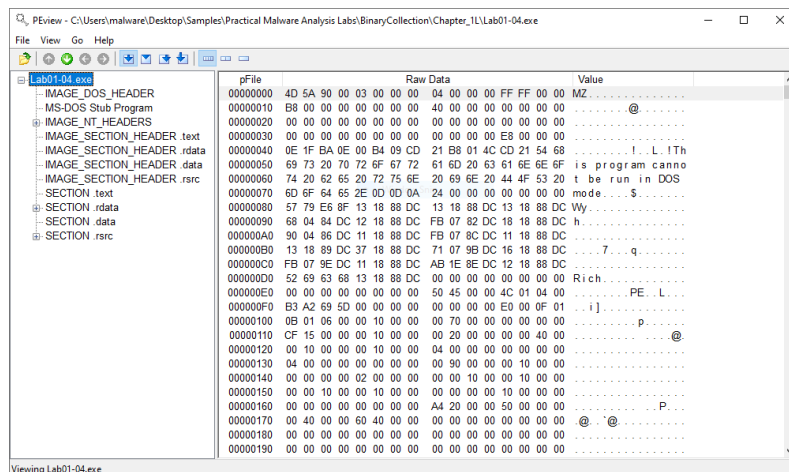


Figura 2.10: Schermata iniziale mostrata da PEView al caricamento di un file.

Nella parte sinistra della finestra di PEView è possibile vedere anche il contenuto delle sezioni dell'eseguibile e dei vari header. Spesso all'interno dei malware viene inserito del codice eseguibile all'interno della sezione dedicata alle risorse; con PEView è possibile identificare eventuali risorse eseguibili. Nella figura 2.11 è possibile vedere il contenuto dell'import address table.

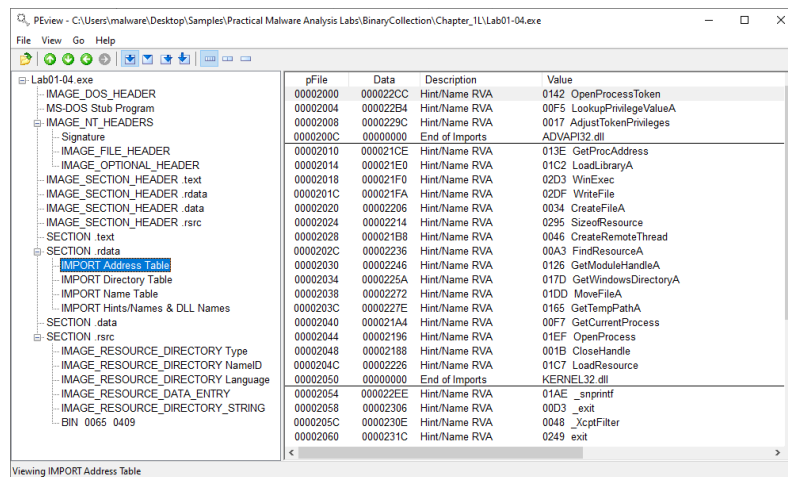


Figura 2.11: Sezione di un file PE contenente la Import Address Table visualizzata con PVIEW.

## CFF Explorer

Consente anche di modificare eventualmente i campi del PE file.

## Capitolo 3

# Analisi dinamica di base

L'analisi dinamica è quel tipo di analisi in cui si esegue il malware e si valutano le modifiche effettuate sul sistema operativo. L'esecuzione di un malware può portare a modifiche del sistema come la creazione, l'eliminazione o la modifica dei registri di Windows, può generare traffico di rete nel momento in cui si collega a server remoti e scarica altri file sul sistema, può lanciare ulteriori processi e fare in modo di avviarsi in automatico oppure modificare a proprio piacimento il file system della vittima.

Per comprendere meglio questo tipo di analisi sui sistemi Windows si consiglia la lettura dell'appendice sul sistema Windows, in particolare sulle Windows API e sul registro di sistema.

### 3.1 Modifica del file system

I malware agiscono sul file system creando, eliminando o modificando file. Le funzioni interessate sono `CreateFile`, `ReadFile` e `WriteFile`. Se, tramite analisi statica, notiamo che un malware importa queste funzioni significa che ha capacità di creare, leggere e scrivere file.

Altre funzioni potenzialmente utilizzate da un malware sono `CreateFileMapping` e `MapViewOfFile`. Vengono usate per emulare il comportamento del Windows Loader, cioè il processo che si occupa di caricare un file in memoria, per caricare in autonomia dell'altro codice malevolo in memoria.

### 3.2 Modifica dei registri

Tipicamente l'accesso ai registri non prevede l'utilizzo dei privilegi di amministratore. I malware utilizzano particolari chiavi per raggiungere determinati scopi.

Le funzioni tipicamente utilizzate per modificare i registri sono: `RegOpenKeyEx`, `RegSetValueEx` e `RegGetValue`. Un malware che importa queste funzioni ha la capacità di leggere e impostare nuove chiavi di registro e valori associati alle chiavi.

#### Creare persistenza

La cartella di registro `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` stabilisce quali programmi vengono lanciati nel momento in cui un utente effettua l'accesso alla macchina. Un malware può modificare questa cartella e aggiungere una chiave con il proprio nome e un valore con il percorso del file da eseguire all'avvio per assicurarsi di essere eseguito ogni volta che l'utente accede al proprio account. In questo modo si dice che il malware ha *raggiunto persistenza*.

Altre cartelle di registro simili utilizzate per creare persistenza sono:

- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices`
- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce`
- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run`
- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Windows\AppInit_DLLs`

... spiegazione + controllare che HKLM sia giusto.

## Autoruns

Autoruns è un semplice software interno di Windows che mostra tutti i possibili modi in cui un malware può raggiungere persistenza sulla macchina. Questo strumento si occupa di scansionare tutti i registri e tutto ciò che un malware può cambiare per auto avviarsi e lo mostra all'utente per ogni cartella di registro citata precedentemente.

Autoruns cerca di evidenziare in rosso i programmi che non hanno una firma valida, ma bisogna fare attenzione perchè possono essere evidenziati anche programmi legittimi che però non sono verificati.

## 3.3 Attività di rete del malware

Un malware, durante la sua esecuzione, può generare traffico di rete e connettersi a server remoti oppure fare esso stesso da server e accettare connessioni in entrata. Può quindi avere un ruolo da client o da server ed è importante capire il suo funzionamento, e per farlo si può analizzare l'ordine di chiamata delle funzioni della libreria di Windows `Ws2_32.dll`.

Nel caso in cui il malware si comporti come un **server** il suo scopo sarà quello di mantenere attiva una socket in ascolto per le connessioni in arrivo da remoto, quindi, a livello di programmazione le funzioni saranno chiamate nell'ordine:

1. `socket`
2. `bind`
3. `listen`
4. `accept`
5. `send` e `recv`

Inanzitutto il malware chiama la funzione `WSAStartup()`, dopodichè crea una socket (funzione `socket`) e la collega ad una porta con `bind` e la mette in ascolto delle connessioni (funzione `listen`). All'arrivo di una richiesta di connessione remota, utilizza la funzione `accept` per accettarla e successivamente inviare oppure ricevere dati (`send` e `recv`).

Nel caso invece in cui il malware si comporti come un **client** il suo scopo sarà quello di connettersi ad un socket remoto ed inviare dati personali della vittima oppure scaricare dati come un ulteriore codice malevolo da eseguire. In questo caso l'ordine di chiamate delle funzioni della libreria `Ws2_32.dll` è il seguente:

1. `socket`
2. `connect`
3. `send` e `recv`

Per prima cosa il malware utilizza la funzione `socket` per creare una socket e successivamente la funzione `connect` per connettersi, tramite la socket appena creata, al server remoto. Una volta stabilita la connessione invia e riceve dati a proprio piacimento (funzioni `send` e `recv`).

Analizzare il traffico con strumenti di sniffing è una tecnica di analisi molto importante risoluzione di nomi dns inusuali vengono usati per scaricare dati dai malware. Utilizzo del protocollo HTTP GET request utilizzate in due modi: scaricare malware sulla macchina della vittima, raccogliere informazioni utili per contattare il server Command and Control. Le richieste POST sono utilizzate per salvare dati copiati sulla macchina della vittima (es. password). Dalle richieste POST è possibile recuperare i dati del pacchetto. Anche il protocollo FTP può essere utilizzato. Ultimamente vengono usati anche protocolli per Samba o protocolli per il servizio email.

## Wireshark

Wireshark è uno dei più conosciuti software per effettuare lo sniffing di pacchetti di rete ed analizzare così il traffico di una rete. In questo contesto è molto utile per capire quali operazioni un malware compie attraverso internet e come cerca di comunicare con altri dispositivi in rete.

All'apertura di Wireshark come amministratore viene chiesto di scegliere quale interfaccia di rete monitorare; una volta scelta si vedrà subito attiva la cattura di tutti i pacchetti che viaggiano in rete in una schermata simile a quella in figura 3.1.

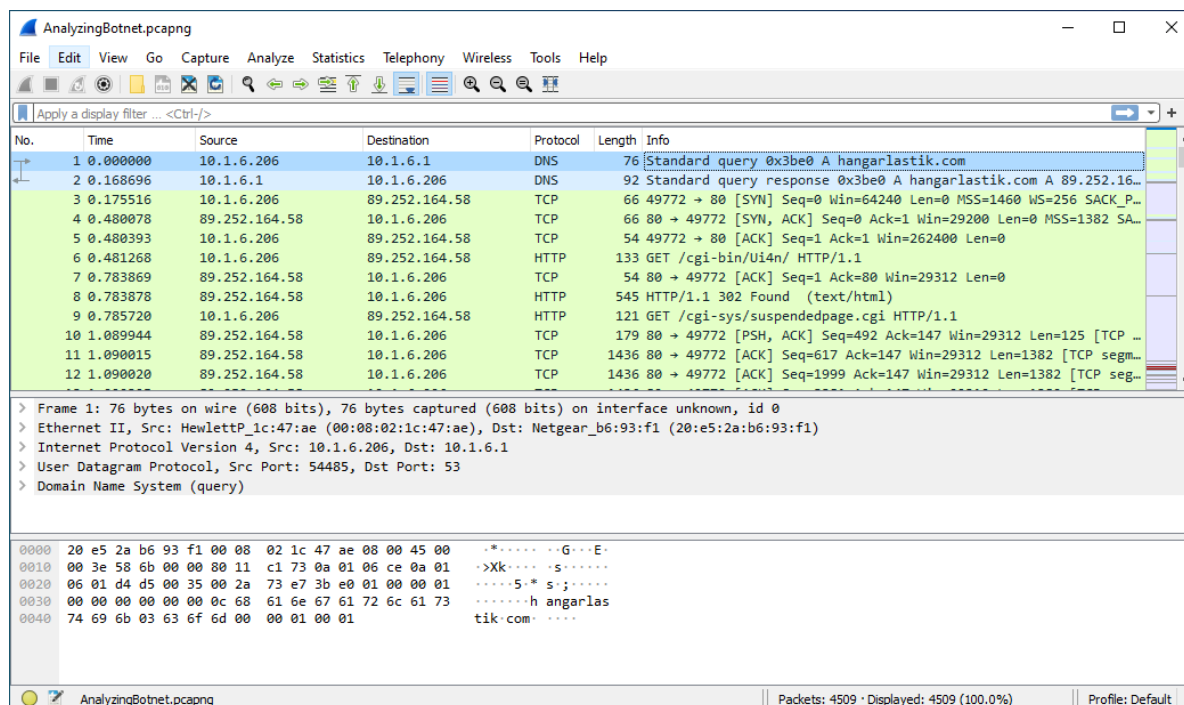


Figura 3.1: Schermata principale di Wireshark e cattura di alcuni pacchetti che viaggiano attraverso la rete.

La finestra principale di Wireshark è suddivisa in tre schede: *Packet List*, *Packet Details* e *PacketBytes*.

La prima scheda in alto (*PacketList*) mostra essenzialmente la lista dei pacchetti catturati e alcuni campi con le informazioni essenziali. Questi campi sono:

- **No.:** indica il numero in ordine del pacchetto;
- **Time:** tempo espresso in millisecondi che indica l'arrivo di un pacchetto;
- **Source:** l'indirizzo IP sorgente del pacchetto;
- **Destination:** l'indirizzo IP di destinazione del pacchetto;
- **Protocol:** il protocollo dello stack TCP/IP a cui appartiene il pacchetto;
- **Length:** la lunghezza in byte del pacchetto;
- **Info:** informazioni sommarie sul pacchetto selezionato;

È inoltre possibile aggiungere ulteriori campi come le porte sorgente e destinazione di un pacchetto. Per farlo basta cliccare con il tasto destro su una colonna a caso e cliccare su **Column Preferences...**, si aprirà una finestra popup dove si può aggiungere una nuova colonna con il tasto + nella sezione **Appearance** > **Columns**. Se si vogliono aggiungere i campi per la porta sorgente e destinazione nei valori delle nuove colonne si deve impostare rispettivamente *Src port (unresolved)* e *Dest port (unresolved)*.

La seconda scheda nel mezzo è la scheda *Packet Details* e mostra proprio i dettagli degli header per ogni pacchetto che si seleziona. Gli header vengono suddivisi per livelli dello stack TCP/IP e ogni livello mostra i valori di ogni campo. Si possono creare nuove colonne nella lista dei pacchetti della scheda precedente con il valore di un particolare campo cliccando con il tasto destro sul campo e cliccando **Apply as Column**.

L'ultima scheda, *PacketBytes*, che si trova nella parte inferiore in figura 3.1 mostra sulla sinistra il valore in byte del contenuto dell'intero pacchetto e ne fornisce, sulla destra, una possibile decodifica in un linguaggio umano. Selezionando con il mouse i campi degli header nella scheda *Packet Details* è possibile vedere in modo automatico la corrispondente selezione in byte di quei campi.

## Impostare dei filtri

Su Wireshark è possibile impostare dei filtri per mostrare solo i pacchetti di reale interesse senza tenere conto di tutti gli altri pacchetti del traffico di rete.

```
http.request or ssl.handshake.type == 1
```

serve a mostrare solo i pacchetti di tipo HTTP o SSL/TLS.

Se si vogliono filtrare pacchetti per indirizzi IP sorgente e/o destinazione si possono scrivere filtri come

```
ip.src == 192.168.1.1  
ip.dst == 151.183.38.3  
ip.addr == 192.168.1.2
```

Se invece si vogliono filtrare solamente pacchetti HTTP (sia richieste che risposte) si può scrivere il filtro

```
http.request or http.response
```

Nel caso in cui si voglia filtrare solo uno o più pacchetti che contengono una determinata stringa (può essere ad esempio il nome o il percorso di una risorsa di un pacchetto HTTP GET) si può inserire filtri del tipo

```
frame contains stringa
```

e si otterranno tutti i pacchetti in cui c'è la stringa *"stringa"*.

## Salvare una configurazione

Per salvare una configurazione andare su **Edit > Configuration Profiles...** e una volta aperta la finestra di configurazione cliccare sul tasto + e assegnare un nuovo nome.

## Seguire un flusso TCP

Su Wireshark possiamo selezionare un pacchetto TCP o di livello applicativo e con il tasto destro cliccare **Follow > TCP Stream**. Questa funzione è molto utile poichè permette di mostrare in una sola finestra l'intero scambio di pacchetti in quella connessione TCP, riuscendo anche a mostrare i dati inviati tramite eventuali richieste POST o GET. (è giusto?)

## 3.4 Esecuzione di ulteriore codice

Eseguendo un malware è interessante analizzare i metodi in cui un malware esegue ulteriore codice generato dal malware stesso, oppure scaricato da un server malevolo. Ci sono principalmente quattro modi: tramite *DLL*, con *processi*, utilizzando *thread* o utilizzando *servizi*.

Le DLL sono utilizzate da un malware per caricare codice malevolo contenuto all'interno del DLL. I malware possono anche dirottare il caricamento di DLL fruttando il loro ordine di ricerca da parte dell'eseguibile che deve caricarne le librerie. Si può immaginare uno scenario in cui un programma eseguibile, tale **programma.exe**, ha bisogno dei riferimenti alle librerie utilizzate all'interno del proprio codice ed inizia a cercare il proprio DLL, detto **esempio.dll**, per prima cosa all'interno della cartella dove è contenuto il programma. All'interno della cartella però non è presente il file ed allora lo si cerca in `windows/system32` ma non è presente nemmeno lì, allora si va a cercarla in `windows`, poi nella cartella corrente ed infine, se non è presente nemmeno nella cartella corrente si va in `%PATH%`. Un malware che vuole cercare di far eseguire a **programma.exe** una copia malevola di **esempio.dll** al posto di quella autentica sfrutta questo ordine di ricerca stabilito dal sistema inserendo la propria copia nella cartella del programma: in questo modo, supponendo che **esempio.dll** autentico sia presente in `%PATH%`, il sistema si interromperà subito

avendo trovato un file con lo stesso nome all'inizio della sua ricerca. Verrà quindi eseguita la copia malevola di `esempio.dll` al posto di quella originale.

L'esecuzione di codice tramite processi permette al malware di lanciare un eseguibile con proprie risorse e thread. Viene sfruttata la funzione `CreateProcess` con il passaggio del parametro `STARTUPINFO` che contiene la gestione dello standard input, dello standard output e dello standard error. Un malware può creare una socket per connettersi ad una shell remota e ...?

L'esecuzione tramite Thread, ovvero i vari pezzi di un processo, permette al malware di avere uno spazio di memoria per permettere l'esecuzione parallela dei thread ed ogni thread ha la propria stack e registri. Il malware utilizza `createThread` con parametro `start` che indica l'indirizzo di una funzione per dare a `start` un'istruzione malevola. Il malware può comunicare con il server di Command And Control senza lanciare un'applicazione e quindi senza che sia rilevabile in modo evidente dal sistema operativo.

I servizi sono le applicazioni che vengono eseguite in background dal sistema operativo senza l'input di un utente. I malware usano due tipologie di servizi per raggiungere persistenza: servizi `WIN32_SHARE_PROCESS` (`svchost.exe`) e servizi `WIN32_OWN_PROCESS`. I servizi implementati si possono trovare nel registro di sistema `HKLM\System\CurrentControlSet\Services` dove un valore di `Start` uguale a `0x03` sta per "Load on Demand" e un valore di `Type` uguale a `0x20` sta per "`WIN32_SHARE_PROCESS`". Per sapere se un malware vuole raggiungere persistenza si devono ricercare le librerie: `OpenSCManager` `CreateService` `StartService`

### 3.5 System Integrity Monitoring

È una tecnica per effettuare l'analisi dinamica di base e consiste nel catturare un'istantanea del file system prima di eseguire il malware, una dopo l'esecuzione del malware e confrontarle per capire quali modifiche sono state apportate al sistema. Tipicamente si attende qualche minuto prima di eseguire la seconda istantanea in modo da dare al malware il tempo di compiere tutte le operazioni per il quale è stato programmato.

Con questa tecnica si possono identificare eventuali cambiamenti apportati a file come modifiche, creazioni, eliminazioni e cambiamenti apportati alle chiavi di registro.

Questa tecnica ha però delle limitazioni; non si possono conoscere eventuali modifiche riguardanti il kernel, quindi i malware di tipo rootkit non sono rilevabili, inoltre non sempre si riesce a tenere traccia dei file temporanei creati durante l'esecuzione malevola. È importante sapere che non tutti i cambiamenti rilevabili nella seconda istantanea sono attribuibili al malware ma alcune modifiche sono apportate normalmente dal sistema operativo.

#### RegShot

RegShot è un software che si occupa di effettuare il System Integrity Monitoring del sistema per rilevare le modifiche apportate da un malware che viene eseguito. Il software effettua una prima scansione delle cartelle e successivamente una seconda che, terminata, genera un documento con un elenco di tutti i cambiamenti rilevati dal confronto delle due scansioni.

Per effettuare la prima istantanea avviare RegShot come amministratore e ci si troverà davanti ad una schermata come quella mostrata in figura 3.2.

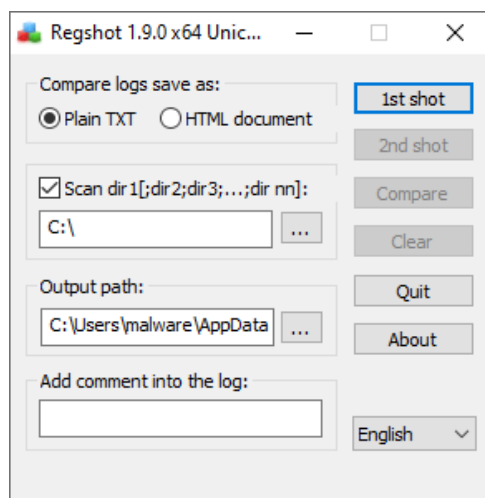


Figura 3.2: Schermata iniziale all'apertura di RegShot. Cliccando su **1st shot** si può avviare la scansione per generare la prima istantanea.

Prima di avviare la scansione del sistema cliccando su **1st shot**, è possibile scegliere quali cartelle far scansionare dal software; per personalizzare questa opzione è necessario spuntare **Scan dir1[;dir2;dir3;...;dir nn]** e scrivere il percorso desiderato. Nel caso della figura 3.2 si è deciso di effettuare l'istantanea per tutto il disco C:\ che è sempre l'opzione consigliabile durante l'analisi dinamica di base. È possibile anche scegliere il percorso di salvataggio del documento finale con i risultati del confronto modificando **Output path**, oltre al poter scegliere se il documento deve essere in formato TXT oppure HTML. In figura 3.2 si è scelto di salvare il file in formato TXT.

Una volta cliccato su **1st shot**, RegShot crea la prima istantanea al termine della quale risulterà selezionabile il tasto **2nd shot** per effettuare la seconda istantanea come in figura 3.3. Ovviamente tra la prima e la seconda istantanea bisogna eseguire il malware e stare attenti a non effettuare altre operazioni che non siano l'avvio della seconda scansione per non inquinare i risultati finali.

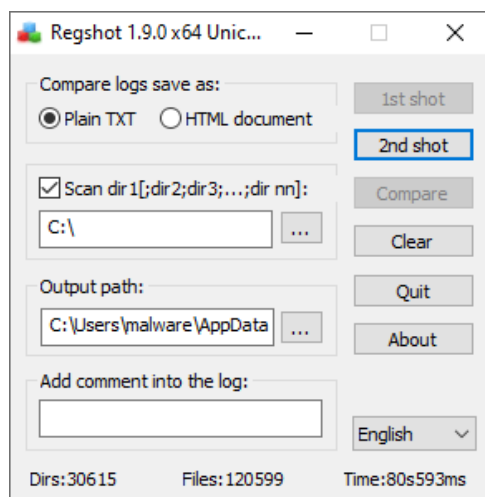


Figura 3.3: Schermata di RegShot al termine della prima scansione. Il tasto **1st shot** è selezionabile e cliccandolo si può avviare la scansione per generare la seconda istantanea.

Quando è stata completata anche la seconda scansione il tasto **Compare** diventa selezionabile e cliccandolo si potrà visualizzare il documento di testo finale che mostra i cambiamenti rilevati.

Il documento finale avrà la forma... forma documento ed esempio per un malware.



## 3.6 Behavioural Monitoring

Tecnica che permette di intercettare in tempo reale i momenti in cui un processo in esecuzione compie determinate azioni, come utilizzare delle funzioni nelle API di Windows oppure accedere e modificare delle chiavi di registro.

Un modo per utilizzare questa tecnica è concentrarsi sul trovare i momenti in cui un processo accedere una serie di operazioni chiave come WriteFile o RegSetValue, oppure determinati processi come Process Create.

Con questa tecnica si possono intercettare i file temporanei non visibili con la tecnica del System Integrity Monitoring, ma non si possono comunque identificare le modifiche a livello del kernel compiute da rootkit.


### Process Monitor

Process Monitor (o ProcMon) è uno strumento fornito da Windows utilizzato per effettuare il Behavioural Monitoring in tempo reale.

Una volta avviato Process Monitor come amministratore il programma inizierà da subito a registrare tutti gli eventi che accadono nel sistema di tutti i processi attivi, come mostrato in figura . Per focalizzarci solo sugli eventi generati da un certo malware dobbiamo impostare dei filtri e bloccare il monitoraggio nei momenti in cui non ci serve: in questo modo avremmo dei risultati più specifici e accurati.

Time of Day	Process Name	PID	Operation	Path	Result	Detail
12:00:05.2543...	Explorer.EXE	5056	ReadFile	C:\Windows\System32\TaskFlowDataEngine.dll	SUCCESS	Offset: 1,411,584, Length: 16,384, I/O Flags: Non...
12:00:05.2546...	svchost.exe	1080	Thread Create		SUCCESS	Thread ID: 3680
12:00:05.2554...	Explorer.EXE	5056	ReadFile	C:\Windows\System32\windows.storage.dll	SUCCESS	Offset: 6,948,864, Length: 16,384, I/O Flags: Non...
12:00:05.2555...	Explorer.EXE	5056	ReadFile	C:\Windows\System32\TaskFlowDataEngine.dll	SUCCESS	Offset: 1,395,200, Length: 16,384, I/O Flags: Non...
12:00:05.2558...	Microsoft Edge...	6244	RegQueryKey	HKLM	SUCCESS	Query: HandleTags, HandleTags: 0x0
12:00:05.2558...	Microsoft Edge...	6244	RegQueryKey	HKLM	SUCCESS	Query: Name
12:00:05.2558...	Microsoft Edge...	6244	RegCreateKey	HKLM\Software\WOW6432Node\Microsoft\EdgeUp...	SUCCESS	Desired Access: All Access, Disposition: REG_OP...
12:00:05.2559...	Microsoft Edge...	6244	RegSetInfoKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Edge...	SUCCESS	KeySetInformationClass: KeySetHandleTagsInfor...
12:00:05.2559...	Microsoft Edge...	6244	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Edge...	SUCCESS	Type: REG_DWORD, Length: 4, Data: 3
12:00:05.2559...	Microsoft Edge...	6244	RegCloseKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Edge...	SUCCESS	
12:00:05.2561...	Explorer.EXE	5056	ReadFile	C:\Windows\System32\ui70.dll	SUCCESS	Offset: 1,242,112, Length: 16,384, I/O Flags: Non...
12:00:05.2565...	msedge.exe	5432	ReadFile	C:\Windows\System32\PHLPAPI.DLL	SUCCESS	Offset: 210,432, Length: 6,656, I/O Flags: Non-ca...
12:00:05.2565...	Explorer.EXE	5056	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
12:00:05.2565...	Explorer.EXE	5056	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTags, HandleTags: 0x0
12:00:05.2566...	Explorer.EXE	5056	RegOpenKey	HKCU\Software\Classes\Users\malware\AppData...	NAME NOT FOUND	Desired Access: Read
12:00:05.2566...	Explorer.EXE	5056	RegOpenKey	HKCR\Users\malware\AppData\Local\Temp\Pr	NAME NOT FOUND	Desired Access: Read
12:00:05.2566...	Explorer.EXE	5056	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
12:00:05.2566...	Explorer.EXE	5056	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTags, HandleTags: 0x0
12:00:05.2567...	Explorer.EXE	5056	RegOpenKey	HKCU\Software\Classes\Users\malware\AppData...	NAME NOT FOUND	Desired Access: Read
12:00:05.2567...	Explorer.FXE	5056	RegOpenKey	HKCR\Users\malware\AppData\Local\Temp\Pr	NAME NOT FOUND	Desired Access: Read

Figura 3.4: Schermata iniziale all'avvio di Process Monitor senza nessun filtro impostato. Il software, come si può vedere dalla figura, inizia subito a raccogliere tutti gli eventi del sistema.

I filtri su Process Monitor sono impostabili cliccando sul tasto  e, una volta aperta una nuova finestra, selezionando il tipo di filtro, la relazione e il nome dell'operazione come mostrato in figura . Il primo filtro da impostare, che è anche il filtro principale è il nome del processo malevolo generato dall'esecuzione del malware e si può selezionare come asserzione **Process Name is malware.exe**, con malware.exe il nome del processo in analisi. Altri filtri che si possono tipicamente settare sono riassunti nella tabella .

Operation is WriteFile

Operation is RegSetValue

Operation is Process Create

Operation is SetDispositionInformationFile

Operation begins with TCP

La figura 3.5 mostra un esempio di impostazione dei filtri principali mostrati nella tabella.

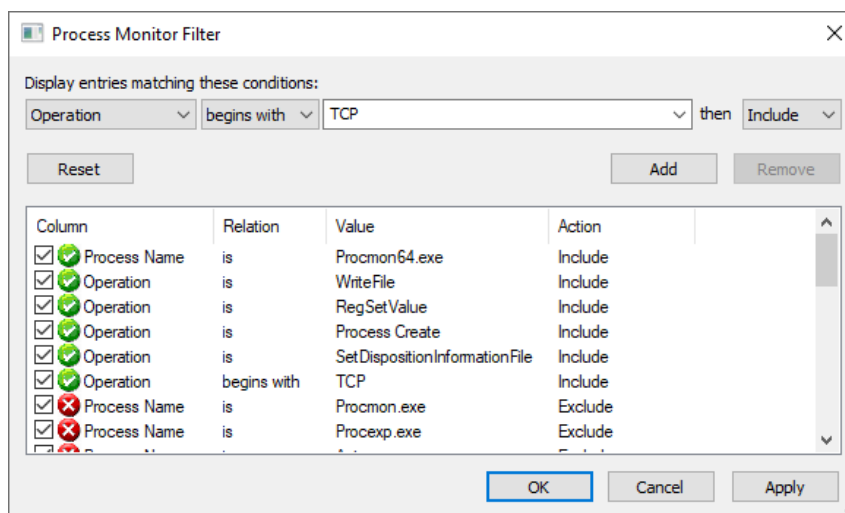


Figura 3.5: ...

Per utilizzare Process Monitor in maniera pratica ed analizzare il comportamento di un malware, per prima cosa, oltre ad aver impostato i filtri, bisogna fermare la cattura cliccando sull'icona ? che diventerà ? e starà ad indicare che in quel momento non si stanno più registrando eventi. Per pulire la memoria di Process Monitor da tutti gli eventi registrati fino ad ora cliccare sull'icona ? . A questo punto si deve riattivare la cattura ed eseguire il malware da analizzare, aspettando un po' di tempo per lasciare che il malware esegua le proprie funzioni. Supponiamo, ad esempio, di eseguire un malware chiamato *budget-report.exe* che si maschera da file PDF.

Dopodichè fermare la cattura ed aggiungere ai filtri il nome del processo malware: in questo caso il filtro sarà **Process Name is budget-report.exe**. Si ha quindi il risultato di tutte le azioni filtrate compiute dal malware sul sistema, come in figura 3.6.

Time of Day	Process Name	PID	Operation	Path	Result	Detail
12:10:06.8085...	budget-report.exe	2148	WriteFile	C:\Users\malware\ntuser.dat.LOG2	SUCCESS	Offset: 233,472, Leng...
12:10:06.8798...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:06.8822...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	SUCCESS	Type: REG_DWORD...
12:10:06.8823...	budget-report.exe	2148	WriteFile	C:\Users\malware\ntuser.dat.LOG2	SUCCESS	Offset: 344,064, Leng...
12:10:06.8827...	budget-report.exe	2148	WriteFile	C:\Users\malware\AppData\Local\Temp\12648430.bat	SUCCESS	Offset: 0, Length: 23...
12:10:07.0724...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	SUCCESS	Type: REG_DWORD...
12:10:07.0724...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	SUCCESS	Type: REG_DWORD...
12:10:07.0724...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	SUCCESS	Type: REG_DWORD...
12:10:07.0725...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	SUCCESS	Type: REG_DWORD...
12:10:07.0756...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	SUCCESS	Type: REG_DWORD...
12:10:07.0756...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	SUCCESS	Type: REG_DWORD...
12:10:07.0756...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	SUCCESS	Type: REG_DWORD...
12:10:07.0756...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	SUCCESS	Type: REG_DWORD...
12:10:07.1326...	budget-report.exe	2148	WriteFile	C:\Users\malware\AppData\Local\Temp\12648430.bat	SUCCESS	Offset: 0, Length: 4,09...
12:10:07.1399...	budget-report.exe	2148	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS	PID: 5796, Command I...
12:10:07.1619...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:07.1621...	budget-report.exe	2148	WriteFile	C:\Users\malware\ntuser.dat.LOG2	SUCCESS	Offset: 364,544, Leng...
12:10:09.6836...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:12.1920...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:12.1973...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:12.1975...	budget-report.exe	2148	WriteFile	C:\Users\malware\ntuser.dat.LOG1	SUCCESS	Offset: 0, Length: 36,8...
12:10:14.7076...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:14.7101...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:14.7102...	budget-report.exe	2148	WriteFile	C:\Users\malware\ntuser.dat.LOG1	SUCCESS	Offset: 36,864, Length...
12:10:17.2282...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:17.2332...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:19.7397...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:19.7436...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:22.2702...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:22.2768...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:24.7874...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...
12:10:24.7936...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\12648430	SUCCESS	Type: REG_SZ, Leng...
12:10:27.3033...	budget-report.exe	2148	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	SUCCESS	Type: REG_DWORD...

Figura 3.6: Risultato parziale dell'analisi degli eventi registrati dal malware budget-report.exe. Si può notare nelle righe indicate dalle frecce rosse che il malware agisce sulle chiavi di registro per crearsi una persistenza, mentre nelle righe indicate dalle frecce blu che il malware ha creato dei file temporanei durante la sua esecuzione.

Il malware per prima cosa scrive su un file chiamato *ntuser.dat.LOG2* e si crea una chiave di registro con un proprio numero identificativo

HKCU\SOFTWARE\12648430  
ed accede ad un'altra chiave di registro  
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden  
per modificarne il valore da 4 a 2. Quest'ultima chiave gestisce la visualizzazione di file nascosti su Windows quindi si può dedurre che il malware adotti strategie per nascondersi.

Le funzionalità principali di questo malware, mostrate anche in figura 3.6, sono il raggiungimento della persistenza sulla macchina della vittima, che gli garantisce l'esecuzione ad ogni accesso dell'utente. Oltre alla persistenza è importante notare anche la creazione e scrittura di file temporanei (indicati in figura 3.6 con delle frecce blu) che non sarebbero in grado di rilevare con un'analisi di tipo System Integrity Monitoring.

Su Process Monitor è anche possibile visualizzare informazioni sugli eventi come le chiavi di registro modificate e il nuovo valore impostato, basta fare doppio click sulla chiave di registro. La figura 3.7 mostra la finestra di dettaglio di un'operazione di RegSetValue.

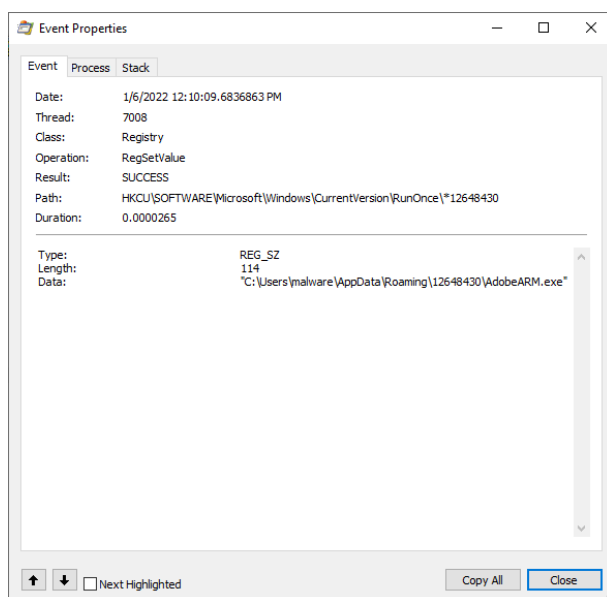


Figura 3.7

## 3.7 Altri strumenti utili per l'analisi dinamica di base

### 3.7.1 Simulare la presenza di una rete

Può capitare che alcune tipologie di malware cerchino di capire se il computer sul quale sono installati sia connesso alla rete oppure no, in questo modo possono evitare di eseguire alcune operazioni e mettere in difficoltà gli analisti che effettuano analisi di base. Per questo esistono degli strumenti che servono a simulare del traffico di rete e far credere ai processi in esecuzione sulla macchina che è presente una connessione ad Internet.

#### FakeNet

FakeNet è uno strumento utilizzato per simulare la presenza di una rete in un ambiente in cui la rete è stata disattivata.

Avviando FakeNet come amministratore si aprirà un Prompt dei Comandi come quello in figura 3.8 che segna le operazioni compiute da FakeNet.

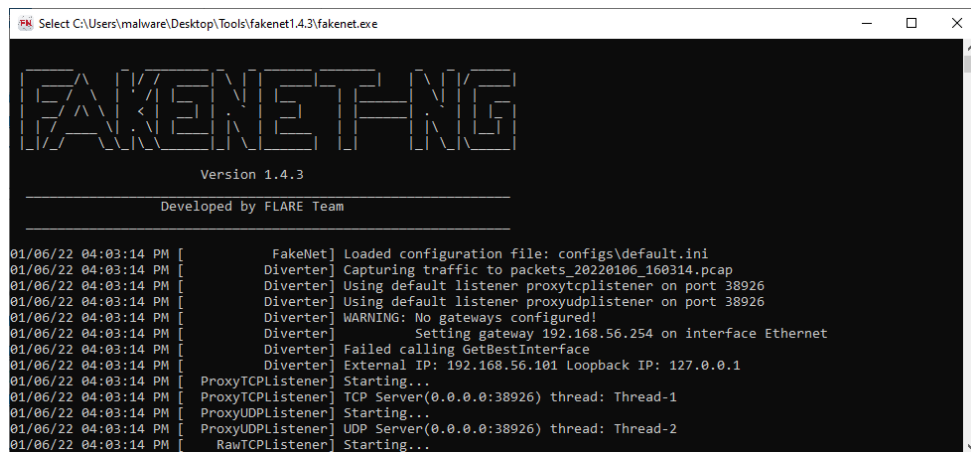


Figura 3.8: Prompt dei Comandi visualizzato all'apertura di FakeNet.

## InetSim

InetSim è un semplice strumento da linea di comando che può essere lanciato da terminale. Siccome questo strumento viene sviluppato solo per sistemi Linux si consiglia di avviare una macchina virtuale aggiuntiva alla macchina Windows e connettere entrambe le macchine ad una rete interna (sarà comunque spiegato nella prossima sezione).

Si può avviare semplicemente con il comando:

```
sudo inetsim
```

Le configurazioni di InetSim sono modificabili con il comando

```
sudo nano /etc/inetsim/inetsim.conf
```

in cui è possibile scommentare le righe che interessano e salvare il file con **Ctrl+O** e uscire con **Ctrl+X**.

## 3.8 Come effettuare un'analisi dinamica di base completa

Per effettuare una analisi dinamica di base completa, utilizzando tutti gli strumenti descritti finora nel capitolo, si deve seguire una precisa procedura. I passi da seguire sono:

1. Avviare FakeNet;
  2. Aprire Process Monitor e fermare la cattura;
  3. Creare la prima istantanea del sistema con RegShot;
  4. Riprendere la cattura di Process Monitor;
  5. Eseguire il malware e aspettare qualche minuto;
  6. Fermare la cattura di ProcessMonitor;
  7. Creare la seconda istantanea del sistema con RegShot e compararle;
  8. Applicare i filtri a Process Monitor e visualizzare i risultati;
- ...importante seguire l'ordine perchè...

### 3.8.1 Analisi del traffico di rete

Si può anche utilizzare un metodo alternativo a FakeNet per simulare un'attività di rete e nel contempo effettuare una cattura dei pacchetti per monitorare le connessioni ad internet del malware.

## Capitolo 4

# Analisi statica avanzata

I malware vengono scritti generalmente in linguaggi di programmazione ad alto livello, la maggior parte in C o C++ e vengono compilati da un compilatore che genera un file EXE. Questo file EXE è essenzialmente una sequenza di byte. Guardando il codice macchina si vede che i byte in esadecimale sono rappresentati da due cifre, quindi ci si trova davanti a contenuti del tipo:

... esempio sequela di esadecimali

Per effettuare l'analisi statica avanzata ed analizzare il codice partendo da istruzioni binarie si ricorre al cosiddetto *Reverse Engineering*.

### 4.1 Reverse Engineering

#### 4.1.1 General Stack Frame

da riguardare

#### 4.1.2 Riconoscere le funzioni in Assembly

##### IDA Freeware

IDA è un software utilizzato nell'ambito del Reverse Engineering e permette il disassemblaggio di un file eseguibile, ovvero permette di risalire alle istruzioni Assembly di cui è composto. Esistono tre versioni di IDA: *Pro*, *Freeware* e *Demo*. La versione Pro permette di disassemblare non solo file eseguibili EXE di Windows ma anche file eseguibili per Linux e altri formati. La versione Freeware invece permette solo il disassemblaggio di eseguibili Windows in 32 bit e 64 bit. Per semplicità qui di seguito verrà descritto l'utilizzo della versione Freeware.

Per iniziare bisogna aprire IDA e cliccare su **New** per scegliere un file da disassemblare. Una volta fatto dovremmo trovarci davanti ad una schermata come quella in figura 4.1.

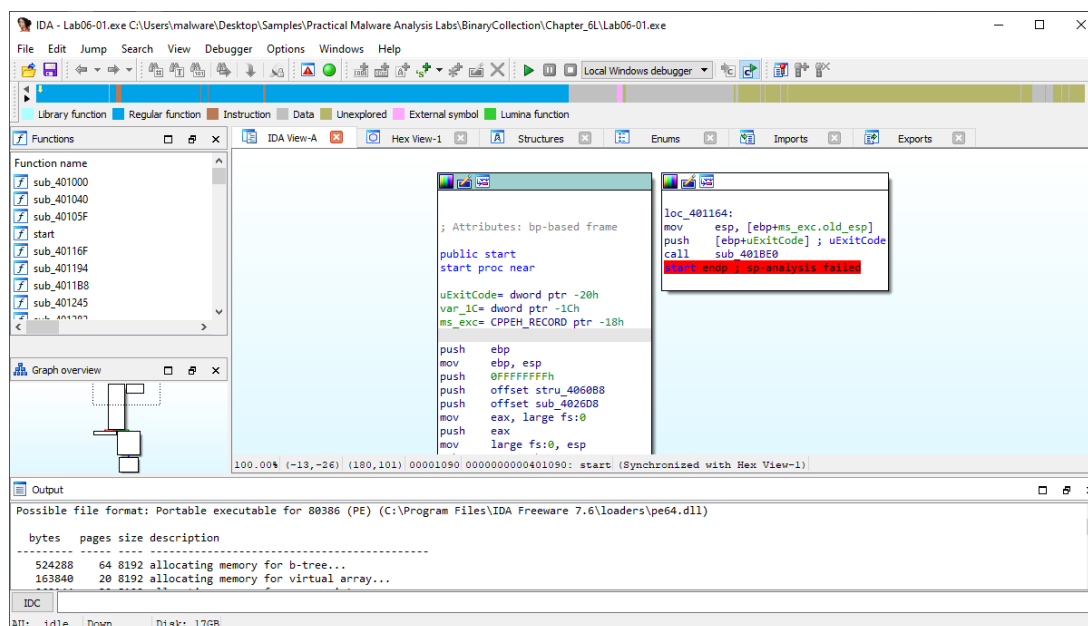


Figura 4.1: Schermata iniziale di IDA Freeware all'apertura di un file eseguibile EXE.

IDA è composto da una finestra centrale in cui è possibile visualizzare il codice disassemblato in due forme: o in *modalità grafo*, dove viene mostrato il codice come una sorta di diagramma di flusso, oppure in *modalità testuale*, mostrando le istruzioni riga per riga. Per passare da una modalità all'altra si può premere la barra spaziatrice oppure cliccare il tasto destro e selezionare **Graph View** o **Text View**.

Quando viene aperto un file, IDA simula il caricamento in memoria del file e si posiziona sul punto del codice in cui partirebbe la memoria. Con la modalità testo è possibile vedere per ogni riga di istruzioni la sezione dell'eseguibile in cui è presente in memoria (per esempio **.text**) e il possibile indirizzo di memoria in cui verrebbe allocato.

Nella modalità grafo, il codice viene organizzato in modo che ad ogni JUMP (**jmp**) condizionale vengono visualizzati nel diagramma due possibili percorsi a due blocchi: uno collegato con una freccia verde che indica il caso in cui la condizione della jump è verificata e uno collegato ad una freccia rossa che indica il caso in cui la condizione della jump non è verificata. I blocchi di istruzioni sono separati anche nel caso di call ad una routine o nel caso di loop dove troviamo delle frecce blu che tornano ad un blocco precedente. Nel caso della modalità testuale si può trovare la divisione delle routine del codice per la presenza di linee tratteggiate nel lato sinistro delle istruzioni, mentre le JUMP che soddisfano le condizioni sono unite da linee continue, sempre a lato del codice.

La finestra **Functions**, che solitamente si trova nella parte sinistra all'interno della finestra di IDA, mostra tutte le funzioni (o subroutine) presenti nel codice e selezionandole è possibile collocarsi automaticamente sulla parte di codice dove inizia quella funzione. La finestra di **Output** in basso mostra alcune informazioni sul caricamento del codice.

Altre finestre interessanti sono quelle presenti accanto alla finestra dove è contenuto il codice. Ad esempio **Imports** mostra le funzioni importate dal malware e il loro indirizzo di memoria, mentre **Exports** mostra eventuali librerie esportate. È possibile anche visualizzare le stringhe presenti nell'eseguibile con la finestra **Strings** (se la finestra non è presente la si può inserire da **View > Open subviews > Strings**). La finestra **Hex View-1** mostra il codice che si sta analizzando come sequenze di byte in esadecimale insieme ad una decodifica in caratteri ASCII.

## Analizzare il codice con IDA Freeware

All'avvio di IDA, le istruzioni verranno mostrate a partire da una funzione chiamata **start** che corrisponde alla AddressOfEntryPoint (? o qualcosa di simile) e non corrisponde alla funzione **main** (che bisogna trovare manualmente come descritto in precedenza). Selezionando una funzione dalla lista nella finestra **Functions** spesso è possibile visualizzare la lista delle variabili utilizzate nella funzione e degli argomenti della funzione tipicamente identificate rispettivamente come **var\_xx** e **arg\_xx**, con xx offset rispetto



al puntatore sullo stack; per variabili utilizzate per le chiamate a librerie importate note IDA riesce ad assegnargli il nome della libreria. La figura 4.2 mostra un esempio di variabili e argomenti di una funzione.

```
.text:004037AC ; ===== S U B R O U T I N E =====
.text:004037AC
.text:004037AC ; Attributes: bp-based frame
.text:004037AC
.text:004037AC sub_4037AC      proc near                ; CODE XREF: sub_403795+E↑p
.text:004037AC
.text:004037AC var_C          = dword ptr -0Ch
.text:004037AC var_8          = dword ptr -8
.text:004037AC var_4          = dword ptr -4
.text:004037AC arg_0          = dword ptr  8
.text:004037AC arg_4          = dword ptr  0Ch
.text:004037AC arg_8          = dword ptr  10h
.text:004037AC arg_C          = dword ptr  14h
.text:004037AC
.text:004037AC      push     ebp
.text:004037AD      mov      ebp, esp
.text:004037AF      sub      esp, 0Ch
.text:004037B2      push     ebx
.text:004037B3      and      [ebp+var_8], 0
```

Figura 4.2: Visualizzazione in IDA Freeware delle variabili e degli argomenti di una funzione chiamata `sub_4037AC` nella modalità testuale. In questo caso le variabili contenute nella funzione sono 3: `var_C`, `var_8` e `var_4`, mentre gli argomenti passati alla funzione sono 4: `arg_0`, `arg_4`, `arg_8` e `arg_C`. Il commento all'inizio `Attributes: bp-based frame` è stato inserito da IDA e indica che gli offset dei nomi delle variabili e argomenti sono riferiti al registro EBP.

Tipicamente valori come `var_4` e `var_8` significano che la prima variabile sarà accessibile attraverso il registro EBP con un offset di `-4` e la seconda sarà accessibile con un offset di `-8`. Per gli argomenti delle funzioni invece un valore come `arg_0` sarà accessibile con un offset dall'EBP di `+8` e `arg_4` con un offset di `+12`. Quando in un istruzione troviamo la parola `offset` significa che si riferisce all'indirizzo di memoria e non al valore all'indirizzo di memoria. Ad esempio:

```
push offset stru_4060B8
```

dove `stru_4060B8` è una variabile globale, significa che non si andrà a salvare il valore della variabile globale ma il suo indirizzo.

È possibile assegnare un nome a variabili e funzioni che posso identificare con determinati comportamenti a cui, per praticità, voglio dare un nome evocativo. Posso farlo cliccando sulla variabile o la funzione selezionata e premere la lettera `N` oppure con il tasto destro cliccare su **Rename**; si aprirà una finestra in sovraimpressione che mi permetterà di assegnare un nome che verrà mantenuto per ogni ricorrenza all'interno del file. Si possono inoltre aggiungere dei commenti ad ogni riga premento il tasto `:` oppure con il tasto destro selezionando **Enter comment...**

Per posizionarsi all'indirizzo di una determinata funzione e poterla visualizzare si può semplicemente selezionarne il nome con il mouse e premere la lettera `G`, fare doppio click oppure anche premere il tasto destro e selezionare **Jump to operand**. Per tornare alla posizione precedente si può cliccare la freccia blu che va verso sinistra nella barra degli strumenti in alto. Se si vuole invece vedere la lista di tutti i riferimenti di una funzione ovvero tutti gli indirizzi dei punti in cui è stata chiamata si può selezionare con il mouse la funzione e premere `X` oppure con il tasto destro selezionare **Jump to xref to operand...** e si visualizzerà un lista dei riferimenti. Andando inoltre al punto in cui viene dichiarata una funzione o una variabile, IDA mostra un commento che indica il riferimento nella forma `[DATA|CODE] XREF: <function><offset>[↑|↓] <type>`, per esempio:

```
sub_401000 ; CODE XREF: sub_401040+4↓p
```

significa che la funzione `sub_401000` è richiamata dalla funzione `sub_401040` all'offset `+4` dall'indirizzo della funzione. Un'esempio simile si può notare anche nel codice all'interno della figura 4.2. Per una variabile invece un riferimento del tipo:

```
dword_409938 ; DATA XREF: start+30↑w
```

significa che la variabile `dword_409938` è modificata nella funzione `start` all'offset `+30`. Il fatto che la variabile venga modificato lo si può capire dalla `w` alla fine, se ci fosse stato la `r` sarebbe stata riferita in lettura. IDA può anche mostrare una visualizzazione grafica dei riferimenti ad una funzione selezionata con l'opzione `View > Open subviews > Proximity browser`. La figura 4.3 mostra un esempio di questa modalità.

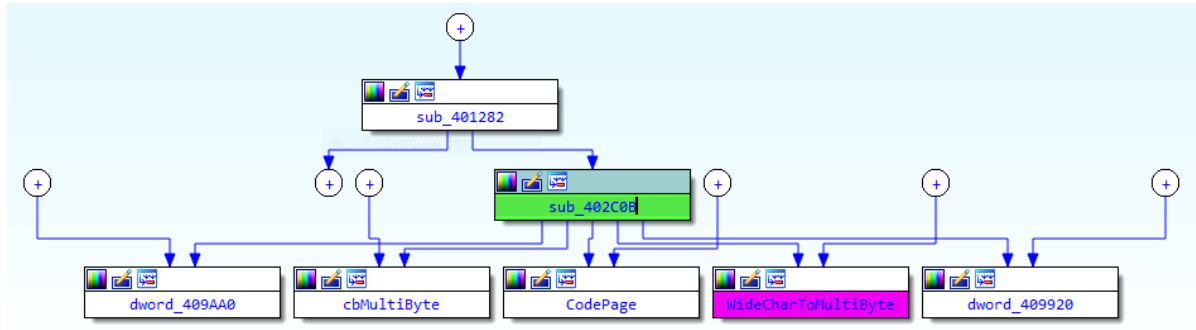


Figura 4.3: Esempio di visualizzazione dei riferimenti per la funzione `sub_402C0B`. Si può notare che questa funzione è chiamata da `sub_401282`, mentre al suo interno chiama le variabili `dword_409AA0`, `cbMultiByte`, `CodePage`, `WideCharToMultiByte` e `dword_409920`. Per espandere l'albero a tutte le altre funzioni connesse si può cliccare su i simboli `+`.

I riferimenti possono essere trovati anche per le stringhe: basta cliccare su una stringa da cercare nella finestra **Strings** e premere la lettera **X**. Questo mi darà la lista di tutte le funzioni che chiamano quella stringa.

## Ghidra

Questo software è stato progettato dalla NSA (National Security Agency) americana.



## Capitolo 5

# Analisi dinamica avanzata

### 5.1 Debugging

x32dbg

## Capitolo 6

# Analisi di script e documenti

### 6.1 Analisi di documenti PDF

#### 6.1.1 Struttura di un PDF

#### 6.1.2 Analisi di un PDF

### 6.2 Analisi di codice Javascript

## Capitolo 7

# Tecniche di offuscamento

### 7.1 Modifica delle stringhe

Gli attaccanti possono tentare di rendere difficile l'analisi delle stringhe inserendo all'interno del programma stringhe casuali o di tutt'altro significato volte a far credere che il file non sia un file malevolo.

In alcuni casi possono cifrare delle stringhe per oscurarne il contenuto oppure usare dei sistemi di codifica in base64. L'utilizzo di una codifica in base64 all'interno del malware può essere indicata, tramite l'analisi delle stringhe, dalla presenza di stringhe come ABC...abc...+/. Si faccia riferimento al capitolo 2(?). Può essere utilizzata anche la codifica XOR. Il problema di utilizzare una codifica XOR è il fatto che, a causa della presenza del carattere nullo alla fine di una stringa, riesco a riconoscere la chiave e recuperare l'intera stringa.

### 7.2 Impacchettamento

#### 7.2.1 Come spaccettare un malware

Ci sono diverse tecniche per fare l'unpacking di un malware.

##### UPX

Nel caso un malware sia impacchettato con UPX si può scaricare il software upx e dal terminale dare il comando:

```
upx -d -o <percorso_output> <percorso_malware>
```

dove <percorso\_output> corrisponde al percorso in cui volete salvare il file spaccettato, includendo anche il nome del nuovo file, mentre <percorso\_malware> corrisponde al percorso in cui si trova il malware da spaccettare. Nella figura 7.1 è mostrato l'output risultante se l'operazione va a buon fine.

```
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96w Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020

  File size      Ratio      Format      Name
  -----
  57344 <-    43520    75.89%    win32/pe    unpacked.exe

Unpacked 1 file.
```

Figura 7.1: Messaggio che viene mostrato dopo aver eseguito UPX per spaccettare un file e salvarlo in un'altra cartella.

## Capitolo 8

# Esempi pratici di Malware Analysis

## Capitolo 9

# Introduzione alla Malware Analysis per sistemi Unix

### 9.1 Sistemi Linux

### 9.2 Sistemi Mac

# Appendice A

## Sistema operativo Windows

### A.1 Librerie ed API in Windows

Le librerie più comunemente utilizzate in Windows sono riassunte nella seguente tabella e verranno analizzate singolarmente citando le funzioni più importati.

### A.2 DLL

Una DLL è un file che contiene il riferimento alle librerie che un programma carica durante la sua esecuzione. Sono molto simili ai file .exe.

La funzione `DllMain` è la funzione principale all'interno di un DLL ed è quella che verrà chiamata da un file eseguibile che vuole caricare le proprie librerie.

#### A.2.1 Tipi di dati

Le API di Windows lavorano con particolari tipi di dati diversi dai tipi a cui un normale programmatore potrebbe essere abituato. Sono elencati di seguito:

- **WORD**: con prefisso `w`, è un valore senza segno a 16 bit;
- **DWORD**: con prefisso `dw`, è un valore senza segno a 32 bit;
- **Handle**: con prefisso `H`, è un riferimento ad un oggetto;
- **Long Pointer**: con prefisso `LP`, sono puntatori ad altri tipi di dati.

I prefissi sopracitati corrispondono ai valori iniziali delle variabili che contengono un particolare tipo, ad esempio un valore intero senza segno a 32 bit inizierà con `dw`. Questa notazione è detta **notazione ungherese**.

I tipi **Handle** sono in pratica puntatori ad oggetti e per ottenere un handle associato ad un file basterà chiamare la funzione `CreateFile` nel momento in cui si vuole crearlo e restituirà un tipo handle. Gran parte delle funzioni per modificare file e oggetti richiede come parametro un handle a tale file.

#### A.2.2 Kernel32.dll

#### A.2.3 Advapi32.dll

#### A.2.4 User32.dll

#### A.2.5 Gdi32.dll

#### A.2.6 Ntdll.dll

#### A.2.7 WSock32.dll

#### A.2.8 Ws2\_32.dll

È una libreria per gestire le connessioni in rete a basso. Ha le seguenti funzioni: `socket`: crea un socket `bind`: attach una socket ad una particolare porta `listen`: indica che una socket è in ascolto in attesa di

una connessione in entrata. accept: apre una connessione ad un socket remoto e accetta la connessione. connect: apre una connessione ad una socket remota che deve attendere per la connessione recv: riceve dati dalla socket remota send: invia dati sulla socket remota

Prima è necessario chiamare WSStartup (?)

### A.2.9 Wininet.dll

Implementa funzioni che utilizzano i protocolli di rete ad alto livello, in particolare del livello applicativo della suite di protocolli TCP/IP. Alcuni esempi sono HTTP e FTP. Le funzioni contenute in questa libreria sono: InternetOpen: connette ad Internet InternetOpenURL: connette ad un URL InternetReadFile: legge il contenuto di un file scaricato dalla rete.

## A.3 Formato Portable Executable

Il formato PE (o Portable Executable) è un formato di file molto importante durante l'esecuzione di un file eseguibile. Il Loader, che si occupa di caricare in memoria l'eseguibile, deve caricare il codice del file, la sezione Data e importare le librerie. Per sapere tutte le informazioni per allocare lo spazio di memoria corretto e le librerie utilizzate da caricare utilizza il file PE. Il file PE contiene al suo interno informazioni come quanto spazio di memoria deve essere allocato per eseguire un file .exe, quali librerie o file DLL caricare, come sono divise le sezioni all'interno dell'eseguibile ovvero dove finisce il codice e iniziano i dati.

Il Loader quando alloca lo spazio di memoria da un file PE, associa un indirizzo di memoria che punta alla sezione allocata chiamato **Image Base**, inoltre il file PE ha un relative virtual address, che parte dalla prima sezione di memoria del PE e non è quindi assoluto. Per recuperare una certa allocazione nello spazio dedicato al file PE bisognerà calcolare:  $ImageBase + RelativeVirtualAddress$ .

Nel PE file ci possono essere anche informazioni su come viene eseguito il file .exe, ad esempio se il programma è un programma che viene lanciato da linea di comando oppure se ha un'interfaccia grafica. Un file PE è composto da diverse sezioni o header:

- DOS Header
- PE Header

### A.3.1 DOS Header

La prima componente di un file in formato Portable Executable è il DOS Header. Questo è rappresentato dalla struttura sottostante che si trova all'interno dell'header winnt.h:

... struttura DOS header

e \_lfanew è un puntatore al header successivo, ovvero il PE Header.

### A.3.2 PE Header

Il PE Header è una struttura di tre elementi: una signature, il File Header e l'Optional Header. La signature è una stringa ascii dal valore "PE" per indicare che il file in questione è un file PE.

### A.3.3 File Header

In questo header ci sono quattro campi che servono a dare informazioni sul file corrente. Il primo campo è Machine che indica l'architettura su cui l'eseguibile è stato progettato per girare: se a 32 bit (014C) o a 64 bit (8664). Un altro campo è il TimeDateStamp che indica quando il file è stato compilato in seconda dal 31 dicembre 1969 alle 4 PM. Il NumberOfSections indica il numero di sezioni contenute all'interno del PE file. L'ultimo campo sono le caratteristiche: una struttura con un insieme di flag e in base a quale flag è settato capiamo che tipo di file si ha davanti.

### A.3.4 Optional Header

L'Optional Header è un'altra struttura che, nonostante il nome, non è opzionale ed ha all'interno molti campi.

... struttura ... descrizione di tutti i campi.

### A.3.5 Section Header

.text: contiene il codice da eseguire. .data: contiene sia variabili globali o statiche il cui valore è stato inizializzato al momento della compilazione .bss: contiene dati non inizializzati .idata: contiene la import address table. edata: contiene funzioni esportate dalla libreria. .reloc: ??? .src: risorse

Per ciascuna sezione esiste un header con valore Name di un byte che ci indica il nome, VirtualAddress, PointerToRawData, VirtualSize e SizeOfRawData. Esiste anche un campo Characteristics che contiene i permessi di una sezione, ovvero di scrittura, lettura ed esecuzione. .text deve essere sia leggibile che eseguibile .data deve essere sia leggibile che scrivibile.

### A.3.6 Risorse

Possono essere icone, immagini o qualsiasi componente dell'interfaccia grafica di un file eseguibile.

## A.4 Registro di sistema

Contengono informazioni su quali programmi eseguire quando un utente si collega alla macchina o quale programma è associato ad un determinato tipo di file.

I registri nel sistema Windows sono organizzati seguendo una struttura gerarchica, simile a quella del file system. Il registro radice contiene 5 cartelle principali:

- HKEY\_CLASSES\_ROOT
- HKEY\_CURRENT\_USER
- HKEY\_LOCAL\_MACHINE
- HKEY\_USERS
- HKEY\_CURRENT\_CONFIG

### A.4.1 HKEY\_CLASSES\_ROOT

Contiene tutte le informazioni su quali programmi eseguire associati ad ogni estensione dei file. Per esempio ...

### A.4.2 HKEY\_CURRENT\_USER

Contiene tutte le informazioni sui programmi e le impostazioni legate all'utente corrente che sta utilizzando la macchina.

### A.4.3 HKEY\_LOCAL\_MACHINE

Contiene tutte le informazioni legate alla macchina.

### A.4.4 HKEY\_USERS

Contiene le informazioni per ciascun utente creato sulla macchina. Ogni utente ha una stringa che li associa. ... Le informazioni in HKEY\_CURRENT\_USER sono una copia dell'utente corrente in questo registro.

### A.4.5 HKEY\_CURRENT\_CONFIG

...



## Appendice B

# Linguaggio Assembly

In questa sezione si illustrerà in maniera sintetica il funzionamento del linguaggio Assembly per architettura x86 al fine di comprendere il processo di disassemblaggio del codice malevolo che si vuole analizzare.

Il linguaggio Assembly è il linguaggio specifico per ogni architettura, qui di seguito si descriverà il linguaggio per architettura x86.

### B.1 Richiamo sull'architettura x86

L'architettura x86, ma un po' tutte le architetture, ha tre componenti fondamentali: la *RAM*, la *CPU* e i *dispositivi di Input/Output*.

La RAM (o Random Access Memory) è utilizzata per l'esecuzione dei programmi in quanto per ogni programma viene allocata un'area di memoria per contenerlo.

La CPU (o Central Processing Unit) si occupa dell'esecuzione delle istruzioni del programma. Dentro la CPU ci sono tre componenti: la Control Unit, che si occupa di recuperare le istruzioni da eseguire dallo spazio di memoria allocato per il programma, la ALU che riceve le istruzioni dal Control Unit e si occupa di decodificare le istruzioni ed eseguirle, i Registri che sono delle piccole unità di memoria utili per memorizzare i valori durante l'esecuzione delle istruzioni.

I dispositivi di Input/Output sono ad esempio la tastiera, il mouse, il monitor e servono a ricevere in ingresso i dati che l'utente vuole elaborare e restituirglieli.

..figura

### Struttura della memoria

Contiene le librerie importate, lo stack, l'heap, e sezione data non allocata.

Lo stack ha una struttura a pila LIFO (Last In First Out). Gli argomenti di una funzione vengono inseriti nello stack.

### B.2 Endianess

Specifica l'ordine in cui viene memorizzato un codice esadecimale associato ad una procedure. In poche parole si decide da che parte deve stare il byte meno significativo. Si può distinguere tra **Big Endian** e **Little Endian**.

Se si parla di **Big Endian**, un eventuale indirizzo esadecimale 0x12345678 viene salvato così com'è scritto, ovvero con il carattere più significativo a sinistra. Big Endian si utilizza all'interno dei pacchetti di rete.

Con il **Little Endian** invece, l'indirizzo utilizzato precedentemente verrà salvato come 0x78563412 (si ricorda che raggruppiamo i caratteri su 8 bit quindi si considerano coppie di caratteri esadecimali),

ovvero con il byte più significativo a destra. Questa tipologia viene utilizzata per salvare gli indirizzi in memoria RAM.

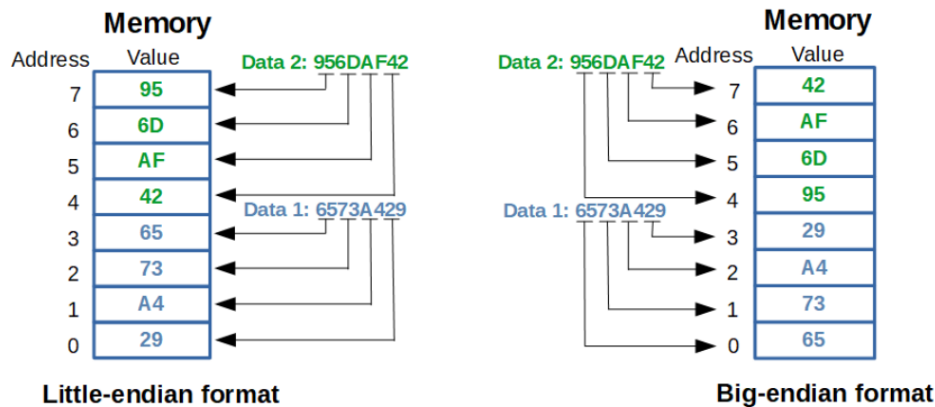


Figura B.1: Rappresentazione del salvataggio degli indirizzi secondo le politiche di Endianess.

## B.3 Registri della CPU

I registri si dividono in categorie: registri **generali**, registri di **segmento**, **flag di stato** e l'*Instruction pointer*.

### Registri generali

Detti anche *general-purpose registers*, sono registri che hanno lo scopo di memorizzare dati di operazioni aritmetiche o indirizzi di memoria. Questi registri sono 8 e possono essere immaginati come array lunghi 32 bit. Qui di seguito sono elencati i principali registri:

- **EAX**: registro utilizzato tipicamente per memorizzare i valori di ritorno delle funzioni;
- **EBX**: utilizzato come *base pointer* alla sezione .data;
- **ECX**: è spesso utilizzato per implementare dei contatore per cicli come for e while;
- **EDX**: puntatore Input/Output;
- **ESI**: è il *source pointer* per operazioni con le stringhe;
- **EDI**: è il *destination pointer* per operazioni con le stringhe;
- **ESP**: utilizzato come *stack pointer*, ovvero è un puntatore che punta sempre alla cima dello stack;
- **EBP**: utilizzato come *base pointer*, ovvero è un puntatore che punta all'inizio del frame allocato nello stack per gestire la chiamata di funzioni.

I registri EAX, EBX, ECX e EDX vengono utilizzati per salvare gli operandi durante le operazioni aritmetiche, mentre ESI e EDI sono utilizzati per operazioni con le stringhe.

32 bits	Lower 16 bits	Lower 8 bits	Upper 8 bits
EAX	AX	AL	AH
EBX	BX	BL	BH
ECX	CX	CL	CH
EDX	DX	DL	CL
ESI	SI		
EDI	DI		
EBP	BP		
ESP	SP		

Figura B.2: Tabella che rappresenta i registri generali e i nomi assegnati in base ai bit presi in considerazione.

Ogni registro, come mostrato nella figura B.2, può essere acceduto anche solo in parte: per esempio, per il registro EAX si può accedere solo ai 16 bit meno significativi e in questo caso il registro viene denominato solo AX (la E di EAX sta infatti per Extended). Il registro AX a sua volta può dare accesso solo agli 8 bit più significativi (i bit a sinistra) denominandosi AH, oppure solo agli 8 bit meno significativi (i bit a destra) denominandosi AL.

## Registri di segmento

Sono utilizzati per tenere traccia della posizione in memoria dello stack.

## Flag di stato

I flag di stato sono bit che servono a immagazzinare dei valori dovuti a risultati di operazioni aritmetiche e che possono essere utilizzati da altre istruzioni per prendere decisioni. C'è un solo registro che contiene tutti i bit di stato ed è chiamato **EFLAGS** ed è a 32 bit.

I bit contenuti in EFLAGS sono:

- **ZF**: o Zero Flag, è un bit che viene impostato a 1 quando il risultato di un'operazione è uguale a zero. Viene anche utilizzato per formulare istruzioni condizionali (equivalenti a if).
- **CF**: o Carry Flag, è un bit che viene impostato a 1 quando il risultato di un'operazione è troppo grande o troppo piccola per essere memorizzata nel registro di destinazione.
- **SF**: o Sign Flag, è un bit che viene impostato a 1 se il risultato di un'operazione è un numero negativo.
- **TF**: o Trap Flag, è un bit che viene impostato a 1 nei momenti in cui si eseguono operazioni di debug.
- ....

..tabella che mostra la distribuzione dei bit in EFLAG

## Instruction pointer

È un singolo registro che viene utilizzato per tenere traccia in memoria dell'indirizzo della prossima istruzione da eseguire.

## B.4 Istruzioni

Le istruzioni in linguaggio Assembly hanno la seguente forma: il nome mnemonico dell'operazione da eseguire seguito dagli operandi, ad esempio:

```
mov ecx 0x42
```

Gli **operandi** possono essere di tre tipi diversi:

- Valore immediato, ovvero una costante come 0x42;
- Il nome di un registro dove è presente un valore, come eax, ebx o ecx;
- Un valore presente ad una determinata locazione di memoria, per farlo si usa la notazione [eax] che significa il valore contenuto nell'area di memoria rappresentata da eax.

Qui di seguito è fornito un elenco delle istruzioni più utilizzato nel linguaggio Assembly.

### MOV

L'istruzione **MOV**, il cui nome deriva da move, è un'istruzione utilizzata per spostare valori da un registro ad un altro, oppure per copiare un valore in un registro. La sintassi di **MOV** è:

```
mov op2 op1
```

dove **op1** è un registro o un valore che viene copiato nel registro **op2**.

.. tabella esempi di mov

### LEA

L'istruzione **LEA**, acronimo di Load Effective Address, è un'istruzione simile a **MOV** ma con la differenza che in un registro viene copiato l'indirizzo di memoria di una allocazione e non il valore che si trova all'interno. La forma è:

```
lea reg1, [mem_addr]
```

dove **mem\_addr** è un indirizzo di memoria che verrà copiato nel registro **reg1**.

### ADD

È un'istruzione per effettuare l'addizione. Ha la forma:

```
add <dest> <src>
```

dove **<src>** è la sorgente dove andare a prendere il primo operando, che può essere un registro, un allocazione di memoria o un valore immediato, e **<src>**. Il risultato dell'addizione viene salvato nel registro di destinazione.

### SUB

Analogamente all'addizione, l'istruzione **SUB** compie la sottrazione tra il valore dell'operando sorgente e l'operando destinazione. Il risultato viene salvato nel registro di destinazione. Ha la forma:

```
sub <dest> <src>
```

Se il risultato è zero viene impostato a 1 lo Zero Flag.

## MUL

L'istruzione mul effettua la moltiplicazione e, a differenza di addizione e sottrazione, ha un unico operando. La forma è quindi:

```
mul <op>
```

con <op> un operando. La particolarità di questa istruzione è che utilizza un argomento implicito basato sulla grandezza dell'operando esplicito <op> come secondo fattore della moltiplicazione. Se l'operando esplicito ha una grandezza di 8 bit, quello implicito andrà a leggere il valore dentro un registro di uguale grandezza (ad esempio AL) e salverà il risultato in AX. Se invece l'operando esplicito è una parola di 16 bit si andrà a leggere come secondo fattore AX e si salverà il risultato in DX e AX. La tabella mostra un esempio del funzionamento.

..tabella

## DIV

L'istruzione div serve per effettuare la divisione ed ha lo stesso funzionamento dell'istruzione mul. Ha la forma:

```
div <op>
```

con <op> operando esplicito. Il quoziente della divisione viene salvato in AL, AX o EAX, mentre il resto in AH, DX o EDX. La tabella mostra il funzionamento della divisione.

..tabella

## AND

Effettua l'operazione di and e segue la tabella di verità ... Ha la forma:

```
and <dest> <src>
```

con <dest> operando di destinazione e <src> operando sorgente. L'operando di destinazione può essere solo un'area di memoria o un registro, mentre l'operando sorgente può essere anche un valore immediato.

## OR

Stesso funzionamento di and

```
or <dest> <src>
```

## XOR

È un'istruzione che effettua lo XOR logico ed ha la forma:

```
xor <dest> <src>
```

e viene spesso utilizzato per impostare a zero i valori di un registro.

## NOT

Ha solo un operando.

## SHL

SHL, che significa Shift Logical Left, è un'istruzione che effettua lo spostamento di un certo numero di bit verso sinistra. Ha due operandi e si può scrivere nella forma:

```
shl <op> <shift>
```

dove <op> può essere un registro o un area di memoria ed è il valore su cui verrà effettuato lo spostamento e salvato nello stesso registro o memoria, mentre <shift> può essere un registro di un byte o un valore immediato (sempre di 1 byte).

## SHR

SHR, che significa Shift Logical Right, è un'istruzione che effettua, come lo fa la SHL, lo spostamento di un certo numero di bit però verso destra. Ha due operandi e si può scrivere nella forma:

```
shr <op> <shift>
```

dove <op> può essere un registro o un area di memoria ed è il valore su cui verrà effettuato lo spostamento e salvato nello stesso registro o memoria, mentre <shift> può essere un registro di un byte o un valore immediato (sempre di 1 byte).

## NOP

NOP è un'istruzione che non fa nulla. In esadecimale ha valore 0x90.

### B.4.1 Istruzioni condizionali o roba simile

#### JMP

È un'Istruzione del controllo del flusso non condizionale, ovvero non verifica nessuna condizione prima di essere eseguita e salta all'istruzione dell'operando

#### JZ e JE

Sono istruzioni di controllo del flusso condizionali ed entrambe vengono eseguite se il bit di EFLAG Zero Flag (ZF) è uguale a 1. In quel caso saltano all'istruzione passata come operando.

#### JNZ e JNE

Sono istruzioni di controllo del flusso condizionali ed entrambe vengono eseguite se il ZF = 0

#### JL e JNGE

Se SF = 1

#### JLE e JNG

Se ZF = 1 o SF = 1

#### JGE e JNL

Se SF = 0

#### CMP

È un'istruzione che compara due operandi e imposta i valori dei bit di flag nel registro EFLAG in base al risultato in modo tale che possano essere acceduti da un'istruzione di jump. Ha forma:

```
cmp <dest> <src>
```

dove <dest> è l'operando di destinazione, mentre <src> è l'operando sorgente. In pratica viene sottratto l'operando sorgente all'operando destinazione e impostati i bit di flag, ma si distingue comunque dall'istruzione sub per il fatto che non salva il risultato dell'operazione nel operando <dest>. Modifica i bit CF, OF, SF, ZF, AF e PF. La tabella mostra un esempio di settaggi dei flag.

...tabella

## TEST

TEST è un'istruzione che, come cmp, effettua una comparazione dei suoi due operandi ma al contrario di cmp non effettua la sottrazione ma calcola l'AND logico e in base al risultato va a cambiare i bit di flag.

### B.4.2 Gestione dello stack

#### PUSH

Con l'istruzione push si va ad aggiungere un valore in cima allo stack della memoria. L'istruzione ha la forma:

```
push <source>
```

dove la sorgente <source> è il valore, immediato o di un registro, che sarà inserito nello stack. L'istruzione push, una volta eseguita, decrementa automaticamente lo Stack Pointer (ESP) di 4, per allinearla al nuovo valore.

#### POP

È l'operazione opposta alla push e va infatti a estrarre il valore che si trova in cima allo stack e lo ritorna come DWORD mettendolo in un registro e incrementando lo Stack Pointer ESP di 4.

#### CALL

L'istruzione call effettua una chiamata alla procedura rappresentata dal nome che viene passato.

#### RET

Istruzione di ritorno da una procedura....

## B.5 Codice C vs. codice Assembly

### B.5.1 Call Convention: stdcall

La convenzione prevede che la stdcall rappresenti la funzione function(1,2,3) come

```
push 3  
push 2  
push 1  
call function2
```

### B.5.2 Strutture condizionali IF e IF-ELSE

esempio.

### B.5.3 Struttura ciclica FOR

esempio.

### B.5.4 Struttura ciclica WHILE

esempio

## Visual Studio

Visual Studio può esserci molto utile per visualizzare il codice assembly generato da uno script in linguaggio C che abbiamo scritto.

Per prima cosa aprire Visual Studio e creare un nuovo progetto con **Create a new project > Empty Project** e assegnargli un nome di progetto desiderato, dopodiché cliccare su **Create**. Una volta apertosi il progetto vuoto dobbiamo creare un nuovo file in C dove scrivere il codice da visualizzare in Assembly;

per farlo cliccare con il tasto destro sopra **Source Files** nella finestra **Solution Explorer** e cliccare **Add > New Item**. Dopo aver fatto ciò si aprirà una finestra in sovrapposizione che permetterà di scegliere il tipo di file da creare e il nome da assegnarli: l'importante è che il file abbia estensione **.c**. Cliccare su **Add** per aggiungere il file al progetto, ora dovrebbe essere visualizzabile sotto **Source Files**.

## Impostazioni del compilatore

Questa parte è fondamentale per far sapere al compilatore C di Visual Studio che deve evitare di eseguire alcune operazioni mentre compila. Le operazioni in questione riguardano ottimizzazioni del codice che potrebbero mostrare, una volta che abbiamo ottenuto il codice Assembly, delle istruzioni aggiuntive o addirittura togliere delle istruzioni scritte dal programmatore perché considerate inutili. Altri esempi sono tecniche di modifica del codice utilizzate per prevenire il buffer overflow.

Per iniziare, cliccare con il tasto destro sul nome del proprio progetto nella finestra **Solution Explorer** e selezionare **Properties**. Si aprirà una finestra con un menù laterale e per ogni voce del menu una lista di opzioni. Cambiare le seguenti impostazioni come descritto qui di seguito:

- Su **C/C++ > General** impostare:
  - *Debug Information Format* con il valore: **Program Database (/ZI)**
  - *Support Just My Code Debugging* con il valore: **No**
- Su **C/C++ > Code Generation** impostare:
  - *Basic Runtime Checks* con il valore: **Default**
  - *Enable C++ Exceptions* con il valore: **No**
  - *Security Check* con il valore: **Disable Security Check (/GS-)**
- Su **C/C++ > Advanced** impostare:
  - *Compile As* con il valore: **Compile as C Code (/TC)**
- Su **Linker > General** impostare:
  - *Enable Incremental Linking* con il valore: **No (/INCREMENTAL:NO)**

Una volta cambiate le impostazioni cliccare su **Apply**.

## Visualizzazione del codice Assembly

Supponiamo di aver scritto all'interno del file **c** del progetto il seguente codice C:

```
int main(void)
{
    int a = 1;
    int b = 20;
    a = a + b;
    return 0;
}
```

Per visualizzare il codice Assembly corrispondente dobbiamo prima compilare il codice C e per farlo dobbiamo selezionare **Build > Build Solution**, dopodiché inseriamo un breakpoint sulla riga 3, ovvero la prima istruzione dentro il main, per avviare un debug e analizzare il codice Assembly riga per riga. La figura B.3 mostra la schermata del progetto con il file in linguaggio C e il breakpoint inserito.



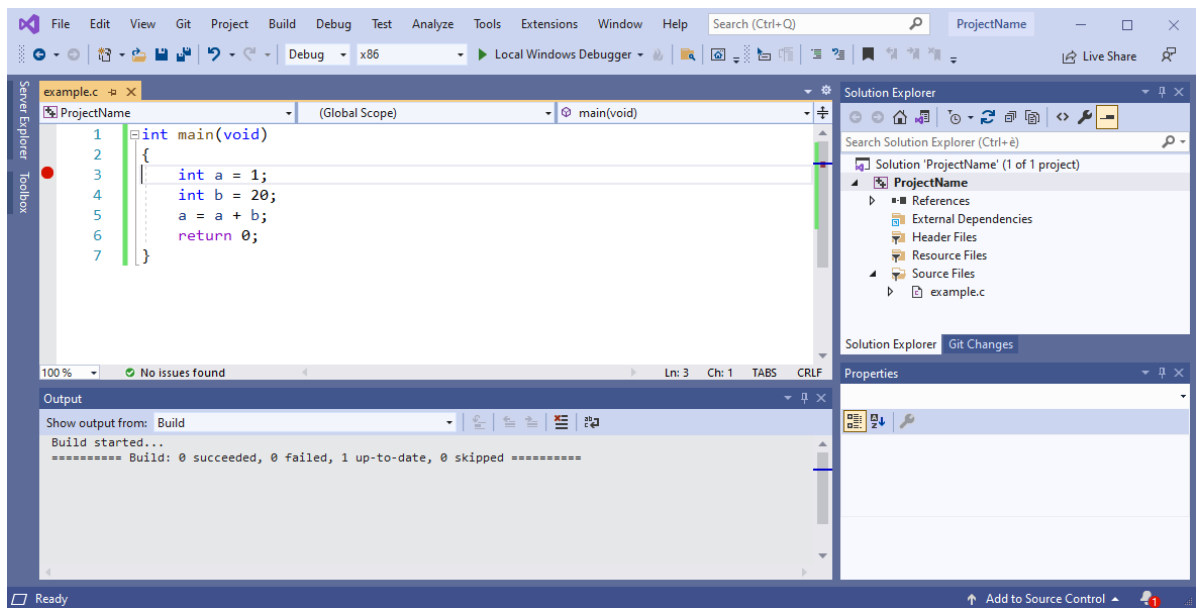


Figura B.3: Schermata di VisualStudio in cui è stato creato, in un progetto, il file `example.c`, contenente il codice C mostrato e compilato. È stato inoltre inserito un breakpoint alla riga 3.

Per avviare il debug premere **F5** oppure **Debug > Start Debugging**. Dopo aver avviato il debug è possibile passare al codice Assembly cliccando con il tasto destro nella finestra del codice e selezionando **Go to Disassembly**. Fatto questo sarà possibile eseguire il debug visualizzando le istruzioni in Assembly accanto a quelle in C. La figura B.4 mostra il codice in questione.

```

int main(void)
{
    00D81002  in          al,dx
    00D81003  sub          esp,8
        int a = 1;
    00D81006  mov         dword ptr [a],1
        int b = 20;
    00D8100D  mov         dword ptr [b],14h
        a = a + b;
    00D81014  mov         eax,dword ptr [a]
    00D81017  add         eax,dword ptr [b]
    00D8101A  mov         dword ptr [a],eax
        return 0;
    00D8101D  xor         eax,eax
}
    00D8101F  mov         esp,ebp
    00D81021  pop         ebp
    00D81022  ret

```

Figura B.4: Codice Assembly fornito da Visual Studio durante il debug.

In basso a sinistra nella finestra di Visual Studio comparirà una nuova finestra incorporata che mostra le schede **Autos** e **Locals**, le quali visualizzano rispettivamente i registri della CPU con i loro valori durante l'esecuzione e le variabili locali in C.

Altre due finestre molto utili che possono essere aggiunte cercandole nell'apposita barra di ricerca in alto sono: **Registers** e **Memory** <sup>1</sup>. La finestra **Registers** può essere cercata con **Debug > Windows > Registers** e rappresenta i valori dei registri della CPU durante l'esecuzione e viene mostrata dalla figura B.5.

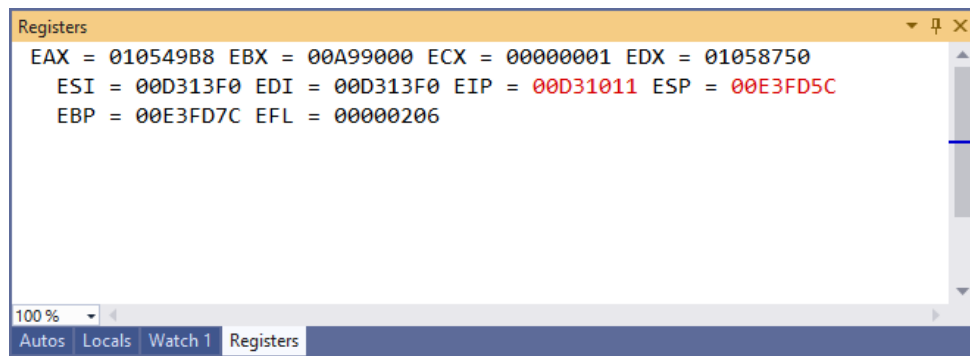


Figura B.5: Finestra che mostra il valore attuale dei registri della CPU, i valori in rosso sono appena stati riassegnati per esecuzione di una linea di codice all'interno del debug.

La finestra **Memory 1** invece può essere cercata con **Debug > Windows > Memory 1** e mostra il contenuto della memoria durante l'esecuzione. Si possono impostare le dimensioni dei byte, la codifica e si possono cercare i registri per nome e visualizzare la riga corrispondente con il valore. La figura B.6 mostra la finestra in questione.

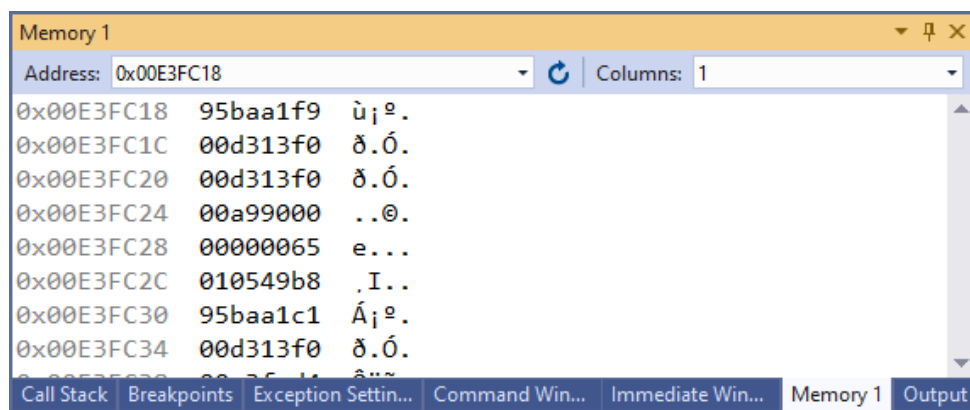


Figura B.6: Finestra che mostra il contenuto della memoria in byte e permette la ricerca tramite nomi dei registri.