POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

UN'ANALISI DEL TRADE-OFF PERFORMANCE-PRECISIONE NEL CLUSTERING DI MALWARE

VPLAB Lab
Laboratorio di Valutazione delle performance del Politecnico di Milano

Relatore: Prof. Stefano Zanero
Correlatore: Ing. Guido Salvaneschi

Tesi di Laurea di:
Maurizio Mollica, matricola 704944

Anno Accademico 2010-2011

A Gaia, senza la quale tutto questo sarebbe stato impossibile.

Sommario

Un algoritmo di clustering consente di suddividere una popolazione iniziale in gruppi di elementi aventi caratteristiche simili. Nell'ambito dell'analisi dinamica del malware, le caratteristiche sono delle attività svolte sul sistema operativo dal singolo campione di malware una volta eseguito. Per poter effettuare tale raggruppamento, è necessario confrontare le attività svolte da coppie di elementi e decidere se aggregare tali elementi o meno in un gruppo. Questa decisione viene presa in base a dei valori di similitudine, il cui calcolo può essere preciso o meno, che inficia dunque la composizione dei gruppi. Il confronto tra le caratteristiche di tutti i campioni risulta computazionalmente oneroso e pertanto sono stati sviluppati degli algoritmi che prediligono la riduzione del tempo di esecuzione rispetto ad un calcolo preciso dei valori di similitudine che sono dunque due componenti in trade off tra di loro. Un algoritmo efficiente risolve questo trade off concentrandosi sulla diminuzione del tempo di esecuzione a discapito della precisione.
L'obiettivo della tesi è lo studio e la quantificazione delle perdite, rispetto ai gruppi formati, derivate da questo tipo di scelta. Per raggiungere questo obiettivo, è stato costruito un algoritmo i cui requisiti siano il miglioramento della precisione nel calcolo dei valori di similitudine. Si vuole dimostrare che la mancanza di precisione si riversa sui gruppi formati mediante l'aumento del numero di falsi negativi, l'aumento del numero di gruppi prodotti per varianti polimorfiche dello stesso malware, variazioni di precisione e richiamo. Queste sono le metriche che verranno utilizzate per confrontare l'algoritmo appositamente sviluppato nell'elaborato rispetto all'algoritmo di clustering efficiente presentato in [65] rispetto alla stessa popolazione iniziale.

Ringraziamenti

Ringrazio in modo particolare il Prof. Stefano Zanero per i preziosi insegnamenti durante i due anni di laurea specialistica e l'Ing Guido Salvaneschi per aver supervisionato il lavoro di tesi, ed entrambi per aver dispensato preziosi consigli e per essermi stati di supporto. Ringrazio inoltre il gruppo VPLAB, fonte di competenze tecniche di grande aiuto per il lavoro svolto. Ringrazio infine i miei genitori per avermi dato supporto morale ed economico in questi anni di studi, ed i miei fratelli per essermi stati accanto in questo periodo.

Indice

Sommario **I**

Ringraziamenti **III**

1 Introduzione **5**
1.1 Inquadramento generale 5
1.2 Descrizione del problema 7
1.3 Descrizione del lavoro 9
1.4 Metodologie utilizzate 10
1.5 Struttura della tesi 11

2 Stato dell'arte **13**
2.1 Stato dell'arte nell'analisi di malware 13
2.2 Algoritmi di clustering di malware 17
2.3 Critiche e studi riguardanti gli algoritmi di clustering 20
2.4 Anubis 21
2.4.1 Punti di forza di Anubis 23
2.4.2 Problematiche di Anubis 24

3 Impostazione del problema di ricerca **27**
3.1 Definizioni preliminari 27
3.1.1 Parametri di precisione e di richiamo 27
3.1.2 Decomposizione ai valori singolari 29
3.1.3 Misure di distanza 30
3.2 Problematiche affrontate e metodologie utilizzate 31
3.2.1 Scelta del malware da analizzare 31
3.2.2 Scelta di una ground truth 32
3.2.3 Scelta di una misura di distanza per il calcolo dei confronti tra le coppie di malware 33
3.2.4 Scelta degli elementi estrapolati dai report comportamentali da utilizzare 33

3.2.5 Scelta dell'algoritmo di clustering gerarchico 34

4 Progetto logico ed architettura del sistema 37
4.1 Introduzione all'analisi semantica latente 37
4.2 Traduzione dei concetti dell'analisi semantica latente nell'ambito del clustering di malware 39
4.3 Motivazione dell'utilizzo dell'analisi semantica latente nello studio 41

5 Implementazione 43
5.1 Elementi di implementazione 43
5.1.1 Parsing dei report XML di Anubis 43
5.1.2 Dettagli implementativi algoritmo LSH 44
5.1.3 Dettagli implementativi algoritmo LSA 46
5.1.4 Algoritmo di calcolo delle distanze senza approssimazioni 47
5.2 Considerazioni sugli algoritmi di clustering gerarchico 48

6 Realizzazioni sperimentali e valutazione 51
6.1 Dataset coinvolto nello studio 52
6.2 Performance temporale asintotica 52
6.2.1 Preprocessamento della matrice caratteristiche-malware 53
6.2.2 Calcolo delle misure di distanza tra tutti i campioni . 53
6.2.3 Clustering 54
6.3 Valutazioni circa i cluster prodotti 54
6.3.1 Numero di falsi negativi 56
6.3.2 Precisione e Richiamo 57
6.3.3 Consistenza 57
6.3.4 Dendrogramma LSH 60
6.3.5 Dendrogramma LSA 62
6.3.6 Dendrogramma SC 64

7 Direzioni future di ricerca e conclusioni 67
7.1 Conclusioni 67
7.1.1 Lavoro svolto 67
7.1.2 Deduzioni derivate dallo studio 68
7.2 Sviluppi futuri 69

Bibliografia 72

A Il manuale utente 79

Elenco delle figure

6.1 Metriche di precisione e richiamo in funzione di t 56
6.2 Dendogramma LSH 60
6.3 Dendogramma LSA 62
6.4 Dendogramma SC 64

Elenco delle tabelle

6.1 Performance temporale asintotica 54
6.2 Valori del parametro t scelti 57
6.3 Flat Cluster a livello di distanza $t = 0.3$ per LSH, $t = 0.35$ per LSA e $t = 0.3$ per SC 58

Capitolo 1

Introduzione

"Un mondo senza gli hacker sarebbe un mondo senza curiosità e innovazione."

Jon Erickson

1.1 Inquadramento generale

Nelle reti di calcolatori e nei sistemi informativi in generale, il malware è diventato un problema onnipresente e difficile da risolvere. Il codice dannoso può essere eseguito sui sistemi utilizzando una varietà di metodi, come ad esempio attacchi contro debolezze conosciute del software, funzionalità nascoste in programmi utili e l'ingegneria sociale [19]. Una *istanza di malware* è un programma che possiede funzionalità dannose. Esempi di questi programmi includono:

1. *Virus*: programmi che si auto replicano all'interno di un sistema incorporandosi in programmi o documenti che diventano i vettori del codice dannoso,

2. *Worms*: programmi complessi la cui prerogativa è la diffusione attraverso la rete,

3. *Cavalli di troia*: si mascherano da programmi utili, ma contengono codice dannoso che attacca il sistema o sottrae informazioni sensibili,

4. *Back doors*: aprono il sistema ad entità esterne sovvertendo le politiche locali di sicurezza al fine di permetterne l'accesso remoto ed il controllo in rete,

5. *Spyware*: pacchetti software che eseguono varie utilità ma nel contempo trasmettono dati privati dell'utente ad una entità esterna.

Gli eseguibili dannosi non ricadono sempre precisamente in queste categorie e possono esibire differenti combinazioni di questi comportamenti. Un *identificatore di malware* è un sistema che tenta di identificare il malware. Il fine ultimo di un programmatore di malware è di sottrarre informazioni sensibili, guadagnare accesso non autorizzato alle risorse di sistema, inibire l'operatività del sistema ed altri comportamenti abusivi. Per ottenere questo scopo, i programmatori modificano o trasformano i loro programmi per evadere il rilevamento da parte degli identificatori di malware. Un antivirus commerciale utilizza delle firme, ovvero delle "impronte" presenti nell'eseguibile del malware, e di conseguenza è un esempio di identificatore di malware che conduce una tipologia di analisi di natura statica poichè il campione analizzato non viene eseguito durante l'analisi. Gli antivirus ed altri identificatori di malware statici osservano dunque solo sequenze di istruzioni, non preoccupandosi di ciò che queste istruzioni fanno. Una firma è quindi una sequenza di istruzioni presente all'interno dell'eseguibile del malware che identifica il particolare campione e che viene rilevata mediante un'espressione regolare.
Tra i vari metodi utilizzati per evadere l'analisi da parte di un identificatore di malware citiamo: l'offuscamento di codice, il polimorfismo, il metamorfismo [19] e l'anti-debugging per quanto riguarda l'analisi condotta manualmente dall'analista [22]. Usando queste tecniche, è possibile generare un gran numero di varianti di malware [14]. Ad esempio un virus può trasformarsi crittando il suo payload dannoso e decrittandolo durante l'esecuzione.
Un virus polimorfico offusca il suo ciclo di decrittazione utilizzando diverse trasformazioni, come ad esempio l'inserimento di *nop*, la trasposizione del codice, ovvero il cambiamento dell'ordine delle istruzioni attraverso l'inserzione di *jump* al fine di mantenerne la semantica originale, e la permutazione dell'allocazione dei registri. Importante notare dunque che tutte le varianti polimorfiche di uno stesso malware, forniscono la stessa funzionalità e mostrano un comportamento simile pur avendo differenti rappresentazioni sintattiche [14].
I virus metamorfici cercano di evadere il rilevamento offuscando il virus intero; quando si replicano, questi virus cambiano il loro codice in una varietà di modi, come ad esempio la trasposizione del codice e la sostituzione di sequenze di istruzioni equivalenti, il cambiamento dei loop condizionati ed il reassignamento dei registri. La trasposizione del codice mescola le istruzioni in maniera tale per cui l'ordine di esse nell'immagine binaria risulta

differente dall'ordine di esecuzione o dall'ordine delle istruzioni assunto dalla firma usata per la rilevazione. Per far questo è sufficiente inserire delle *jump* non condizionali per ripristinare l'ordine originario delle istruzioni. La reassignazione dei registri sostituisce l'utilizzo di un registro con un altro in uno specifico intervallo di vita. Ad esempio, se il registro *ebx* è inutilizzato durante l'intervallo di vita del registro *eax*, esso può essere utilizzato per rimpiazzare il registro *eax*. Le tecniche di sostituzione delle istruzioni infine, consentono di sfruttare la varietà dell'instruction set del processore (ad esempio IA-32) in modo tale da cambiare sequenze di istruzioni con delle equivalenti dal punto di vista della funzionalità [19]. Il fine è pertanto l'evasione del rilevamento del malware basato su firme.
Le tecniche di antidebugging comprendono infine: l'utilizzo di file malformati, tecniche specifiche di individuazione del debugger in uso, tecniche che sfruttano bug noti di specifici debugger. Si possono specificare ad esempio delle opzioni del formato PE (il formato dei file eseguibili da Windows NT in poi) che simulano la corruzione del file e quindi l'impossibilità di renderlo analizzabile da parte di un debugger: si può ad esempio settare il campo dell'header $PE \rightarrow NumberOfSections$ a 0 simulando pertanto il fatto che l'eseguibile non contenga alcuna sezione. Un altro esempio è l'utilizzo del valore 0 come entry point dell'eseguibile, il che consente l'esecuzione dell'header MZ da parte del debugger *Immunity Debugger* causando la corruzione della memoria. Per ultimo, un debugger può essere individuato facilmente con l'utilizzo della funzione *FindWindow()* passando come parametro di ricerca il nome del debugger di cui si sta testando la presenza [22].
Una delle principali problematiche degli analizzatori statici è che il rilevamento mediante analisi statica non è robusto rispetto alle trasformazioni del codice che abbiamo descritto. Anche piccole variazioni nella forma sono già sufficienti ad evadere questa tipologia di analisi e questo è il principale motivo per cui le firme degli antivirus commerciali devono essere costantemente aggiornate per tenere il passo con le modifiche apportate nelle varianti di malware.

1.2 Descrizione del problema

In ambito di ricerca, è stato proposto un altro metodo di analisi che supera i limiti dell'analisi statica: l'analisi dinamica di malware. Questa metodologia, a differenza dell'analisi puramente statica, prevede l'esecuzione del campione in ambiente monitorato (chiamato sandbox). L'ambiente monitorato può essere una macchina virtuale o fisica ma quasi sempre la scelta ricade sui sistemi di virtualizzazione per poter ritornare velocemente ad uno

stato del sistema operativo, precedente l'esecuzione del malware, prima di poter analizzare un nuovo campione. Dell'esecuzione del campione nell'ambiente virtualizzato, vengono tracciate le varie attività eseguite sul sistema operativo e tipicamente ne viene generato un report.
Il clustering è una tecnica di analisi mutivariata dei dati che è possibile utilizzare in congiunzione con l'analisi dinamica del malware. Questa metologia prevede la suddivisione della popolazione di malware in sottoinsiemi aventi caratteristiche funzionali simili. I report comportamentali generati dalla sandbox vengono filtrati al fine di eliminare informazione spuria e rindondate per poter semplificare il più possibile la trattazione analitica di essi. Il risultato è un'insieme di attività eseguite dal singolo campione di malware. Si rappresenta dunque il singolo campione di malware m_i come un insieme delle attività svolte: $m_i = a_1, \ldots, a_m$. Il clustering prevede il calcolo delle similitudini tra le attività svolte dai vari campioni di malware mediante una funzione analitica $g(m_1, m_2)$, che lavora tra coppie di malware, chiamata funzione di distanza (da cui si ricava la similitudine come $f(m_1, m_2) = 1 - g(m_1, m_2)$). Tale funzione confronta l'insieme delle attività svolte dai due campioni e restituisce un valore reale che indica la distanza tra i due campioni in un determinato spazio. La funzione viene applicata a tutte le coppie di campioni di malware analizzati. Quest'attività risulta molto onerosa in termini prestazionali dal momento che il numero di campioni da analizzare è molto spesso dell'ordine del milione. Sono stati pertanto implementati algoritmi che effettuano delle approssimazioni per rendere la computazione fattibile in tempi accettabili. L'algoritmo presentato in [65] ad esempio lavora su funzioni di hash dell'insieme di tutte le attività svolte dal malware al fine di determinare una firma delle attività che sia concisa al punto da permettere un calcolo agevole delle distanze che viene effettuato sulle firme hash anzichè sull'informazione completa delle attività svolte dal malware. L'algoritmo di clustering si occupa infine di formare dei raggruppamenti, in base a determinate politiche, utilizzando le metriche di distanza calcolate. Il fatto di aver utilizzato un'approssimazione nel calcolo delle misure di distanza, fa sì che sia presente un degrado in questa ultima fase dell'analisi che assume forti ripercussioni sui risultati da essa prodotta.
Lo scopo della tesi è lo studio e la quantificazione, secondo determinate metriche, degli aspetti che in un algoritmo efficiente hanno una ricaduta sui gruppi formati nella fase di clustering. Nei paragrafi successivi si introduce brevemente il lavoro svolto e si spiegano le metodologie utilizzate in questo studio.

1.3 Descrizione del lavoro

La ricerca inizia con la descrizione delle metriche che si adotteranno per valutare le ricadute sui gruppi formati in fase di clustering conseguenti l'utilizzo di un algoritmo di calcolo delle distanze efficiente. L'aspetto più evidente nelle approssimazioni effettuate da questo tipo di algoritmi è che è possibile per un determinato numero di campioni avere dei valori di distanza molto più bassi rispetto al valore che si otterrebbe considerando la totalità delle attività del campione anzichè un valore compresso, ad esempio un valore in hash, di esse. Questo si traduce di conseguenza nel mancato rilevamento di malware ed il tasso che lo misura è definito come numero di falsi negativi.
Il secondo aspetto, non meno rilevante, è che l'imprecisione nel calcolo delle distanze si traduce anche in una eccessiva segmentazione del cluster che dovrebbe contenere tutte le varianti di un unico malware. Questo avviene perchè nel calcolo della funzione di hash, nel caso dell'algoritmo in [65], si perde informazione significativa dal momento che non si effettua nessuna ottimizzazione inerentemente al calcolo della funzione di hash. Questi due aspetti sono l'oggetto di studio della tesi. Per poterli analizzare dettagliatamente, si sono sviluppati due algoritmi: un primo algoritmo che effettua approssimazioni differenti nel calcolo delle misure di distanza ed un algoritmo che non effettua nessuna approssimazione. Il fine è di confrontare come la scelta tra i tre algoritmi impatti sui gruppi formati a parità di funzione di distanza utilizzata (funzione coseno) in base alle seguenti metriche:

1. Numero di falsi negativi,
2. Numero di gruppi formati contenenti varianti polimorfiche dello stesso malware (consistenza),
3. Metrica di precisione utilizzata in information retrieval,
4. Metrica di richiamo utilizzata in information retrieval.

Si vuole ottenere poi che i 2 algoritmi sviluppati soddisfino i seguenti requisiti:

1. Riuscire a clusterizzare in poche famiglie il maggior numero di varianti polimorfiche dello stesso malware,
2. Ridurre la percentuale di falsi negativi del processo di clusterizzazione.

Il terzo algoritmo deve inoltre confrontare i campioni di malware senza effettuare approssimazioni che diminuiscano il contenuto informativo della

totalità delle attività. Il paragone diretto con l'algoritmo in [65] secondo le metriche di cui sopra, porterà alla luce gli aspetti che si vogliono studiare e se ne potrà ricavare anche una misura precisa. Il lavoro si concluderà con la presentazione dei risultati ottenuti dal confronto tra i diversi algoritmi.

1.4 Metodologie utilizzate

Dopo aver descritto brevemente il lavoro, si passa a delineare le metodologie utilizzate. Per effettuare un corretto confronto, si è deciso di utilizzare la stessa ground truth, ovvero la conoscenza a priori di quante siano le famiglie comportamentali finali a cui debbano appartenere i campioni analizzati.
Utilizzando la stessa metodologia proposta in [65], si sono selezionati mille campioni di malware che hanno ottenuto un alto ranking in Virus Total [12], ovvero campioni che siano stati etichettati in maniera congruente dalla maggior parte dei 42 antivirus commerciali utilizzati nell'analisi statica degli eseguibili. L'unica differenza, rispetto alla selezione della ground truth effettuata in [65], consiste nella distribuzione del numero di malware per classe, la cui motivazione verrà spiegata nel capitolo tre. Tali campioni sono stati sottoposti alla sandbox Anubis [1], utilizzata anche in [65] per la generazione dei report comportamentali. I mille campioni di malware sono stati poi sottoposti ai tre algoritmi per il calcolo della matrice delle distanze, le quali sono state in seguito sottoposte ad algoritmi di clustering gerarchico sui cui gruppi formati sono state rilevate infine le metriche analizzate. Per ottenere questo risultato, si è utilizzata la funzione coseno per il calcolo delle misure di distanza, a partire da input differenti generati dai tre algoritmi. La valutazione degli algoritmi è stata quindi effettuata in base all'effettiva mappatura tra i cluster rilevati dall'analisi statica e quelli rilevati dai metodi proposti per quanto concerne la segmentazione delle famiglie comportamentali, in base ad una misura diretta del numero dei falsi negativi sul dataset utilizzato ed in base alle metriche classiche di precisione e richiamo utilizzate in information retrieval per quanto riguarda la qualità del cluster prodotto. L'algoritmo sviluppato in [65] ha ottenuto valori inferiori rispetto alle 4 metriche analizzate. I restanti due algoritmi si sono comportati in maniera del tutto simile denotando innanzitutto che si guadagna in numero di falsi negativi ed in percentuale di varianti polimorfiche individuate scegliendo un'approssimazione sempre meno grossolana. L'approssimazione effettuata nel secondo algoritmo, l'algoritmo che chiameremo LSA (latent semantic analysis) nel seguito, permette di mantenere l'informazione che genera la maggior varianza nei dati ed è quindi un'approssimazione frutto di una scelta del trade-off tra tempo di esecuzione e precisione di calcolo delle distanze

che privilegia maggiormente la seconda componente. Il terzo algoritmo infine non esegue nessuna approssimazione e quindi utilizza interamente l'informazione sulle attività denotando una scelta esclusiva della seconda componente del trade-off.
I risultati ottenuti permettono di quantificare precisamente, secondo queste metriche, le perdite derivate dall'utilizzo di un algoritmo efficiente. Di conseguenza, se si utilizza un algoritmo molto scalabile, necessariamente la qualità del cluster ne risulta inficiata rispetto ai parametri di analisi, mentre se si vuole ottenere una qualità del cluster il più possibile elevata, bisogna comunque tenere in considerazione la totalità del contenuto informativo presente nei report comportamentali generati dalla sandbox. Lo studio ha permesso quindi di mettere alla luce ed esprimere in maniera precisa gli aspetti che cambiano significativamente in base a diverse scelte del trade-off.

1.5 Struttura della tesi

La tesi è strutturata nel modo seguente.
Nel capitolo 2 si mostrano gli obiettivi raggiunti in ambito di ricerca nel settore relativo all'analisi del malware, e si espongono le motivazioni che hanno spinto alla tesi.
Nel capitolo 3 si illustrano gli obiettivi della ricerca, le problematiche affrontate ed alcune definizioni preliminari.
Nel capitolo 4 si descrive come è stato affrontato il problema concettualmente e la soluzione che ne è scaturita.
Nel capitolo 5 si presenta la scomposizione dei vari moduli che compongono il progetto.
Nel capitolo 6 viene presentata la valutazione in termini di performance del sistema e le attività sperimentali svolte.
Nel capitolo 7 si riassumono gli scopi, si presentano conclusioni riassuntive e le prospettive future.
Nell'appendice A mostriamo i comandi con relativi parametri da effettuare per il lancio degli script Python, implementazione del sistema.
Nell'appendice B si riportano alcune parti significative del codice sorgente commentate.

Capitolo 2

Stato dell'arte

"Il calcolatore è straordinariamente veloce, accurato e stupido. Gli uomini sono incredibilmente lenti, imprecisi e creativi. L'insieme dei due costituisce una forza incalcolabile."

Albert Einstein

Nel primo capitolo si sono presentate in modo generale alcune tipologie di analisi di malware che in questo capitolo verranno riprese e approfondite. Si presenteranno diverse tecniche di analisi ponendo particolare attenzione agli algoritmi di clustering di malware che costituiscono l'ambito preciso di cui lo studio si occupa; si presenta inoltre la sandbox che si è deciso di utilizzare per la generazione dei report dell'attività del malware, Anubis.

2.1 Stato dell'arte nell'analisi di malware

Nel primo capitolo si è detto che l'analisi di tipo statico può essere evasa mediante semplici metodologie di offuscamento del codice. La problematica di base dell'approccio di pattern matching all'individuazione di malware è infatti la mancata conoscenza di ciò che effettivamente fanno le istruzioni una volta realmente eseguite. Le tecniche di metamorfismo dimostrano che i programmi con funzionalità simili non devono condividere necessariamente istruzioni e sequenze di dati simili. Di conseguenza, in ambito di ricerca sul malware, si ripone l'attenzione su caratteristiche strutturali di alto livello quali ad esempio i grafi di flusso di controllo, estraibili a partire da un disassemblatore.
L'approccio semantico [20, 48] è un'altra tecnica di analisi statica di malware in cui si presta attenzione alla semantica delle istruzioni piuttosto che alla sintassi. Infatti, nell'approccio statico, viene presa una sequenza di

istruzioni come firma del campione di malware. Per ottenere un mancato riconoscimento di questa firma da parte di un analizzatore statico, è sufficiente introdurre una sequenza di istruzioni che non alteri il funzionamento del malware. Questo è l'esempio più semplice di offuscamento del codice. Un identificatore semantico di malware prende in input un modello di codice dannoso (ad esempio il codice di un preciso malware) e cerca di individuare nel campione analizzato la presenza di sottoinsiemi di istruzioni del modello in input. Questa strategia permette di rilevare esempi semplici di offuscamento ma non riesce ad individuare trasformazioni più complesse.
Uno studio recente [40] dimostra infatti che l'analisi statica nel complesso, usata al fine di identificare il malware, può essere facilmente evasa attraverso delle semplici tecniche di offuscamento che sono peraltro già utilizzate dagli scrittori di malware. In tale studio, viene introdotta una tipologia di metamorfismo in grado di ingannare anche gli identificatori di malware di natura semantica, il metodo delle *costanti opache*, che consente di codificare un problema NP-completo che crei una sequenza di istruzioni che calcoli una costante utilizzata dal programma. Il problema codificato è quello della 3-soddisfacibilità booleana (nel seguito 3SAT), che è stato dimostrato essere NP-completo. L'idea che sta alla base della tecnica, è di creare una variabile mediante una sequenza di istruzioni che, se dovesse essere analizzata staticamente per predirne il risultato, si ricondorrubbe a dover risolvere il problema 3SAT; quindi il problema 3SAT è riducibile in tempo polinomiale al problema dell'analisi statica esatta del valore della costante opaca, che quindi risulterà almeno NP-difficile. La risoluzione deve determinare se esiste una combinazione di variabili che possa soddisfare una particolare formula booleana. Lo schema generale di calcolo della costante opaca prevede la generazione di valori random delle variabili che fanno parte della formula da verificare. Viene poi valutata la formula e, se la formula è soddisfatta, una costante (la costante opaca) assumerà un dato valore, altrimenti resterà immutata. Dato che per la maggior parte degli assegnamenti delle variabili che compongono la formula la renderanno insoddisfacibile, si rende necessario introdurre appositamente dei valori delle variabili che talvolta la rendano vera. Per ottenere questa funzionalità, si prevede la presenza di una catena di blocchi basici (sequenza di istruzioni che non prevede *jump* condizionali). Le variabili che compongono la formula sono realizzate come variabili globali. I valori di tali variabili possono essere poi settate, alla fine di ogni blocco basico, o mediante valori statici random, o valori random generati a runtime, oppure mediante valori generati a runtime che validino la formula del prossimo blocco basico. Per analizzare un programma che è offuscato in questo modo, l'analizzatore non può fare affidamento sul fatto che la formula

sia sempre insoddisfacibile. In questo modo si riescono a caricare variabili nei registri in maniera difficilmente individuabile da un analizzatore statico. La conclusione derivata in [40] è che l'analisi statica del codice è un problema almeno NP-difficile.
Nell'analisi comportamentale, una proposta di soluzione al problema dell'analisi di codice dannoso che supera i limiti dell'analisi statica, il malware è visto come una scatola chiusa e sono analizzati solo i suoi effetti sul sistema, il suo comportamento [30]. Esistono diversi sistemi infatti che monitorano complessivamente tutte le operazioni svolte dal malware, tra cui citiamo CWSandbox [4], BitBlaze [2], Norman [6], ThreatExpert [11] ed Anubis [1]. Eseguendo un campione di malware in un ambiente controllato e tracciandone le attività svolte sul sistema operativo, è possibile creare un report di analisi che riassuma il comportamento osservato del campione.
Per far fronte al numero crescente e alla diversità del malware, l'analisi dinamica può essere combinata con algoritmi di clustering [14,16,26,29,46,53,65] e di classificazione [50, 55, 60], due tecniche di machine learning rispettivamente non supervisionata e supervisionata [44]. Un algoritmo di clustering comportamentale aiuta a trovare nuove famiglie di malware, mentre la classificazione di malware assegna malware sconosciuto a famiglie sconosciute [30]. La classificazione del malware filtra campioni sconosciuti e perciò riduce i costi di analisi. La classificazione fatta a livello di identificatori di virus commerciali, fa capo a classificatori euristici che sono generati da un gruppo di esperti di virus che individuano nuovi malware. Questo tipo di analisi consuma una quantità notevole di tempo e spesso ancora fallisce nell'individuare nuovi eseguibili dannosi [55]. Da qui la necessità di dedurre automaticamente caratteristiche dal comportamento osservato che sono essenziali per l'individuazione e la categorizzazione del malware. Queste caratteristiche possono essere usate per aggiornare le firme o come input per l'aggiustamento delle regole euristiche degli identificatori di virus commerciali [50].
Il vantaggio immediato di questa analisi, rispetto a quella statica, è la trattazione di famiglie comportamentali di malware. Vale a dire che tutti i campioni che sono varianti polimorfiche ognuno dell'altro, sono trattati come membri della stessa famiglia comportamentale. Al fine di sviluppare tecniche successive che facciano utilizzo di questa idea, è necessario produrre un metodo automatico che la sfrutti, raggruppando i campioni, in maniera appropriata, nelle rispettive famiglie comportamentali. Tuttavia, ottenere una corrispondenza esatta tra le varianti polimorfiche di un malware ed una famiglia comportamentale, è un problema difficile. Possedere un metodo che identifichi automaticamente la famiglia comportamentale di un determinato campione, permette la creazione di una firma relativa ad una famiglia, venen-

do meno alle problematiche delle firme derivate dall'analisi statica, che sono inerentemente suscettibili ad inaccuratezze [26]. Una firma siffatta, individua tutti i campioni dell'intera famiglia (e, nel caso ideale, nessun campione di qualsiasi altra famiglia) ed aiuterebbe significativamente a ridurre il quantitativo globale di firme richieste. Il mappaggio tra campioni di malware basato su alcune caratteristiche e famiglie aventi comportamento simile, è un tipico scenario applicativo dei metodi di clustering.
Un'altra problematica delle tecniche di analisi dinamica è rappresentata dall'innesco del percorso di esecuzione dannoso da parte del programma sotto osservazione (problema dell'esecuzione multipath), dove per innesco si intende il fornire un determinato input o condizione al programma che permetta l'esecuzione di un segmento di codice che altrimenti, in assenza dell'input o della condizione, non verrebbe eseguito. Nell'analisi in ambiente monitorato infatti, non esiste alcuna interazione con l'esecuzione del campione nella sandbox. Il campione viene infatti fatto eseguire sino allo scadere di un timeout di esecuzione impostato dalla sandbox utilizzata per l'analisi. Ne deriva necessariamente che, per i campioni che necessitano di un qualche input o condizione per manifestare il comportamento dannoso, non si ottiene un inquadramento globale delle attività svolte, ed in particolare possono risultare anche soltanto comportamenti del tutto innoqui. Per fare qualche esempio, un malware può attivarsi solo in una determinata data, può ricercare un dato file, mutex, chiave di registro, può necessitare di una qualche forma di input da parte dell'utente. In assenza di questi elementi, il programma terminerebbe l'esecuzione o semplicemente eseguirebbe un'altra serie di istruzioni rispetto a quelle che ci interessa monitorare. Un tentativo di risolvere il problema dell'esecuzione multipath è presentato in [39], dove si propone di monitorare come il codice usi certi input; più precisamente, si tiene traccia dinamicamente di certi valori in input che il programma legge (come ad esempio il contenuto di un file, l'ora di sistema, il risultato di un test di connettività internet) e si identificano punti nell'esecuzione dove questi input sono utilizzati al fine di prendere decisioni di controllo di flusso. Viene eseguito un salvataggio dello stato della macchina prima che la decisione venga presa in modo da poter forzare l'esecuzione dei path multipli cambiando l'input al momento immediatamente precedente il punto decisionale. Risulta quindi evidente che l'ambito di ricerca nel settore è molto attivo e vengono proposti molti tentativi di risoluzione delle problematiche ancora irrisolte e metodologie di analisi innovative.

2.2 Algoritmi di clustering di malware

Un algoritmo di clustering di malware si pone l'obiettivo di costruire famiglie comportamentali, partendo da un insieme di malware raccolti mediante vari metodi. Esponiamo dunque l'insieme di passi che questi algoritmi hanno in comune, per poi spiegare come questi passi sono stati implementati nel singolo algoritmo.
Il primo passo nell'analisi di malware in generale, è la raccolta di nuovi campioni di malware. Strumenti di raccolta di malware esistenti, si affidano ad honeypots, spam traps, ed altre tecniche passive. Tuttavia, queste tecniche sono lente, dal momento che è necessario attendere fino al momento in cui un nuovo malware comincia a propagarsi su larga scala prima di poter collezionare una copia di esso [45]. Ad esempio, dal momento in cui un nuovo malware colpisce una honeypot, esso dovrebbe già aver infettato un numero elevato di macchine. La ricerca attiva di malware d'altra parte focalizza l'attenzione su un numero limitato di URL ritenute sospette e responsabili di essere vettori di download di malware. Un'alternativa è il crawling del web e delle reti P2P e questa tecnica si mostra particolarmente efficace per quelle tipologie di malware che adottano le reti P2P per la diffusione. Per ultimo, porre uno sniffer degli eseguibili scaricati dalle utenze delle rete, è un ulteriore metodo attivo che colleziona però anche un gran numero di eseguibili benigni che quindi dovrebbero subire una fase di filtraggio dispendiosa. Quest'ultimo problema è preso in considerazione in [45].
Successivamente alla raccolta di campioni, si procede all'esecuzione degli stessi in un ambiente monitorato, la sandbox, attraverso la quale il campione viene fatto girare in memoria e ne vengono tracciate le azioni svolte sul sistema operativo. Vengono poi memorizzati in un report di analisi i tratti che riflettono pattern comportamentali (ad es. apertura di un file, lock di un mutex, settaggio di una chiave di registro). L'insieme di tutti i report costituisce l'input dell'algoritmo utilizzato per valutare la similitudine o differenziazione delle varie attività rilevate per ogni coppia di malware.
L'output di questa fase è una matrice quadrata simmetrica contenente i valori delle distanze tra tutte le coppie di malware; tale valore è generalmente normalizzato per assumere valori reali compresi tra 0 e 1. La matrice quadrata così costituita, è data in input ad un algoritmo di clustering che si occupa di generare sottoinsiemi di malware aventi comportamento simile, generando così il sottoinsieme dell'insieme di tutti i campioni di malware che deriva le varie famiglie comportamentali. Per valutare l'efficacia degli algoritmi di clustering di malware, è necessario avere a disposizione una ground truth, cioè la conoscenza a priori del risultato ottimo che il classificatore implementato

dovrebbe darci una volta eseguito. Nel caso vengano utilizzati più antivirus, la ground truth risulta sicuramente più robusta. Tuttavia, dal momento che la concordanza tra più antivirus si traduce necessariamente in una diminuzione del numero di campioni di malware che possono essere utilizzati come ground truth, si presenta la problematica che tali campioni siano effettivamente facili da analizzare [44].
Nel primo algoritmo di clustering di cui discutiamo, presentato in [16], i campioni vengono eseguiti in una macchina virtuale con Microsoft Windows XP installato. I report sono generati mediante Backtracker ed esportati ad un server esterno. Dopo la raccolta dati, vengono calcolate le distanze tra i vari report comportamentali utilizzando la misura di distanza *Normalized Compression Distance*. Questa scelta è seguita all'utilizzo della *Edit Distance* per due motivi:

1. Una enfatizzazione delle differenze rispetto alle similitudini, dal momento che quando la dimensione dei campioni da analizzare è grande, la edit distance è equivalente ad un clustering dei campioni basato sulla dimensione dell'insieme totale di comportamenti,

2. Un semplice polimorfismo comportamentale, ad esempio nomi di file random, che rendono i cambiamenti di stato del malware differenti a parità di reale operazione.

L'utilizzo della Normalized Compression Distance tenta di risolvere questi problemi fornendo un'approssimazione del contenuto informativo nel senso che, intuitivamente, fornisce la sovrapposizione dell'informazione tra due campioni; come risultato, comportamenti che sono simili ma non identici, sono visti come vicini. Infine, una volta calcolata la matrice delle distanze tra tutti i campioni di malware, si utilizza l'algoritmo di clustering single linkage per la suddivisione in famiglia vera e propria.
L'algoritmo proposto in [65] utilizza la sandbox Anubis per la generazione dei report comportamentali. Le distanze tra report comportamentali fanno uso della distanza Jaccardi, che nel caso di confronto tra vettori contenenti solo 0 e 1 si riduce alla distanza coseno come vedremo nel capitolo successivo. Questa misura di distanza non viene calcolata tra l'insieme delle caratteristiche di ogni campione di malware, bensì su una piccola quantità di dati chiamata firma comportamentale. L'algoritmo presenta pertanto alta scalabilità per via del fatto che non vengono confrontati i report interi, bensì una funzione di hash degli stessi. Per quanto riguarda il clustering, viene utilizzato l'algoritmo single linkage. L'algoritmo viene fatto girare su di un sottoinsieme dei campioni di partenza che abbiano un valore di distanza al

di sotto di una certa soglia definibile.
Un tentativo di utilizzo di sole firme comportamentali riguardanti il traffico HTTP è presentato in [46]. Dagli autori del sistema, viene posta particolare attenzione su similitudini strutturali nel traffico HTTP generato da comportamenti maligni derivati dall'esecuzione di malware. Viene inoltre fatto l'assunto che le firme a livello di rete, possiedano delle proprietà migliori paragonate alle firme comportamentali di livello sistema. L'assunto di base è che il malware odierno abbia una assoluta necessità di connessione di rete per poter perpretare attività ai danni del sistema. Il sistema sviluppato mira a scoprire similitudini tra campioni di malware che non possono essere catturate da attuali sistemi di clustering comportamentale a livello di sistema; esso si occupa inoltre di imparare un modello di comportamento di rete per ogni gruppo o famiglia di malware, al fine di identificare la presenza di macchine compromesse da malware in una rete monitorata. Si cercano similitudini strutturali tra le sequenze di richieste HTTP generate come conseguenza di una infezione. I campioni di malware vengono eseguiti in un ambiente controllato simile a BotLab per un certo periodo t; si partiziona poi l'insieme dei malware in base a similitudini rilevate tra le traccie di rete. Per ridurre il costo computazionale, viene utilizzato un algoritmo di clustering multifase che descriviamo:

1. Clustering a grana grossa: vengono raccolte informazioni statistiche sul traffico e calcolate le distanze in modo efficiente dal momento che sono confronti tra vettori di dimensioni molto ridotte.

2. Clustering a grana fine: all'interno dello stesso cluster creato al passo precedente, si cercano differenze strutturali tra le query HTTP, con l'agevolazione che si lavora su dataset ridotti formati dai membri dello stesso cluster. Il confronto è dispendioso in termini computazionali. Viene utilizzata la distanza "Normalized Levenshtein" per confrontare le stringhe che compongono il path dell'url ed il concatenamento dei valori dei parametri, la Jaccardi tra il nome dei parametri della get o post.

3. Accorpamento dei cluster: dal momento che il passo precedente produce cluster molto ridotti, si tenta di ricondursi a cluster con comportamento generale simile, riaccorpando cluster che esibiscono un comportamento sufficientemente simile. Si calcolano i centroidi dei cluster ridotti e poi si calcolano le similitudini tra centroidi per poter effettuare l'accorpamento; come conseguenza di ciò, dovrebbe aumentare il tasso di rilevamento.

Per quanto concerne l'algoritmo di clustering utilizzato, in tutte e tre le fasi del clustering multifase, viene adottato il clustering single linkage. Per ultimo, si noti la necessità di dover discriminare tra richieste legittime e richieste potenzialmente dannose. Per raggiungere tale scopo, anzichè effettuare il prefiltraggio delle url, si effettua un pruning delle url basato su un database di richieste legittime servite da server legittimi.

2.3 Critiche e studi riguardanti gli algoritmi di clustering

In questo paragrafo, vengono presentate le principali critiche ed osservazioni mosse contro gli algoritmi di clustering presentati nel paragrafo precedente. La principale problematica per qualsiasi metodo di clustering deriva dalla sua natura non supervisionata, ad esempio la mancanza di qualsiasi informazione esterna fornita per guidare l'analisi dei dati. Inoltre, è molto difficile decidere quanti cluster sono presenti nei dati [50]. Venendo alle misure di distanza utilizzate nell'analisi, lo studio presentato in [14] pone a paragone differenti misure nell'ambito di clustering di malware, utilizzando l'algoritmo di clustering single linkage. In questo studio, si evidenzia come il metodo di clustering usato in [16] utilizzi la Normalized Compression Distance senza darne giustificazione; dal momento che è una misura universale, essa è adatta per casi generali e non per il problema specifico dell'analisi di malware. Il sistema in [60], utilizza una variante della Edit Distance che ha una complessità temporale quadratica, dovendo trovare il costo minimo per far diventare due stringhe congruenti, e quindi ha una complessità troppo elevata per l'analisi. Una misura di distanza è considerata migliore rispetto ad un'altra, se risulta possibile trovare comportamento condiviso per più report comportamentali; se il numero di report per i quali il comportamento condiviso è uguale, si passa a considerare il numero di cluster. La distanza di misura che porta ad un numero di cluster inferiore fornisce una descrizione più breve del comportamento condiviso ed è quindi preferibile. I risultati della ricerca in [14] hanno dimostrato che la misura di distanza Normalized Compression Distance (lzma) non può differenziare sufficientemente tra report di malware. Le varianti della Normalized Compression Distance e la Edit Distance non dovrebbero essere dunque utilizzate, mentre i risultati sperimentali hanno condotto ad una preferenza della Manhattan Distance.
Come seconda problematica, notiamo che vi è una notevole difficoltà nel trovare ground-truth mediante le quali valutare i risultati di un classificatore proposto. Inoltre, non vi è consistenza tra le label assegnate tra diversi

produttori di AV [50]. Il fatto di prelevare campioni su cui vi è molta concordanza tra antivirus, come ad esempio in [65], si traduce in una facilità di classificazione da parte di una certa varietà di tecniche. D'altro canto, anche la classificazione manuale risulta problematica.
Nello studio proposto in [44], l'intento è di ripetere il clustering di istanze già processate in [65], usando algoritmi di un dominio differente, chiamati individuatori di plagio. In maniera intuitiva, dal momento che gli individuatori di plagio sono implementati senza porre attenzione alle specifiche dell'offuscamento del malware, risultati altamente accurati di clustering da parte di questi strumenti possono portare alla conclusione che questo metodo di selezione della ground truth discrimina i dati tra istanze facilmente classificabili. Non ci si aspetta dunque che gli identificatori di plagio ottengano risultati simili dal momento che il malware è offuscato e che il malware rappresenta un segmento a parte rispetto ai programmi leciti. Nelle conclusioni individuate dallo studio stesso [44], viene detto comunque che non si sono raggiunte evidenze sufficienti a dimostrare l'ipotesi asserita.
Come puntualizzato in [16], esiste un trade-off tra la dimensione del cluster ed il numero di cluster controllato da un parametro chiamato consistenza che misura il rapporto tra la variazione intra-cluster ed inter-cluster. Un buon algoritmo di clustering, dovrebbe anche mostrare alta consistenza, e quindi si dovrebbe osservare un comportamento uniforme tra elementi appartenenti allo stesso cluster ed eterogeneo tra cluster distinti. Nel caso del comportamento di malware, eterogeneo per natura, questa osservazione implica che, qualora sia necessario mantenere la consistenza alta, osserveremmo un gran numero di piccole classi. Tuttavia, il comportamento desiderabile è esattamente l'opposto: un piccolo numero di grandi cluster nei quali le varianti appartengono alla stessa famiglia. L'unico modo di ottenere questo effetto, utilizzando la consistenza, è di testare differenti livelli di consistenza; questo però è contrario allo scopo della classificazione automatica e può comunque essere difficoltoso da ottenere ad un singolo livello di consistenza [50]. Gli studi effettuati sull'argomento mettono in risalto ancora di più il fatto che questo campo di ricerca resta ancora aperto.

2.4 Anubis

La piattaforma web Anubis [1] utilizza il tool TTAnalize [62] per analizzare eseguibili Windows nel formato PE. Per espletare tale funzione, un campione del malware viene eseguito in ambiente emulato e vengono loggate sia chiamate alle API Windows sia chiamate al sistema nativo. Una prima scelta effettuata in Anubis, a differenza dei virus scanner dove l'ambiente emula il

processore e fornisce una implementazione leggera dell'interfaccia del sistema operativo, consiste nell'emulare l'hardware in modo da poter installare il sistema operativo senza apportarvi modifiche. In particolare [62], la soluzione dei virus scanner soffre di problematiche inerenti la possibile rottura del codice in esecuzione e la possibilità da parte del malware di individuare l'ambiente emulato, mentre la soluzione di TTAnalize soffre di una maggiore tempistica di esecuzione. Per diminuire i tempi di esecuzione si utilizza la tecnica della "traduzione dinamica" dove si traducono blocchi anzichè singole istruzioni.
L'ambiente emulato utilizzato in Anubis [62] è QEMU. In questo ambiente emulato è presente un'installazione di Microsoft Windows service pack 2 per la quale viene creato uno snapshot a boot del sistema operativo appena avvenuto per evitare le latenze dovute all'avvio stesso. L'esecuzione di campioni di malware successivi avviene solo dopo aver ripristinato la macchina in questo stato salvato. Una volta lanciato l'eseguibile in memoria, si tiene traccia delle azioni che il codice esegue sull'ambiente emulato; ogni azione che richiede una comunicazione con tale ambiente, deve essere eseguita da un processo Windows in user mode al fine di poter far uso di chiamate al sistema operativo. Tale comunicazione viene gestita dalle Windows API che effettuano opportune chiamate in user mode alle API native Windows non documentate contenute nella libreria *ntdll.dll*. Il motivo di tale scelta in Windows è di schermare opportunamente le applicazioni dalla complessità delle API native sottostanti che possono cambiare tra di una versione Windows ed un'altra; chiamate dirette alle API native non sono dunque buona norma nella programmazione normale. Vengono tuttavia sfruttate dagli autori di malware, al fine di eliminare dipendenze da dll e per confondere gli antivirus. Anubis tiene dunque traccia delle due tipologie di chiamate di sistema. Per far questo, viene controllato il valore corrente del registro *eip* confrontandolo con gli start address delle funzioni Windows che vogliamo monitorare. Il valore dello start address è calcolato dalla export table della dll che contiene la funzione chiamata. Dal momento che le librerie dinamiche possono essere caricate in memoria in un qualsiasi indirizzo, si aggiunge all'indirizzo della prima posizione in memoria della dll, l'indirizzo relativo della funzione nella dll stessa fornito dalla export table.
Un'altra problematica è il controllo relativo all'identità del processo chiamante la funzione. Per essere sicuri che si stiano monitorando le chiamate del processo del malware, si sfrutta il registro *cr3*: Windows salva in questo registro il valore univoco per processo dell'indirizzo "page directory", che viene caricato ad ogni switch di contesto. Sfruttando questo fatto, siamo in grado di capire se il contesto che sta girando in memoria è relativo al

nostro processo. Questa metodologia consente una strategia meno intrusiva in quanto non introduce modifiche al codice. La lettura dei parametri delle funzioni viene eseguita invece leggendoli direttamente dalle "callback routine". Le routine sono implementate attraverso degli stub delle chiamate di sistema che vogliamo monitorare. Parsando queste dichiarazioni di funzioni, il generatore (sistema che incorpora le "callback routine") è in grado di generare il codice C in grado di leggere le dimensioni e le strutture dei parametri di tali funzioni. Mentre, in linea di principio, qualsiasi attività virale può essere monitorata da TTAnalize, si è deciso di monitorare solo un numero di callback routine che analizzano e loggano azioni rilevanti in ambito di sicurezza.
I report generati contengono le seguenti informazioni:

1. Informazioni generali: questa sezione contiene informazioni circa l'invocazione di TTAnalyze, gli argomenti della command line ed alcune informazioni generali circa il campione del test (ad es la dimensione del file, il codice di uscita, il tempo necessario ad effettuare l'analisi, ...).

2. Attività su file: questa sezione ricopre l'attività su file da parte del campione del test (ad es file modificati, creati, ...).

3. Attività di registro: in questa sezione, vengono tracciate tutte le modifiche fatte al registro di sistema di Windows e tutti i valori di registro che sono stati letti dal campione del test.

4. Attività di servizio: questa sezione documenta tutte le interazioni tra il campione del test ed il Windows Service Manager. Se il campione ad esempio avvia o ferma un servizio, l'informazione è qui tracciata.

5. Attività di processo: in questa sezione viene monitorata la creazione e terminazione di processi, nonchè l'intercomunicazione tra i processi del campione ed altri in esecuzione sul sistema operativo.

6. Attività di rete: questa sezione fornisce un link ad un log che contiene tutto il traffico di rete mandato o ricevuto dal campione del test.

2.4.1 Punti di forza di Anubis

In [64] troviamo un certo numero di statistiche riguardanti le prestazioni di Anubis. Ereditando i pregi di un'analisi di tipo comportamentale, TTAnalize consente di [63]:

1. Eliminare il gap tra l'uscita del virus ed il rilascio delle signature degli antivirus: infatti, analizzando l'attività virale in ambiente emulato, siamo già in grado di stabilire quali attività il campione esegue e quindi prevedere un comportamento sospetto o dannoso sul sistema.

2. Scandire un numero notevole di campioni in tempi brevi: l' analisi dei file eseguibile avviene generalmente in ambiente controllato mediante un debugger da un esperto di sicurezza. L'intervendo di un analista umano infatti è desiderabile dopo l'analisi automatica.

3. Illudere le routine anti-antivirus: gli approcci correnti di scansione automatica vengono elusi da routine di evasione implementate nei virus, sia per quanto riguarda il rilevamento di breakpoint hardware e software, sia per il rilevamento di virtual machine sia di OS simulati.

4. Analizzare solo le istruzioni che effettivamente vengono eseguite dal virus.

5. Immunità ai tentativi di offuscamento e di auto-modifica dei virus.

6. Individuare mutazioni polimorfiche di virus individuate come virus differenti dai comuni antivirus per via delle routine di *crypting* [61].

7. Riconoscere i principali packer con i quali i virus vengono impachettati: Anubis, mediante il tool sigbuster, individua il packer utilizzato e ne permette di conseguenza una facile decompressione, salvo alcune eccezioni [64].

2.4.2 Problematiche di Anubis

In questo paragrafo verranno descritti i limiti di Anubis, la sandbox utilizzata in fase di analisi dinamica dei campioni di malware. Si è detto che la piattaforma genera dei report dell'attività virale riscontrata in ambiente emulato Qemu ed installazione Windows service pack 2. Di seguito si tracciano le possibili problematiche:

1. I report presentano in tutti questi sotto campioni la seguente traccia comportamentale: il file infetto è un eseguibile dll che viene caricato in memoria mediante il *regsvr*32 di Windows con la seguente commandline, registrando la *dll* come servizio di sistema Windows:

   regsvr32.exe /c /s ./d1.tmp.dll

   da questo punto in poi il campione non presenta ulteriori informazioni comportamentali significative.

2. Altra problematica deriva dalla presenza di campioni trojan downloader per i quali viene evidenziato solamente il fatto che il campione tenta di eseguire un download del vero trojan. Inoltre, campioni per i quali viene semplicemente modificato il file autorun.inf vengono clusterizzati dal punto di vista comportamentale nello stesso gruppo. Di conseguenza, il vero eseguibile è quello lanciato dall'esecuzione dell'autorun.inf. Il file eseguibile si replica in una cartella e viene lanciato il file eseguibile replicato come parametro stesso del campione in esecuzione in memoria. Per ultimo, alcuni campioni si occupano di replicarsi in un altro file eseguibile e di modificare la chiave di registro relativa all'esecuzione automatica di programmi all'avvio di Windows. Il codice del campione vero e proprio è insito nell'eseguibile che verrà successivamente lanciato. Si otterrà quindi che il comportamento dei campioni codificati in questa modalità risulterà in una appartenenza allo stesso cluster comportamentale.

3. Infine, è possibile che il campione tenti di avviarsi e: non trovi dll richieste, non riesca a procedere nell'esecuzione (il sistema non è vulnerabile ad un preciso bug richiesto per l'esecuzione del codice del campione), individui Anubis (quest'ultima è molto rara), debba essere eseguito ad una data ben precisa che non corrisponde al giorno dell'esecuzione in Anubis.

4. Il packer del malware non è conosciuto da Anubis e l'analisi è interrotta.

5. Tempo limite di 4 minuti dopo il quale il malware viene comunque interrotto.

Per confrontare i due algoritmi ci aspettiamo che le situazioni suddette vengano individuate in una similitudine tra campioni, anche se virus total ha attribuito famiglie diverse di appartenenza dei due campioni per altri motivi, in particolare per la firma statica del malware riconosciuta dagli antivirus in virus total. I restanti campioni saranno molto più significativi in merito all'analisi da effettuare.

Capitolo 3

Impostazione del problema di ricerca

"La disumanità del computer sta nel fatto che, una volta programmato e messo in funzione, si comporta in maniera perfettamente onesta."

Isaac Asimov

Nel terzo capitolo vengono presentate alcune definizioni preliminari per poter facilitare la comprensione dei capitoli precedenti, si approfondiscono gli obiettivi presentati nel capitolo introduttivo e si entra nel dettaglio delle scelte prese per risolvere le problematiche affrontate.

3.1 Definizioni preliminari

Riassumiamo in questa sezione delle definizioni preliminari che saranno utili nella lettura della parte successiva della tesi.

3.1.1 Parametri di precisione e di richiamo

La precisione ed il richiamo sono due metriche largamente utilizzate in *information retrieval* per valutare la correttezza di un algoritmo di riconoscimento di pattern. Esse possono essere viste come una versione estesa dell'accuratezza, una semplice metrica che calcola la frazione di istanze per le quali viene restituito il risultato corretto. Una precisione pari ad 1.0 per una classe $\mathcal{D}$ significa che ogni campione etichettato come appartenente alla classe $\mathcal{D}$ appartiene davvero alla classe (ma non dice nulla circa il numero di campioni della classe $\mathcal{D}$ che non hanno ricevuto un'etichetta corretta), mentre un

richiamo pari ad 1.0 significa che ogni campione in $\mathcal{D}$ è stato etichettato correttamente come appartenente alla classe $\mathcal{D}$ (ma non dice nulla sulle istanze che sono state scorrettamente etichettate come appartenenti alla classe $\mathcal{D}$). La precisione può essere vista come una misura di esattezza o fedeltà, mentre il richiamo come una misura della completezza dell'algoritmo. In termini ancora più semplici, un'alto richiamo significa che non si è perso nulla ma si possono avere molti risultati inutili da vagliare (che implicherebbero bassa precisione). Un'alta precisione significa che tutto ciò che viene restituito è un risultato rilevante, ma non è stato possibile trovare tutti gli oggetti rilevanti (che implicherebbe basso richiamo). Una visione probabilistica delle due metriche definisce la precisione come la probabilità che un oggetto recuperato in maniera casuale sia rilevante, mentre il richiamo rappresenta la probabilità che un documento rilevante preso a caso sia recuperato. Rappresentiamo con M una collezione di m campioni di malware che devono essere clusterizzate. Siano poi $\mathcal{C} = \{C_i\}_{1 \leq i \leq c}$ e $\mathcal{D} = \{D_i\}_{1 \leq i \leq d}$ due partizioni di M e siano $f : \{1...c\} \leftarrow \{1...d\}$ e $g : \{1...d\} \leftarrow \{1...c\}$ due funzioni.
Definiamo ora le due misure come:

$$prec\,(\mathcal{C}, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{c} |C_i \cap D_{f(i)}| \tag{3.1}$$

$$recall\,(\mathcal{C}, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{d} |C_{g(i)} \cap D_i| \tag{3.2}$$

dove $\mathcal{C}$ è l'insieme di cluster che si stanno valutando e $\mathcal{D}$ è il clustering che rappresenta la "risposta giusta". Nello specifico, nel caso della classificazione, C_i rappresenta l'insieme di tutte le istanze test classificate come classe i e D_i sono tutte le istanze che in realtà appartengono alla classe i. Nel caso particolare della classificazione, $c = d$ ed inoltre f e g sono le funzioni identità.
Ne risulta di conseguenza che $prec\,(\mathcal{C}, \mathcal{D}) = recall\,(\mathcal{C}, \mathcal{D})$ e questa misura è di solito semplicemente chiamata *"accuratezza"*. Nel caso del clustering invece, non esiste un'etichetta specifica per definire il cluster in $\mathcal{D}$ che corrisponde allo specifico cluster in $\mathcal{C}$, e quindi un possibile approccio è di definire:

$$f(i) = \arg\max_{i'} |C_i \cap D_{i'}| \tag{3.3}$$

$$g(i) = \arg\max_{i'} |C_{i'} \cap D_i| \tag{3.4}$$

In questo caso, f e g non saranno in generale le funzioni identità, e di conseguenza la precisione ed il richiamo sono differenti.

3.1.2 Decomposizione ai valori singolari

In questa sezione vogliamo presentare delle definizioni preliminari inerenti la decomposizione ai valori singolari, utilizzata dal nostro sistema per effettuare la analisi semantica latente. Data la forte connotazione matematica, ed in particolare nell'ambito dell'algebra lineare, riportiamo anche concetti generali sugli spazi vettoriali, per giungere infine alle definizioni di interesse per la comprensione del resto dell'esposizione.

1. Sottospazio generato: i vettori v che si ottengono come combinazioni lineari di n vettori fissati, al variare degli scalari $a_1, a_2 \dots a_n$ formano un sottospazio vettoriale di V chiamato sottospazio generato.

2. Base: un insieme di vettori di uno spazio vettoriale V è una base se questi vettori sono indipendenti e generano lo spazio V.

3. Base ortogonale: sia V uno spazio vettoriale di dimensione finita, dotato di prodotto scalare. Una base ortogonale per V è una base composta da vettori a due a due ortogonali. Due vettori sono ortogonali quando il loro prodotto scalare è uguale a zero.

4. Base ortonormale: questa è una base ortogonale in cui ogni vettore ha norma uno, a patto che il prodotto vettoriale sia semidefinito positivo.

5. Matrice di cambiamento di base: matrice quadrata che codifica il cambiamento di una base di uno spazio vettoriale.

6. Matrice ortogonale: matrice di cambiamento di base tra due basi ortonormali, matrice invertibile la cui trasposta coincide con la propria inversa ($MM^t = I$).

7. Decomposizione QR: Se M è una arbitraria matrice di tipo $m \times n$ di rango $n (m \geq n)$, possiamo sempre scrivere

   $$M = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

   dove Q è una matrice ortogonale di tipo $m \times m$ e R è una matrice triangolare superiore di tipo $n \times n$ con valori positivi sulla diagonale principale.

8. SVD intera [24, 36, 49]: sia $M \in \mathbb{R}^{m \times n}$ allora esiste una fattorizzazione della stessa nella forma:

   $$M = U \Sigma V^t$$

   dove:

(a) U è una matrice ortogonale di dimensioni $m \times m$

(b) Σ è una matrice diagonale di dimensioni $m \times n$

(c) V^t è la trasposta di una matrice ortogonale di dimensioni $n \times n$

gli elementi della diagonale di Σ sono detti valori singolari di M e hanno la proprietà di essere tutti quanti positivi, $s_i \geq 0 \quad \forall i$, e $s_1 \geq s_2 \geq \ldots \geq s_n$

Si può dimostrare che il rango della matrice M è uguale a quello della matrice Σ. In particolare si osserva che il rango di Σ dipende dai valori singolari ed è proprio uguale al numero di valori singolari diversi da zero.

9. SVD ristretta: $M = U_n \Sigma_n V^t$

   Soltanto gli n vettori colonna di U corrispondenti ai vettori riga di V^t sono calcolati. I restanti vettori colonna di U non sono calcolati. Questo procedimento è molto più veloce ed economico rispetto a calcolare la decomposizione SVD intera se $n << m$. La matrice U_n è di dimensione $m \times n$, Σ_n è diagonale di dimensione $n \times n$ e V di dimensione $n \times n$. Il primo stadio del calcolo della SVD ristretta è generalmente la decomposizione QR di M, che porta ad un calcolo molto più veloce se $n << m$.

10. SVD troncata: $\tilde{M} = U_z \Sigma_z V_z^t$

    Soltanto i z vettori colonna di U ed i z vettori riga di V^t corrispondenti ai più grandi valori singolari z di Σ_z sono calcolati. Il resto della matrice non è computata. Questo procedimento risulta molto economico se $z << r$. La matrice U_z è di dimensione $m \times z$, Σ_z è diagonale di dimensione $z \times z$ e V_z è di dimensione $z \times n$. Ovviamente la SVD troncata non è più una decomposizione esatta della matrice originale M, ma la matrice approssimata $\tilde{M}$ è l'approssimazione più vicina ad M che può essere raggiunta da una matrice di rango z nel senso indicato dalla norma di Frobenius.

3.1.3 Misure di distanza

Presentiamo brevemente le misure di distanza utilizzate dai due algoritmi presentati in [65] ed in questo elaborato. La misura di distanza coseno tra

due vettori A e B viene calcolata mediante la seguente formula:

$$cosinedist(A, B) = 1 - \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \tag{3.5}$$

mentre la distanza Jaccardi tra due vettori A e B nel seguente modo:

$$jaccarddist(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{3.6}$$

Nel caso dell'analisi dei testi, poiché le frequenze dei termini sono sempre valori positivi, si otterranno valori che vanno da 0 a 1, dove 1 indica che le parole contenute nei due testi sono le stesse (ma non necessariamente nello stesso ordine) e 0 che non c'è nessuna parola che appare in entrambi. Oltre a questo, nel caso di coefficienti binari, la distanza coseno si riduce alla distanza Jaccardi. La distanza Jaccardi sotto queste assunzioni è identica al coefficiente di Tanimoto.

3.2 Problematiche affrontate e metodologie utilizzate

In questo paragrafo si presentano le problematiche affrontate e la particolare metodologia che si è usata per risolverle. La clusterizzazione di malware in generale, richiede di prendere una molteplicità di decisioni in diversi punti di criticità. Riprendendo brevemente ciò che si è detto nel capitolo precedente, possiamo riassumere i punti decisionali nei seguenti:

1. Scelta del malware da analizzare,
2. Scelta di una ground truth,
3. Scelta degli elementi estrapolati dai report comportamentali da utilizzare,
4. Scelta di una misura di distanza per il calcolo dei confronti tra le coppie di malware,
5. Scelta dell'algoritmo di clustering gerarchico che effettui la suddivisione in famiglie vera e propria dei campioni analizzati.

3.2.1 Scelta del malware da analizzare

Per quanto concerne il primo punto, uno dei punti di snodo cruciali per valutare la bontà dell'algoritmo, si è deciso di prendere un insieme di campioni

di malware dal sito virus total [12]. I campioni di malware vengono raccolti sia da sottomissioni di eseguibili da parte di utenti tramite interfaccia web, sia da altre fonti (spamtraps, honeypots, ecc..).

3.2.2 Scelta di una ground truth

Gli eseguibili così raccolti vengono scansionati mediante 42 antivirus commerciali. Dalla scansione del singolo eseguibile, verrà assegnata, nel caso lo stesso fosse rilevato come malware, un'etichetta di appartenenza ad una famiglia derivata dall'analisi statica. Un esempio di questa classificazione determinerà ad esempio che il generico malware x apparterrà alla classe denominata y ricavata dalla scansione di tutti gli antivirus che daranno per questo campione famiglie di appartenenza rifacenti alla stessa (NB: a meno di nomenclatura differente da parte degli antivirus; ad esempio per un campione di zbot possiamo ottenere: "Trojan/Win32.Zbot.gen", "Trojan/-Spy.Zbot.augx"). Il set di campioni che abbiamo prelevato, presenta campioni ad alto ranking: questo si traduce in una concordanza tra la maggior parte (o quasi totalità) degli antivirus circa la famiglia di appartenenza derivata dall'analisi statica. Va infine notato che all'interno di questi campioni abbiamo anche varianti polimorfiche degli stessi e di conseguenza ci aspetteremo di ottenere famiglie comportamentali del tutto simili alle famiglie classificate per via dell'analisi statica; questo approccio è stato utilizzato per determinare una ground truth solida su cui appoggiarsi per la valutazione del nostro algoritmo rispetto all'algoritmo in [65]. Abbiamo deciso inoltre di rendere il dataset paritario (ovvero campioni omogeneamente ripartiti sulle famiglie derivate dall'analisi statica). Le motivazioni di questa scelta sono frutto dell'analisi condotta in [44], dove si pone enfasi sulle criticità derivate dalla selezione di una ground truth e dalla valutazione degli algoritmi di clustering utilizzando le metriche di precisione e richiamo che abbiamo introdotto nel paragrafo successivo. Nello studio citato, vengono fatte ipotesi riguardo la differenza di valori di precisione e richiamo ottenuti facendo girare l'algoritmo in [65] su dataset differentemente ripartiti. La tesi è avvalorata dai risultati differenti, in merito a queste metriche, ottenuti spiegando questa differenza in termini di un parametro chiamato "significato"; questo parametro rappresenta il fatto che dei dati valori di precisione e richiamo possano essere occorsi per un evento casuale, e più esso è basso, più risulterà elevata questa probabilità. Date le due clusterizzazioni C (prodotta dall'algoritmo che stiamo valutando) e D (clusterizzazione prodotta dall'analisi statica, ground truth) viene dimostrato come la distribuzione di questi due insiemi va ad inficiare la precisione ed/o il richiamo. In particolare, se le distribuzioni dei

due insiemi sono paritarie, la precisione ed il richiamo assumono un valore del parametro significato maggiore rispetto a situazioni in cui i due insiemi risultino ripartiti in maniera molto differente [44]. Gli autori di questo studio hanno rilevato dei valori di precisione e di richiamo decisamente inferiori rispetto a quanto dichiarato in [65], valori che abbiamo ritrovato anche nei nostri esperimenti di confronto dei due algoritmi sviluppati.

3.2.3 Scelta di una misura di distanza per il calcolo dei confronti tra le coppie di malware

Altra problematica cui si è andati incontro è la scelta di una misura di distanza congrua all'applicazione che è stata sviluppata. Nel tentativo di trovare una misura di distanza ottima per il dominio applicativo di nostro interesse, l'analisi delle differenze comportamentali tra due campioni di malware, sono state effettuate varie ricerche per dare risposta a questo quesito in [14,26]. Si è giunti alla conclusione che alcune misure di distanza non vanno sicuramente bene e di questo abbiamo già parlato nei capitoli precedenti. Si è quindi deciso di utilizzare una misura di distanza tra vettori, la misura coseno; tale scelta è motivata anzitutto dal fatto che si riduce alla distanza Jaccardi (utilizzata in [65]) nel caso in cui i coefficienti su cui viene calcolata siano binari positivi. Dal momento che si è deciso di non considerare la ripetizione dei comportamenti all'interno dello stesso report, come vedremo nel paragrafo successivo, si è ottenuto che la misura di distanza coseno si riduce al coefficiente di Tanimoto, altro nome per denotare la distanza Jaccardi sotto queste condizioni. Il motivo per cui non si è scelta direttamente la distanza Jaccardi è la relativa facilità di calcolo, in termini di complessità computazionale, e la mancanza di necessità di dover cambiare la rappresentazione vettoriale con cui si rappresenta il singolo campione nella nostra rappresentazione vettoriale nel caso in cui si voglia calcolare un differente valore di distanza.

3.2.4 Scelta degli elementi estrapolati dai report comportamentali da utilizzare

In questa sottosezione vengono descritte le scelte effettuate per filtrare l'informazione contenuta nei report comportamentali prodotti da Anubis. Un report è un file *xml* strutturato in diverse sezioni in cui vengono loggate le attività svolte dal malware sul sistema operativo. Vengono tracciate attività di rete, dipendenze dll, attività su filesystem, informazioni generiche del processo. Ogni attività viene esplicitata in un tag a sè stante contenente

degli attributi che riportano in maniera approfondita i dettagli dell'attività. Prendendo ad esempio l'attività di creazione di un socket:

```
<socket close_time="not-a-date-time" create_time="2010-Aug-21 16:44:40.853617"
    created_by_thread="5" foreign_ip="127.0.0.1" foreign_port="1038" is_listening
    ="0" local_ip="127.0.0.1" local_port="1038" type="udp" />
```

si vede come Anubis scende ad un livello di profondità di analisi spiccato. L'informazione può essere pertanto filtrata al fine di eliminare informazione rindondante, informazione non necessaria alla nostra analisi, ad esempio gli attributi generici, informazione uguale per tutti i report, informazione che renderebbe immotivatamente diverse due caratteristiche. Nell'esempio del socket si può pertanto eliminare informazione che sappiamo a priori essere potenzialmente uguale per tutti i report, l'informazione sull'indirizzo ip locale su cui un socket in listen effettua la bind, e la data di creazione del socket, che renderebbe due caratteristiche sempre diverse essendo univoca. sono state pertanto effettuate semplificazioni simili sino a giungere ad un sottoinsieme di tag ed attributi da mantenere nell'analisi.
Un altro aspetto su cui si è concentrata l'attenzione, è l'attributo, presente in un certo numero di caratteristiche, inerente al conteggio di quante volte viene effettuata un'operazione. Un ragionamento semplice permetterà di esplicare la problematica relativa al conteggio di quante volte viene effettuata un'attività. Prendiamo come esempio un malware contenente attività di keylogging:

```
<key_was_checked count="392" key="VK_LBUTTON (1)"/>
```

l'attributo count contiene il conteggio di quante volte è stata effettuato il controllo della pressione del tasto specificato nell'attributo "key" nell'arco temporale in cui il malware viene monitorato. Questo valore contiene dei valori dell'ordine del centinaio; probabilmente infatti, il check viene effettuato con un ciclo while. Nel calcolo effettuato con una misura di distanza tale per cui questo valore elevato risulterebbe in una preponderanza di esso rispetto alle altre caratteristiche, si otterrebbero dei risultati che potrebbero determinare due malware profondamente differenti sebbene strutturalmente identici. Si è deciso pertanto di non tenere in considerazione attributi di questa tipologia.

3.2.5 Scelta dell'algoritmo di clustering gerarchico

Passando infine alla problematica della scelta di un algoritmo di clustering che risulti il più possibile efficiente, ci si è ricondotti alla scelta dell'algoritmo di clustering gerarchico denominato average linkage. La scelta è stata

motivata da una maggiore efficienza dello stesso dimostrata su matrici delle distanze prodotte da valori di distanza generati da molteplici funzioni differenti. Sono stati effettuati benchmark in [26] prendendo in considerazione 12 misure di distanza. Sono state poi calcolate le misure di distanza tra campioni di malware ed incorporati nella matrice delle distanze. Viene poi calcolata la matrice cofenetica, dove ogni cella rappresenta la misura di prossimità cofenetica, ovvero il livello nel dendogramma al quale due campioni sono inclusi nello stesso cluster. Più la matrice cofenetica è simile alla matrice delle distanze, più la gerarchia derivata dall'algoritmo di clustering si adatta ai dati. Per ognuna delle 12 misure di distanza testate, l'algoritmo average linkage è risultato il migliore; da qui la scelta di utilizzare tale algoritmo nell'implementazione.

Capitolo 4

Progetto logico ed architettura del sistema

"Se, dunque, sempre la verità degli esseri è nella nostra anima, l'anima dovrà essere immortale. Sicché bisogna mettersi con fiducia a ricercare ed a ricordare ciò che attualmente non si sa: questo è infatti ciò che non si ricorda."

Platone - Menone

In questo capitolo, dopo un'introduzione all'analisi semantica latente, si spiega la motivazione del suo utilizzo come preprocessamento per il calcolo delle distanze nel secondo algoritmo utilizzato per lo studio.

4.1 Introduzione all'analisi semantica latente

Si introduce in questo paragrafo la tecnica dell'analisi semantica latente (LSA nel seguito), tecnica utilizzata nel campo dell'information retrieval. Questa tecnica analizza le relazioni intercorrenti tra un insieme di documenti ed i termini in essi contenuti, producendo un insieme di concetti relazionati ai termini e ai documenti. L'assunzione di base è che le parole che sono affini in termini di significato, occorreranno spesso assieme nel testo.
L'algoritmo LSA inizia con una matrice termini-documenti. Questa matrice è una matrice sparsa che contiene in ogni cella $x_{m,n}$ il numero di occorrenze del termine t_i nel documento d_j.

$$\mathbf{t}_i^T \rightarrow \begin{array}{c} \mathbf{d}_j \\ \downarrow \\ \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} \end{array} \tag{4.1}$$

Il prodotto scalare $t_i^T t_p$ tra due vettori riga restituisce la correlazione tra i due termini in tutti i documenti e la matrice prodotto $X^T X$ contiene i prodotti di tutti i vettori riga; similmente, il prodotto scalare $d_j^T d_q$ tra due vettori colonna restituisce la correlazione tra i due documenti su tutti i termini e la matrice prodotto XX^T contiene i prodotti di tutti i vettori colonna. Successivamente, LSA applica la decomposizione ai valori singolari alla matrice (SVD nel seguito), una tipologia di analisi fattoriale. La matrice di partenza viene decomposta nel prodotto di 3 matrici $M = U\Sigma V^t$ assumendo la forma:

$$(\mathbf{t}_i^T) \rightarrow \begin{array}{c} U \\ \\ \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} \end{array} \cdot \begin{array}{c} \Sigma \\ \\ \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \end{array} \cdot \begin{array}{c} V^T \\ (\mathbf{d}_j) \\ \downarrow \\ \begin{bmatrix} \begin{bmatrix} & \mathbf{v}_1 & \end{bmatrix} \\ \vdots \\ \begin{bmatrix} & \mathbf{v}_l & \end{bmatrix} \end{bmatrix} \end{array} \tag{4.2}$$

dove l è pari al minimo tra le dimensioni della matrice di partenza ($l = \min(m, n)$), $\sigma_1, \dots, \sigma_l$ sono i valori singolari, $u_1, \dots, u_l$ e $v_1, \dots, v_l$ gli autovettori singolari destri e sinistri rispettivamente. U contiene gli autovettori di $X^T X$, V^T gli autovettori di XX^T e Σ la radice quadrata degli autovalori in comune ad XX^T e $X^T X$. La matrice frutto della decomposizione può essere troncata ad un certo livello $z << l$ eliminando le colonne di U, le righe di V^T e le righe e colonne di Σ di indice compreso tra z ed l ottenendo l'approssimazione di rango z con il minimo errore della matrice di partenza. La SVD fa uso inoltre della tecnica di analisi delle componenti principali, una tecnica utilizzata in ambito di statistica multivariata. Questa metodologia di analisi viene eseguita mediante una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema cartesiano nel quale la nuova variabile con la maggiore varianza viene proiettata sul primo asse, la variabile nuova, seconda per dimensione della varianza, sul secondo asse e così via. La riduzione della complessità avviene limitandosi

ad analizzare le principali (per varianza) tra le nuove variabili. Si ottiene dunque che le prime dimensioni della decomposizione sono quelle che catturano la maggior parte della varianza. Le dimensioni che spiegano la minor parte della varianza possono essere eliminate (scegliendo un valore per il parametro z) ottenendo un'approssimazione in cui il rumore e la rindondanza vengono rimosse, preservando invece relazioni importanti presenti nei dati [31–33]. Lo spazio vettoriale di dimensionalità ridotte è denominato *spazio dei concetti*. La conseguenza della diminuzione di rango della matrice di partenza è che alcune dimensioni sono combinate e dipendono da più di un termine. Ad esempio, se nella matrice di partenza avevamo in totale 4 termini $t1, t2, t3, t4$, una nuova dimensione u_1 dello spazio vettoriale ridotto potrebbe presentarsi come combinazione lineare di due di esse $u_1 = 1.3 * t1 + 0.2 * t2 + 0.0 * t3 + 0.0 * t4$ ed un'altra dimensione u_2 come combinazione delle restanti due $u_2 = 0.7 * t3 + 0.8 * t4 + 0.0 * t1 + 0.0 * t2$. Incorporando altre dimensioni (aumentando il valore del parametro z) si incorporeranno nella decomposizione relazioni tra termini che sono da un punto di vista statistico trascurabili in quanto spiegano pochissima varianza nei dati. Queste combinazioni lineari tra termini sono le colonne della matrice U (i vettori singolari sinistri) ed i coefficienti moltiplicativi rappresentano l'occorrenza del termine t_i nel concetto z; ad esempio, nel primo concetto, il termine $t3$ occorre per un fattore 0.7. In maniera del tutto simile, le combinazioni lineari tra documenti sono le righe della matrice V^T (i vettori singolari destri) ed i coefficienti moltiplicativi rappresentano la relazione tra il documento d_i ed il concetto z. Il concetto u_1 di esempio esprime dunque il fatto che, nella collezione di testi di partenza, il termine $t1$ cooccorre frequentemente con il termine $t2$. I termini ed i documenti possono essere quindi mappati come vettori in questo nuovo spazio prendendo rispettivamente come coordinate tutti i coefficienti del termine t_i in tutti i concetti ed i coefficienti del documento d_i in tutti i concetti.
La tecnica LSA prevede infine, tra le varie possibilità, il clustering dei documenti in questo nuovo spazio vettoriale. Se i documenti risultano vicini, in termini di distanza coseno tra i due vettori in questo spazio, essi risulteranno nello stesso cluster.

4.2 Traduzione dei concetti dell'analisi semantica latente nell'ambito del clustering di malware

Dopo aver introdotto la tecnica dell'analisi semantica latente nell'ambito dell'information retrieval, si contestualizzanno i concetti nel campo di nostro

interesse, il clustering di malware.
La matrice di partenza dell'analisi è del tutto simile alla matrice delle occorrenze termini-documenti. I documenti vengono sostituiti con i report xml generati da Anubis, mentre i termini non sono più dei vocaboli ma dei comportamenti manifestati dal malware. La matrice verrà denotata nel seguito come matrice caratteristiche-malware. Di conseguenza la matrice dei malware M conterrà in ogni cella $x_{m,n}$ il valore 1 se il malware n ha manifestato il comportamento m e 0 altrimenti. La motivazione della scelta dei soli valori binari è stata argomentata nel capitolo precedente. La matrice così prodotta viene decomposta mediante l'algoritmo SVD nelle 3 matrici U, Σ, V^T. Le colonne di U conterranno le varie dimensioni (i vari concetti nella terminologia dell'informatio retrieval) frutto dell'analisi delle componenti principali. La dimensione v_1 conterrà pertanto una combinazione lineare dei report i cui coefficienti esprimono la relazione che intercorre tra i report e questa prima dimensione. Ad esempio, la dimensione v_1 potrebbe contenere la combinazione lineare $1.3 * r_1 + 0.7 * r_2 + 0.03 * r_3 + 0.02 * r_4$ che esprime il fatto che i documenti r_1 ed r_2 sono fortemente relazionati a questo primo concetto. In corrispondenza della relazione v_1, la relazione u_1 sulle caratteristiche del malware potrebbe contenere la combinazione lineare $1.7 * f_1 + 1.1 * f_2 + 0.03 * f_3 + 0.02 * f_4$ esprimendo così la relazione tra i comportamenti $f_1 \dots f_4$ ed il primo concetto. Astraendo dai valori numerici, le relazioni d'esempio si traducono nel fatto che i report r_1 ed r_2, avendo dei coefficienti alti rispetto ad un concetto in cui i comportamenti f_1 ed f_2 sono associati (in virtù dei coefficienti alti nella relazione u_1) risulteranno vicini in questa dimensione. Interpretando ad esempio la relazione u_1 come f_1=caricamento in memoria della libreria dinamica *ntdll.dll* ed f_2=creazione di un mutex "x" il significato è che i report r_1 ed r_2 ben si mapperanno col concetto che esprime la ricorrenza della coppia dei due comportamenti. Ripetendo i ragionamenti fatti per la singola dimensione su tutti i concetti z (dopo aver troncato la decomposizione per questo valore), si ottiene che due report sono simili se la distanza coseno tra di essi è vicina a 0 in questo spazio vettoriale. La riduzione della dimensione dello spazio vettoriale alle prime componenti z è quindi determinante per la precisione del calcolo delle distanze. Se si fosse troncata la decomposizione alla sola prima dimensione nell'esempio, si sarebbe ottenuto che tutti i documenti che si relazionano bene al concetto u_1 verrebbero messi nello stesso cluster. Il che risulterebbe sicuramente in un'approssimazione grossolana.

4.3 Motivazione dell'utilizzo dell'analisi semantica latente nello studio

Si è visto nel primo paragrafo del capitolo che l'algoritmo SVD consente di approssimare una matrice con una decomposizione che restituisce un'insieme di tre matrici che moltiplicate tra di loro eguagliano la matrice di partenza. Il vantaggio è che la matrice può essere troncata ad un certo livello z ottenendo un'approssimazione della stessa che varia al variare di questo parametro. Il caso limite è rappresentato dalla scelta di non troncare l'approssimazione ottenendo la decomposizione esatta della matrice. Il parametro z rappresenta la dimensione del nuovo spazio vettoriale usato per rappresentare la matrice di partenza. Il calcolo delle distanze, che nel caso della matrice di partenza utilizzerebbe i suoi vettori riga di lunghezza pari al numero totale delle attività di tutti i malware, viene effettuato in questo nuovo spazio vettoriale. Il vantaggio considerevole è che la dimensione di questi vettori nel nuovo spazio è pari a z. Dato infine che i valori di z si aggirano attorno a $z = 250$ [15] per ottenere approssimazioni molto precise di matrici abbastanza grandi, si ottiene sicuramente un guadagno di performance nel calcolo delle distanze rispetto alla decomposizione completa. La scelta di non troncare la decomposizione corrisponde esattamente all'algoritmo numero tre utilizzato in questo studio dal momento che, non approssimando la matrice di partenza, si ottiene esattamente il calcolo delle distanze tra vettori di dimensione invariata (di dimensione dunque pari a tutte le attività rilevate in tutti i malware).
Con questo metodo di analisi è possibile dunque determinare, perseguendo uno degli obiettivi della tesi, come variando le approssimazioni effettuate sui vettori della matrice di partenza per migliorare le performance si vada ad impattare sulle metriche di analisi del nostro studio producendo variazioni nei valori delle distanze calcolate e di conseguenza sui gruppi formati in fase di clustering. Si è scelto questo algoritmo per poter valutare qualitativamente il degrado dei gruppi formati in fase di clustering; variando il valore del parametro z si otterrà dunque una clusterizzazione peggiore in maniera proporzionale alla diminuzione di tale valore.

Capitolo 5

Implementazione

"Una volta un tale che doveva fare una ricerca andava in biblioteca, trovava dieci titoli sull'argomento e li leggeva; oggi schiaccia un bottone del suo computer, riceve una bibliografia di diecimila titoli, e rinuncia."

Umberto Eco

5.1 Elementi di implementazione

Presentiamo nella prossima sezione alcuni aspetti significativi riguardanti l'implementazione dei tre algoritmi dello studio, del processamento dei report generati da Anubis e degli algoritmi di clustering gerarchico per costruire i cluster a partire dalla matrice del calcolo delle distanze.

5.1.1 Parsing dei report XML di Anubis

Il parsing dei report avviene con un parser DOM a struttura ricorsiva, mediante il quale si prelevano i tag con i vari attributi di interesse all'analisi, estrapolando per ogni tag una unica stringa che costituisce l'unità informativa minima dell'analisi semantica latente e che corrisponde ad una singola attività compiuta dal malware sull'ambiente emulato.
Questo insieme di stringhe prelevate da tutti i malware, viene successivamente hashato in MD5 stringa per stringa per poter calcolare in maniera rapida la matrice caratteristiche-malware contenente un valore pari a 1 se nel report m è stata individuata la stringa n. Infatti, nella seconda fase di parsing, si estrapolano le stringhe di ogni attività del singolo campione di malware, se ne effettua l'hash MD5 stringa per stringa e si cerca tale hash nel vettore contenente tutti gli hash MD5 delle attività di tutti i malware

estrapolati nella prima fase di parsing. Laddove il confronto dia esito positivo, si porrà un 1 in corrispondenza della riga e della colonna individuate. Ripetendo questo procedimento per tutti i campioni, si andrà a riempire per intero la matrice.

5.1.2 Dettagli implementativi algoritmo LSH

L'algoritmo LSH definito in [65] parte dalla matrice M caratteristiche-malware manipolandola però in maniera differente rispetto a LSA poiché diversi sono gli obiettivi dell'elaborazione; li discuteremo in seguito. L'algoritmo si divide in cinque parti: il calcolo dei min hash, il filtraggio LSH, il calcolo delle distanze Jaccardi dei campioni filtrati e la clusterizzazione vera e propria. L'idea che sta alla base del calcolo LSH è la seguente: si vuole mappare ogni colonna della matrice, ovvero l'insieme delle caratteristiche di ogni campione, su di una piccola quantità di dati che verrà chiamata nel seguito firma LSH. Si vuole ottenere che due colonne $C1$ e $C2$ siano altamente simili se le loro firme LSH sono altamente simili (quindi che due campioni di malware siano simili se lo sono le loro firme comportamentali). La misura di similitudine che si utilizza in questo contesto è la Jaccardi ed è pari al rapporto tra la cardinalità dell'intersezione e la cardinalità dell'unione dei due vettori colonna $C1$ e $C2$. Posto che la dimensione della firma debba essere di k bits, una prima idea consiste nel prendere k righe a caso, leggere il valore corrispondente nella colonna per ogni riga scelta, e calcolare una firma LSH di k bits del campione. Questo approccio non funziona perché si è assunta la matrice sparsa e quindi si otterrebbero firme con prevalenza di zeri, non significative in termini di paragone.
Il calcolo dei min hash invece, utilizza un approccio differente [27]: si permutano le righe della matrice in maniera random, si vede per ogni colonna, seguendo la permutazione delle righe, quale sia la prima riga nella quale si incontra il valore 1. Il valore così calcolato è il min hash $h(C)$ della colonna corrispondente alla permutazione delle righe. La probabilità che i valori $h(C1)$ e $h(C2)$ coincidano è esattamente uguale al valore della misura di similitudine di Jaccardi dei due vettori [65]. Ripetendo il calcolo dei min hash per k differenti permutazioni di tutte le righe, si ottiene una firma LSH per ogni campione di malware. Es: $LSH(C1) = h_1(C1), h_2(C1) \ldots, h_k(C1)$ e di conseguenza una firma LSH di k interi. Dal momento che permutare realmente in memoria le righe della matrice risulterebbe altamente inefficiente, una strategia possibile è di scegliere k funzioni di hash da una famiglia H di funzioni di hash della forma $h_i = (c1 \times i + c2) \pmod P \pmod{12388}$ (per una definizione teorica vedere [27], [13], [42]) , dove $c1$ e $c2$ sono costanti in-

tere comprese tra 1 e $P-1$, i il numero di riga, P un numero primo maggiore rispetto al numero totale di caratteristiche ($P = 12391 > 12388$) [65]. Anziché permutare fisicamente nella matrice k volte tutte le righe, si generano k permutazioni con le funzioni di hash lineari, ottenendo k permutazioni di tutte le righe della matrice.
Si crea una nuova matrice F funzioni hash-malware di dimensione $k \times n$, inizializzandone gli elementi ad infinito. Per generare le firme LSH dei campioni a questo punto si procede algoritmicamente: si scandisce ogni riga i della Matrice M caratteristiche-malware calcolando i valori delle k funzioni di hash per tale valore di i e si verifica quali colonne j assumono il valore 1 in corrispondenza di questa riga. Si aggiorna poi la nuova matrice F andando ad inserire in corrispondenza delle j trovate il valore calcolato della funzione di hash se e solo se tale valore è inferiore rispetto al valore già contenuto nella matrice. Una volta scandite tutte le righe si ottengono nella matrice F i valori min hash per ogni campione di malware, ovvero il numero di riga corrispondente al primo valore 1 incontrato nella matrice di partenza seguendo la permutazione delineata dalla funzione di hash. Leggendo per colonne tale matrice si ottiene inoltre una firma LSH di k interi del campione di malware.
Il secondo passo dell'algoritmo è il filtraggio Lsh. Si decide un valore di soglia di similitudine t al di là del quale si va ad effettuare la clusterizzazione e si ordinano i campioni in base al valore delle loro firme LSH. Se le firme LSH di due campioni u e v , lunghe k interi ciascuna, sono eguali, si aggiunge la coppia (u, v) all'insieme S. Ovviamente il metodo è probabilistico e si ottengono in S sia falsi positivi (elementi la cui similitudine in realtà è inferiore a t e quindi non dovrebbero stare in S) sia falsi negativi (elementi la cui similitudine in realtà è superiore rispetto a t e non sono stati immessi in S) [27] e, ripetendo il calcolo delle firme LSH per un numero di volte pari a l, è probabile che almeno in una delle l volte le firme dei due campioni realmente simili a livello t coincidano e quindi che il numero di falsi positivi venga ridotto (tuttavia ciò ha come effetto collaterale un aumento dei falsi negativi [27]). Infatti, la probabilità che due min hash siano eguali è pari alla similitudine Jaccardi tra i due campioni ($P(h(a) = h(b)) = Jaccardi(a, b) = v$) e di conseguenza la probabilità che due firme LSH siano uguali è pari alla precedente probabilità elevata alla lunghezza k della firma ($P(LSH(u) = LSH(v)) = Jaccardi(u, v)^k = v^k$). Ripetendo l'esperimento aleatorio per l volte, la probabilità che una coppia (a, b) appartenga ad S è pari a: $g(v) = 1-(1-v^k)^l$. Quindi, per ottenere un valore $g(t)$ vicino ad 1 per ogni coppia di campioni, si devono scegliere dei valori per k ed l che facciano tendere la funzione ad 1, affinché campioni con reale similitudine t siano inclusi in S con probabilità prossima ad 1. Come

dimostrato sperimentalmente in [65] un valore di t pari a 0.7 risulta un buon compromesso tra precision e recall e per tale valore, al fine di far tendere la $g(t)$ a 1, si selezionano $k = 10$ ed $l = 90$.
Si confronteranno i tre algoritmi su un cluster ad un valore di similitudine maggiore o uguale a t, data l'impossibilità di costruire una clusterizzazione gerarchica completa non approssimata in LSH poiché nel secondo passo dell'algoritmo vengono implicitamente scartati i campioni aventi un valore di similitudine inferiore a t. Tale informazione è infatti necessaria per generare la clusterizzazione gerarchica completa, LSH ne produce una gerarchica troncata.

5.1.3 Dettagli implementativi algoritmo LSA

L'algoritmo LSA, come implementato nella libreria Gensim [5], sfrutta gli algoritmi definiti in [37] ed in [51] per quanto riguarda l'incrementalità, in [10] ed in [41] per quanto riguarda il "core" matematico. Il parametro z è stato scelto pari a 250 per ottenere un'approssimazione molto buona (un parametro ottimale è compreso tra 200 e 500 per grandi dataset su cui effettuare la decomposizione SVD [15]), in linea con gli obiettivi dello studio. Ricordando che la decomposizione SVD scompone la matrice A nel prodotto di tre matrici $U\Sigma V^t$, si passa ora a delineare le funzioni relative alla decomposizione SVD come implementate in gensim. Innanzitutto, si è utilizzato l'algoritmo ad una passata come definito in [51] che richiama la routine LAS2 della libreria SVDLIBC [10], la quale restituisce i primi z valori singolari dominanti e le matrici U e V troncate al valore z della decomposizione. Tra le varie funzionalità che l'algoritmo LSA sviluppato in gensim offre si ricordano: l'ottimizzazione della SVD per matrici sparse [10, 41] e la natura incrementale dell'algoritmo, che non richiede di ricalcolare la decomposizione "from scratch" nell'aggiunta di un numero eventualmente infinito di nuovi campioni di malware (corpus training seriale ed online).
Per quanto riguarda i benefici ricavati dalla tipologia della matrice in input, si è ottenuto un forte ausilio dalla natura inerentemente sparsa della matrice, dal momento che l'algoritmo utilizza routine Basic Linear Algebra Subprograms (BLAS nel seguito) dedicate proprio ad agevolarsi della sparsità della matrice. Inoltre, la natura "online" dell'algoritmo, richiede che i nuovi campioni aggiunti in streaming, facciano gradualmente dimenticare le vecchie osservazioni a favore delle nuove, che vengono processate nella decomposizione ed eliminate (corpus training). Questa modalità di esecuzione della SVD tiene in memoria solo la parte sinistra della decomposizione, $P = U\Sigma$. In [51] viene introdotto l'algoritmo che esegue in modalità seriale o distri-

buita a piacimento. Il primo stadio dell'algoritmo distribuito, la decomposizione di base, consiste nel suddividere la matrice $M(m \times n)$ in "chunks" di dimensione variabile. Ogni chunk è una sottomatrice frutto del sottoinsieme delle colonne e di tutte le righe della matrice tale per cui riusciamo a processare il chunk in RAM nel nodo che esegue la parte di decomposizione. Di conseguenza chunk di grandi dimensioni migliorano le performance in ordine temporale ma richiedono più memoria nel nodo del cluster. Ogni chunk viene infatti processato da un nodo del cluster in maniera asincrona ed indipendente. Le varie decomposizioni possono essere calcolate utilizzando o la libreria [10] oppure l'algoritmo stocastico a due passate sulla matrice in input delineato in [41]. Una volta calcolate le varie decomposizioni $P_1 = U_1\Sigma_1 \dots P_n = U_n\Sigma_n$, si entra nel secondo stadio in cui si devono unire le decomposizioni calcolate nella singola, finale decomposizione $P = U\Sigma$. Per il dataset utilizzato nello studio, di dimensioni estremamente ridotte, non si è sfruttata la natura distribuita dell'algoritmo. Il motivo risiede nel fatto che il corpus è di dimensioni sufficienti ad essere elaborato tranquillamente in RAM con un solo chunk nel nodo locale.
Terminata la decomposizione, si è ottenuta una approssimazione della matrice di partenza e si potrà procedere a calcolare i valori delle similitudini tra i vettori dei report dei campioni di malware nello spazio dei concetti. La distanza viene computata tra vettori di dimensioni pari al numero totale delle dimensioni dello spazio dei concetti scelte con il parametro z. Infatti, ricordando dal capitolo precedente che la decomposizione SVD troncata crea una matrice V^T di dimensione $n \times z$ otteniamo che il calcolo delle distanze coseno in questo spazio vettoriale utilizzerà i vettori colonne di questa matrice v_l di dimensione $1 \times z$. Per la clusterizzazione infine, si è utilizzato un algoritmo di clustering agglomerativo che prende in ingresso la matrice 1000×1000 e costruisce un cluster gerarchico completo, il quale può essere troncato in corrispondenza di valori di similitudine Coseno superiori ad un valore di soglia di similitudine pari a t. Si avrà quindi un insieme di cluster (eventualmente vuoto od unico contenente tutti i campioni) nel quale gli elementi presentano una distanza angolare coseno inferiore a $1 - t$.

5.1.4 Algoritmo di calcolo delle distanze senza approssimazioni

L'algoritmo è interamente basato sulle librerie python scipy [8] e numpy [7]. Partendo dalla matrice caratteristiche-malware, vengono calcolate le distanze coseno tra tutti i vettori riga ottenendo la matrice che le contiene. Il calcolo non effettua nessuna approssimazione in quanto utilizza i vettori interi.

L'unica ottimizzazione che si è utilizzata, è il processamento della matrice di partenza mediante algoritmi che si agevolano della struttura sparsa dei vettori quando viene calcolata la distanza tra di essi.

5.2 Considerazioni sugli algoritmi di clustering gerarchico

Gli algoritmi di clustering gerarchico possono essere sia di tipo top-down, sia bottom-up. La tipologia bottom-up (clustering gerarchico agglomerativo) tratta ogni documento inizialmente come un singolo cluster e poi, successivamente, unisce (o agglomera) coppie di cluster sino a che tutti i cluster sono stati uniti in un singolo cluster che contiene tutti i documenti. Invece, il clustering di tipo top-down richiede un metodo per dividere un cluster; esso infatti procede dividendo i cluster ricorsivamente fino ad ottenere documenti singoli [9]. Il cluster gerarchico è tipicamente rappresentato come un dendrogramma: ogni fusione è rappresentata da una linea orizzontale. Le linee orizzontali hanno coordinata y che rappresenta la dissimilitudine dei due cluster che sono stati fusi assieme. Muovendosi in alto dal livello più basso, il dendrogramma permette di ricostruire la storia delle unioni che sono risultate nel cluster rappresentato. Il clustering gerarchico non richiede un numero prefissato di cluster; tuttavia, in alcune applicazioni, si cerca una partizione di cluster disgiunti come quelli dei cluster piani. Si possono utilizzare un certo numero di criteri per determinare il punto di taglio; nel seguito si farà uso della condizione di taglio ad un specifico livello di dissimilitudine: ad esempio, si taglierà il dendrogramma al livello $t = 0.3$ se si vuole una clusterizzazione con una minima combinazione di dissimilitudine pari a $t = 0.3$. Nel clustering gerarchico si possono sfruttare vari criteri per decidere come effettuare la fusione tra cluster:

1. Nel clustering di tipo *single linkage*, la similitudine di due cluster, è la similitudine dei loro membri più simili. Questo criterio è locale, infatti si prendono in considerazione solo l'area dove due cluster sono vicini l'uno all'altro.

2. Nel clustering di tipo *complete linkage*, la similitudine di due cluster è la similitudine dei loro membri più dissimili. Il criterio non è locale; l'intera struttura del cluster può influenzare le decisioni di fusione.

3. Nel clustering di tipo *average linkage*, la similitudine di due cluster è la similitudine tra la media delle similitudini tra tutti i membri.

Si utilizzerà questo criterio in LSA avvalendosi di quanto dimostrato in [50].

Il clustering di tipo *single linkage* presenta il problema delle *chain* (*catene*). Essendo il criterio locale infatti, elementi a due a due simili, causano l'unione a catena di tutti gli elementi in un singolo cluster [9]. D'altra parte, il clustering di tipo *complete linkage* soffre del problema di prestare troppa attenzione agli elementi distanti, ovvero elementi che non rientrano bene nei cluster formati [9]. Per quanto riguarda la complessità temporale, una implementazione nativa degli algoritmi è $\mathcal{O}(n^3)$; in [9] vengono fornite due implementazioni più efficienti per gli algoritmi delle due tipologie di cui sopra, $\mathcal{O}(n^2 \log(n))$ per l'algoritmo *complete linkage* e $\mathcal{O}(n^2)$ per l'algoritmo *single linkage*. L'algoritmo utilizzato, l'average linkage, presenta una complessità uguale al complete linkage. Questi tre algoritmi trovano una implementazione nella libreria scipy che si è utilizzata per effettuare il clustering di tipo gerarchico.

Capitolo 6

Realizzazioni sperimentali e valutazione

"Il test di un programma può essere usato per mostrare la presenza di bug, ma mai per mostrare la loro assenza."

Edsger Wybe Dijkstra

In questo capitolo si presenteranno i risultati ottenuti dallo studio. Riprendendo quanto detto nel primo capitolo, i parametri secondo cui verranno valutati i tre algoritmi sono i seguenti:

1. Numero di falsi negativi,
2. Numero di gruppi formati contenenti varianti polimorfiche dello stesso malware (consistenza),
3. Precisione,
4. Richiamo.

Il capitolo inizierà con una presentazione del dataset coinvolto nello studio, proseguendo con una descrizione della complessità computazionale dei tre algoritmi che servirà a delineare in maniera rigorosa il perchè dell'efficienza dell'algoritmo LSH [65] rispetto all'algoritmo LSA [10, 37, 41, 51] e al calcolo delle similitudini senza approssimazioni (Similitudini Coseno: SC nel seguito). Seguiranno poi delle considerazioni sulle tempistiche di esecuzione sul dataset, per concludere il capitolo con la presentazione dei risultati ottenuti sulle varie metriche.

6.1 Dataset coinvolto nello studio

Il dataset è stato estrapolato dal sito virus total [12] mantenendo una composizione paritaria per i motivi descritti nel capitolo tre; avremo pertanto 1000 campioni suddivisi in 10 classi dalla classificazione del malware effettuata da virus total. Avendo ottenuto un alto ranking, il dataset risulterà in istanze di cui è alta la probabilità che appartengano alla classe a cui sono state associate. Si presenta di seguito la classificazione e la numerazione dei campioni estratti, per i quali l'intervallo fra parentesi rappresenta la numerazione associata ai campioni:

1. *Zbot (*0-99*)*,
2. *Sinowal (*100-199*)*,
3. *Beagle (*200-299*)*,
4. *Conficker (*300-399*)*,
5. *Fujacks (*400-499*)*,
6. *Msnpass (*500-599*)*,
7. *Mydoom (*600-699*)*,
8. *Sillyfdc (*700-799*)*,
9. *Virut (*800-899*)*,
10. *Ackantta (*900-999*)*.

Nel dataset avremo dunque malware dalle più svariate funzionalità e costituito da molteplici varianti. I campioni sono stati successivamente inviati ad Anubis per la generazione dei report comportamentali in formato *xml*. Alcuni campioni non sono stati analizzati per differenti motivi da Anubis e si è quindi provveduto ad analizzarne altri colmando i vuoti. I report così prodotti sono stati successivamenti parsati, con l'analizzatore sintattico descritto nel capitolo precedente, ottenendo un'insieme di caratteristiche di tutti i malware pari a 12388. La matrice malware-caratteristiche, binaria e sparsa, avrà dunque dimensione 12388×1000.

6.2 Performance temporale asintotica

In questa sezione si presentano le complessità computazionali dei tre algoritmi relativamente ai passaggi seguenti:

1. Preprocessamento della matrice caratteristiche-malware,
2. Calcolo delle misure di distanza dei campioni,
3. Clustering.

6.2.1 Preprocessamento della matrice caratteristiche-malware

Gli unici due algoritmi che prevedono questa fase di preprocessamento iniziale sono gli algoritmi LSA e LSH. Si è già evidenziato infatti come l'algoritmo SC non effettui nessuna approssimazione dei vettori della matrice.
Partendo dall'algoritmo LSH, si ottiene che la complessità di questo stadio è pari a $\mathcal{O}(nlkd)$. Il parametro d indica il numero medio di caratteristiche di un campione, l indica il numero di volte che si ripete il calcolo dei min hash, k è la lunghezza in bit della firma ed n il numero di campioni. Tale valore è determinato dal fatto che il calcolo dei min hash prevede di scansionare le righe della matrice seguendo le k permutazioni al fine di trovare la prima colonna, per ogni colonna n, contenente un 1, il tutto eseguito per l volte. Mediamente le righe della matrice verranno valutate d volte. Si ricorda inoltre che le coppie di campioni che hanno firme uguali, avendo un valore di similitudine superiore a t, vengono immesse nell'insieme S.
L'algoritmo LSA in questo primo stadio compie la decomposizione ai valori singolari. L'algoritmo utilizzato per la SVD [17, 28, 43] ha una complessità computazionale di $\mathcal{O}(sz)$ dove s indica il numero di celle della matrice sparsa con valori reali diversi da 0 e z il numero di concetti dello spazio vettoriale della matrice troncata. La libreria utilizzata è infatti la SVDLIBC [17] che implementa nella routine LAS2 l'algoritmo Lanczos con la ortogonalizzazione selettiva [43].

6.2.2 Calcolo delle misure di distanza tra tutti i campioni

In LSH questo passaggio viene effettuato calcolando la distanza Jaccardi tra tutte le coppie presenti in S in $\mathcal{O}(ncd)$ ed il sorting delle coppie in S in $\mathcal{O}(nc \log(nc))$ [65]. Il parametro n indica il numero dei campioni, d il numero medio di caratteristiche e c la cardinalità massima dei cluster ottenuti eseguendo l'algoritmo. Dal momento che $\mathcal{O}(ncd)$ domina la componente $\mathcal{O}(nc \log(nc))$, si ottiene che la complessità dello stadio è pari a $\mathcal{O}(ncd)$. Il caso pessimo si presenta eseguendo il calcolo delle distanze su campioni identici. Alla fine del procedimento la dimensione massima dell'unico cluster ottenuto sarebbe pari a $c = n$. Si nota tuttavia che nei dataset normalmente utilizzati c è di cardinalità molto inferiore rispetto ad n. Per quanto riguarda il sorting delle coppie in S, dal momento che viene utilizzato un algoritmo di

sorting rapido (mergesort) che lavora in $\mathcal{O}(z \log(z))$ e che tale lavoro va fatto per la cardinalità di S che vale $n \times c$, si ottiene la complessità $\mathcal{O}(nc \log(nc))$. Quindi il processo va ripetuto per $z = n \times c$ volte nel caso pessimo. Il procedimento utilizzato da LSA invece risulterà di complessità inferiore se il parametro z sarà inferiore al parametro d di LSH dal momento che, lavorando su uno spazio vettoriale di dimensione z, si dovrà effettuare il prodotto scalare tra il vettore colonna di dimensione $1 \times z$ e la matrice di dimensioni $z \times n$ in $\mathcal{O}(nz)$. Il tutto ripetuto per gli n vettori. Di conseguenza la complessità vale $\mathcal{O}(n^2 z)$. In SC infine, questo passaggio ha ordine di grandezza $\mathcal{O}(n^2 m)$. Evidentemente l'algoritmo SC, essendo frutto della scelta estrema nel trade-off risulta l'algoritmo avente complessità computazionale maggiore in questo stadio.

6.2.3 Clustering

L'ultimo passo è la creazione dei cluster a partire dalla matrici di distanza ottenute dai tre algoritmi. La complessità della fase finale di creazione del cluster agglomerativo a partire dalla matrice delle similitudini Coseno in LSA e dalla matrice delle similitudini Jaccardi in LSH è pari a $\mathcal{O}(n^2)$ per la clusterizzazione single linkage di LSH ed SC, mentre $\mathcal{O}(n^2 \log(n))$ per quella average linkage di LSA [9]. Si specifica che il parametro n in LSH è inferiore rispetto al numero totale dei campioni perchè dalle fasi precedenti ne viene selezionato solo un sottoinsieme, ed in particolare il sottoinsieme di quelli aventi valore di similitudine superiore ad $1 - t$.

Tabella 6.1: Performance temporale asintotica

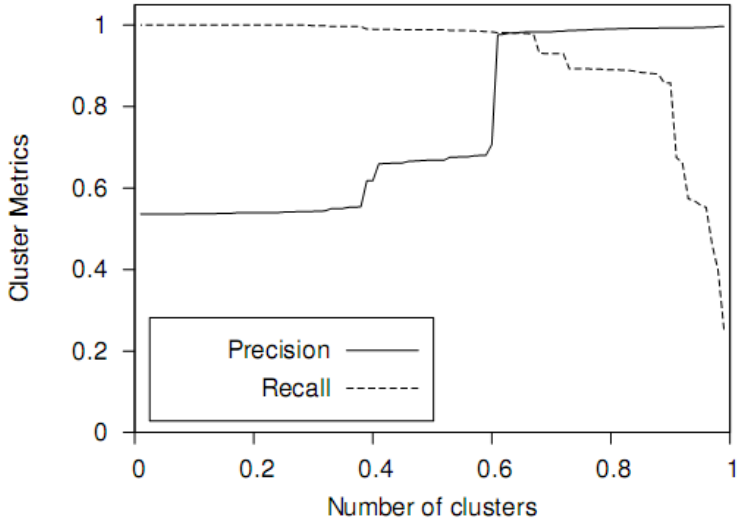
Passo algoritmo	LSH	LSA	SC
Preprocessamento	$\mathcal{O}(nlkd)$	$\mathcal{O}(sz)$	-
Distanze	$\mathcal{O}(ncd)$	$\mathcal{O}(n^2 z)$	$\mathcal{O}(n^2 m)$
Clustering	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 \log(n))$	$\mathcal{O}(n^2)$
Globale asintotica	$\mathcal{O}(n^2 d)$	$\mathcal{O}(nmz)$	$\mathcal{O}(n^2 m)$

6.3 Valutazioni circa i cluster prodotti

In questa sezione si presentano i risultati ottenuti dallo studio. Prima di procedere con l'esposizione, si vogliono chiarire dei punti relativi al calcolo delle metriche. Si sono scelti anzitutto dei valori di t, parametro che regola il livello di taglio in fase di clustering, che massimizzano contemporaneamente i valori di precisione e richiamo dei cluster formati per il singolo algoritmo.

Tale parametro può assumere dei valori compresi tra 0 e 1 e la sua scelta risulta in una precisa composizione dei cluster. In particolare, tagliare il dendrogramma ad un determinato valore t, permette di avere un insieme di cluster i cui membri sono simili almeno al valore $1 - t$. Si ricorda inoltre che gli algoritmi LSA e SC producono dei cluster gerarchici completi, ovvero producono tutte le possibili clusterizzazioni per tutti i valori di t. In maniera differente, LSH produce già il cluster gerarchico a livello di soglia superiore a $1 - t$.
Per quanto concerne l'algoritmo LSH, si è trovato un riscontro con quanto affermato in [65] circa il valore del parametro t che massimizza la precisione ed il richiamo. Tuttavia, rispetto ai risultati ottenuti in [65], si sono ottenuti dei valori più bassi per quanto riguarda il valore massimo ottenibile del richiamo. La motivazione più plausibile è la scelta di differenti campioni e la diversa distribuzione del dataset, come già evidenziato nel capitolo tre. Si ricorda inoltre che si sarebbe potuto creare un cluster gerarchico completo anche in LSH. Per far ciò sarebbe bastato eseguire l'algoritmo con differenti valori dei parametri l e k in modo tale da ottenere un valore di t corrispondente sempre differente e quindi ricostruire approssimativamente il cluster gerarchico completo campionando su valori differenti discreti del parametro t. Guardando tuttavia l'andamento della curva che rappresenta la variazione della precisione e del richiamo in funzione della scelta del parametro t presentata in [65] e che si riporta qui di seguito, risulta evidente che scelte di valori del parametro t troppo basse o troppo alte inficiano notevolmente la precisione o il richiamo. Si è deciso pertanto di eseguire il confronto soltanto per il valore di soglia che massimizza le due metriche nei tre algoritmi.

Figura 6.1: Metriche di precisione e richiamo in funzione di t



Si nota inoltre che l'algoritmo in [65] non è deterministico, ovvero non presenta lo stesso output a fronte dello stesso input. Il motivo è la casualità nella scelta delle permutazioni delle righe nell'algoritmo è del tutto casuale e potrebbe cambiare i cluster prodotti dall'algoritmo in una successiva esecuzione. Per clusterizzare la matrice delle distanze LSA infine, si è utilizzato l'algoritmo di clustering gerarchico denominato "average linkage" che calcola le distanze tra cluster come la media delle distanze tra i membri di cluster differenti. Si è scelta questa modalità di calcolo delle distanze tra cluster poiché si vuole che i campioni si aggreghino al cluster solo ed esclusivamente se tutte le distanze di tutti i campioni da aggregare hanno come distanza media tra di loro il valore di similitudine $1 - t$. La motivazione risiede nel fatto che se si scegliesse ad esempio come algoritmo gerarchico il "single linkage", che sceglie invece la funzione di minimizzazione opposta al "complete linkage", si otterrebbe l'aggiunta al cluster di campioni in cui non tutti i valori di distanza rispetto ai membri del cluster risultano inferiori ad $1 - t$. Di conseguenza il cluster sarebbe troppo permissivo rispetto all'aggiunta di nuovi membri e si perderebbe di precisione, altro motivo a supporto della decisione a favore della scelta dell'average linkage.

6.3.1 Numero di falsi negativi

Il numero di falsi negativi rappresenta i campioni che non sono stati immessi in nessun cluster da parte dell'algoritmo. Nel caso di LSA si è ottenuto un

valore pari a 10, di 4 per SC e di 198 per LSH. Questo dimostra chiaramente che molti campioni, sebbene individuati dall'analisi statica come malware, non sono stati associati a comportamenti dannosi e quindi non sono stati rilevati dall'analisi LSH. Da notare ulteriormente che il numero di falsi negativi aumenterebbe potenzialmente con una scelta diversa del parametro t dal momento che stiamo valutando l'algoritmo nelle sue prestazioni ottimali.

6.3.2 Precisione e Richiamo

La tabella seguente presenta i valori di t scelti per i tre algoritmi con i rispettivi valori di precisione e richiamo:

Tabella 6.2: Valori del parametro t scelti

Algoritmo	Valore di t	Precisione	Richiamo
LSH	0.3	0.83	0.50
LSA	0.35	0.82	0.66
SC	0.25	0.83	0.66

Per quanto riguarda LSH, si è ottenuto un valore basso del parametro richiamo. Ricordando che questo parametro rappresenta la frazione di istanze rilevanti che sono state prelevate, ne risulta che una percentuale considerevole di istanze non sono state immesse in cluster corrispondenti alle famiglie come classificate da virus total ma in cluster differenti. Si sono ottenuti dei valori di richiamo superiori per gli algoritmi LSA ed SC. Molto importante notare che le metriche sono calcolate sui campioni che sono stati effettivamente clusterizzati. Nel paragrafo precedente si è visto che in LSH il numero di falsi negativi è elevato e le metriche vanno calcolate sulla differenza tra il numero totale di campioni ed il numero di falsi negativi. La precisione di conseguenza è alta perchè è calcolata su un numero molto inferiore rispetto al totale dei campioni.

6.3.3 Consistenza

La tabella seguente mostra la suddivisione in famiglie comportamentali individuate dall'esecuzione dei tre algoritmi sul dataset. Si evidenzia il numero di cluster formati rispetto alle famiglie individuate da virus total; viene indicata la percentuale di campioni aggregati in qualche cluster ed il numero di cluster in cui i 100 campioni costituenti la percentuale sono stati aggregati.

Tabella 6.3: Flat Cluster a livello di distanza $t = 0.3$ per LSH, $t = 0.35$ per LSA e $t = 0.3$ per SC

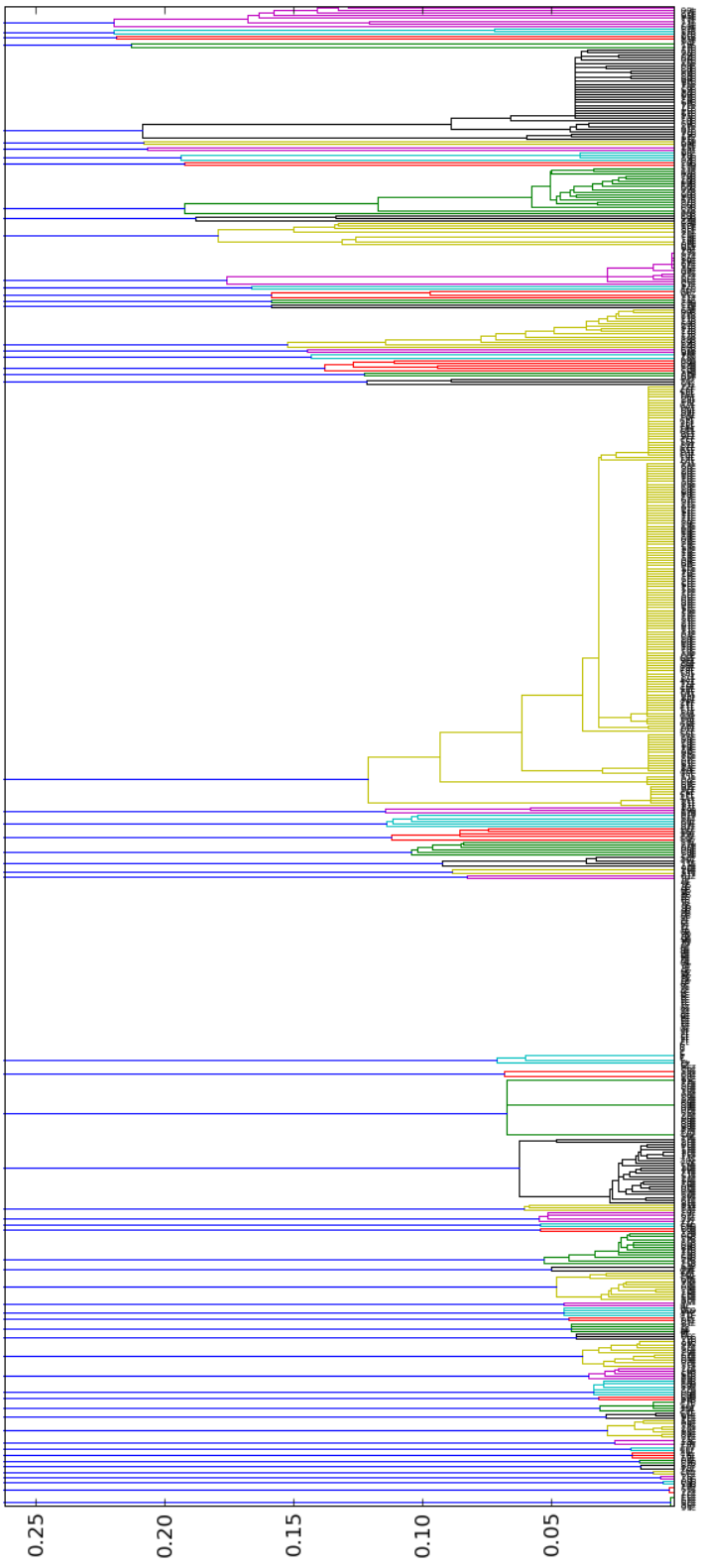
Famiglia	% LSA	# cluster	% LSH	# cluster	% SC	# cluster
Zbot	94%	5	83%	5	94%	5
Sinowal	94%	4	79%	7	94%	3
Beagle	90%	10	76%	15	89%	10
Conficker	99%	1	99%	4	99%	2
Fujacks	81%	3	50%	12	80%	5
MSNPass	87%	3	93%	10	85%	4
My doom	77%	2	50%	7	77%	3
SillyFDC	73%	9	53%	18	71%	5
Virut	95%	2	79%	11	95%	2
Ackantta	87%	6	53%	10	87%	7

Il cluster prodotto dai tre algoritmi presenta delle differenze significative. Si nota subito l'incidenza del numero di falsi negativi sulle percentuali di malware clusterizzato da LSH. Ci si sarebbe aspettato che i campioni, essendo tutti presi da un dataset costituito da solo malware, fossero stati clusterizzati in qualche modo da LSH. Questo perchè il dataset di partenza è stato estrapolato da un sottoinsieme dei campioni di Virus Total ad alto ranking. Un'ispezione manuale dei report, ha permesso inoltre di consolidare l'affermazione poichè report perfettamente identici sono stati messi in cluster differenti; report identici dovrebbero avere un livello di similitudine pari a 1, cosa che avviene di fatto in SC, mentre in LSH è possibile che non siano clusterizzati assieme oppure che uno sia immesso nell'insieme S e l'altro non preso in considerazione nella seconda fase di analisi, risultando pertanto in un livello di distanza superiore a 0.3 sebbene i report siano perfettamente identici. Si ribadisce inoltre che la scelta del parametro $l = 90$ effettuata in LSH ha provocato un aumento significativo del numero di falsi positivi a discapito proprio del numero di falsi negativi. La percentuale molto bassa del numero di falsi negativi degli algoritmi non efficienti, LSA ed SC, ha permesso di clusterizzare tutto il malware. Questa è la prima differenza sostanziale. Il parametro $z = 250$ scelto in LSA, ha permesso infatti di costruire un'approssimazione della matrice malware-caratteristiche molto precisa, tralasciando una percentuale trascurabile della varianza. Si è ottenuta pertanto un'approssimazione molto vicina alla matrice reale utilizzata dall'algoritmo SC. I risultati molto simili ottenuti dai 2 algoritmi sono l'ulteriore conferma della capacità di costruire un'approssimazione sempre

più vicina alla matrice reale scegliendo dei valori del parametro z in LSA calibrati sulla dimensione del problema.
Si passa ora a commentare i cluster ottenuti. Il numero di cluster prodotti per famiglia, confrontato con la suddivisione prodotta da Virus Total, è risultato molto alto per l'algoritmo LSH. L'algoritmo LSH ha ottenuto pertanto un alto livello di consistenza [50], ovvero un alto numero di classi di cardinalità piccola. Gli algoritmi LSA ed LSH sono riusciti invece a clusterizzare in poche classi le varianti del campione di malware da cui si sono generate, aderendo meglio alla ground truth. La problematica principale di valori alti di consistenza è che si ottiene un numero alto di famiglie comportamentali allo stesso modo di come si ottiene un numero alto di etichette da parte degli analizzatori statici per varianti polimorfiche dello stesso malware. L'analisi di malware dovrebbe invece riuscire a categorizzare in poche classi le varianti polimorfiche dello stesso campione di malware.
Si è dimostrato pertanto come l'utilizzo di approssimazioni abbia dei risvolti notevoli sui cluster prodotti e di conseguenza sulla categorizzazione del malware che viene fatta da un algoritmo che predilige l'efficacia rispetto alla completezza d'analisi. Una strategia ancora più aggressiva per LSH è quella utilizzata in [27], dove anziché effettuare il calcolo delle similitudini Jaccardi dei campioni contenuti in S, si calcola invece la distanza Jaccardi sulla firma LSH. Questo approccio produce però un incremento prestazionale ulteriore a forte discapito delle metriche analizzate.
Si conclude lo studio con osservazioni sui dendrogrammi prodotti dai tre algoritmi.

6.3.4 Dendrogramma LSH

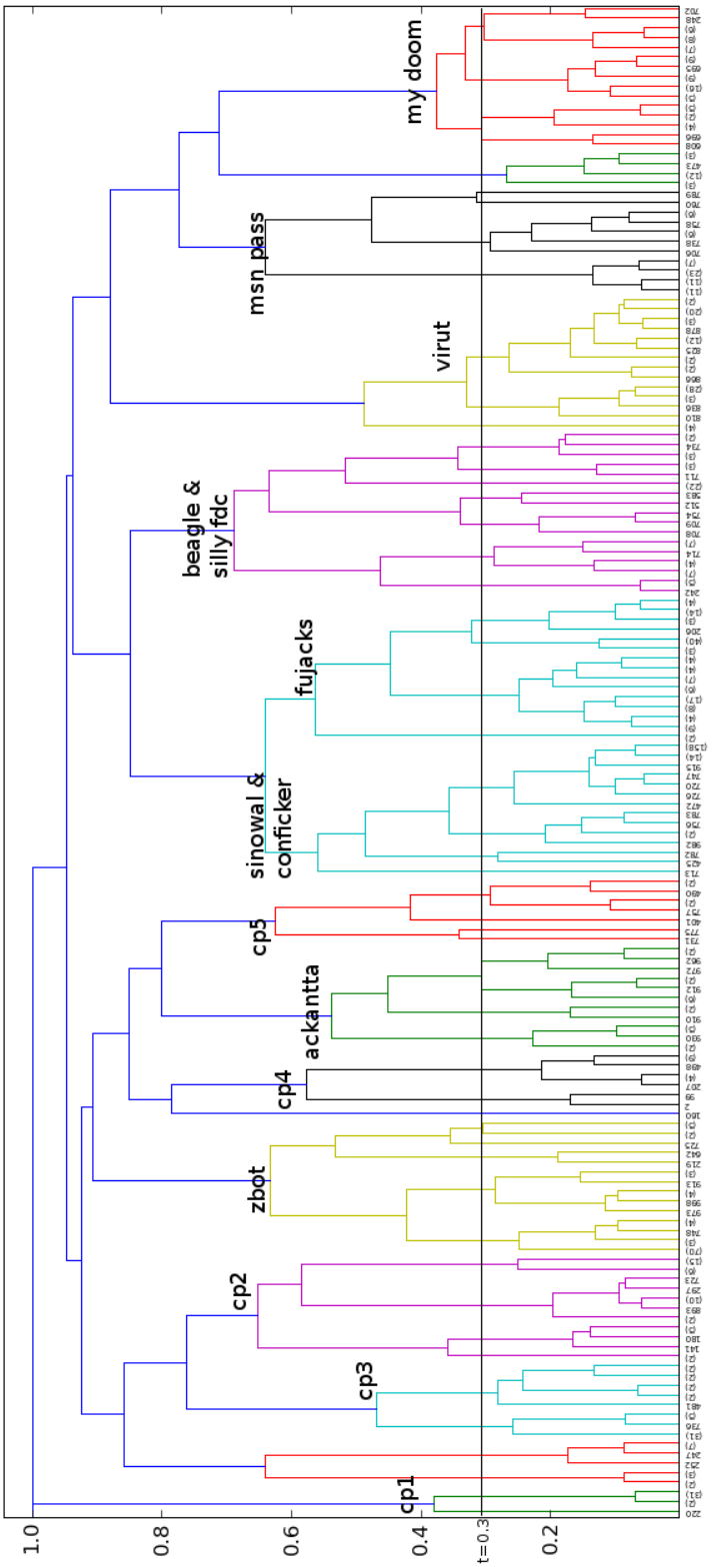
Figura 6.2: Dendogramma LSH



In questa sottosezione si presenta il dendrogramma relativo alla clusterizzazione prodotta dall'algoritmo LSH. In ascissa si ha il valore di distanza, in ordinata i campioni di malware. Si nota innanzitutto come il cluster presenti un taglio in corrispondenza del valore di distanza $t = 0.3$ scelto per l'analisi. Si è già discusso nei capitoli precedenti infatti come l'algoritmo LSH, per sua natura, non consenta di costruire il cluster gerarchico completo per tutti i valori del parametro t. Il cluster risulta molto frammentato e non è possibile trovare un sottoalbero del dendrogramma che contenga molti campioni appartenenti alla stessa famiglia individuata in Virus Total. Si vedrà infatti successivamente come i dendrogrammi LSA e SC siano facilmente mappabili sulle famiglie di malware della ground truth. Non è possibile individuare potenziali analogie tra famiglie di malware e separare i casi discussi nel capitolo 3, riguardo le problematiche di Anubis, dai casi di clusterizzazione utile.

6.3.5 Dendrogramma LSA

Figura 6.3: Dendogramma LSA



In questa sottosezione si presenta il dendrogramma relativo alla clusterizzazione prodotta dall'algoritmo LSA. Si vede chiaramente la composizione delle famiglie e le similitudini individuate tra le famiglie *beagle* e *sillyfdc* da un lato, e tra *conficker* e *sinowal* dall'altro ad un livello di similitudine medio basso (distanza: $t = 0.7$); queste ultime due famiglie di malware presentano infatti alcune similitudini, tra le quali l'utilizzo della tecnica del "domain flux" [18]. Con questa tecnica, ogni bot periodicamente (ed in maniera indipendente) genera una lista di dominii che crea. Il bot successivamente li contatta uno dopo l'altro. Il primo host che manda una reply identificandosi come un server C&C (server comanda e controlla) è considerato legittimo, fino al momento in cui la generazione periodica di dominii successiva viene cominciata. Ne risulta che le due famiglie di malware presentano attività simili che sono state ben individuate da LSA e SC.
Viene presentata di seguito una panoramica sui casi particolari di cluster formati in merito alle problematiche di Anubis che si sono discusse nel capitolo 3.

1. caso particolare 1: Tutti i membri del gruppo sono stati avviati ma il packer utilizzato per comprimere il virus è sconosciuto ad Anubis.

2. caso particolare 2: I campioni appartenenti hanno eseguito pochissime attività sul sistema operativo. Nel caso specifico di questi campioni, viene controllato se nel sistema è in esecuzione remote desktop in modalità application server. Tale test ritorna esito negativo, il sistema non è vulnerabile e quindi il campione cessa la propria attività.

3. caso particolare 3: Gli eseguibili si occupano di modificare il file *autorun.inf* al fine di eseguire potenzialmente altre azioni mediante codice in esso scritto. Il comportamento rilevato da Anubis è identico.

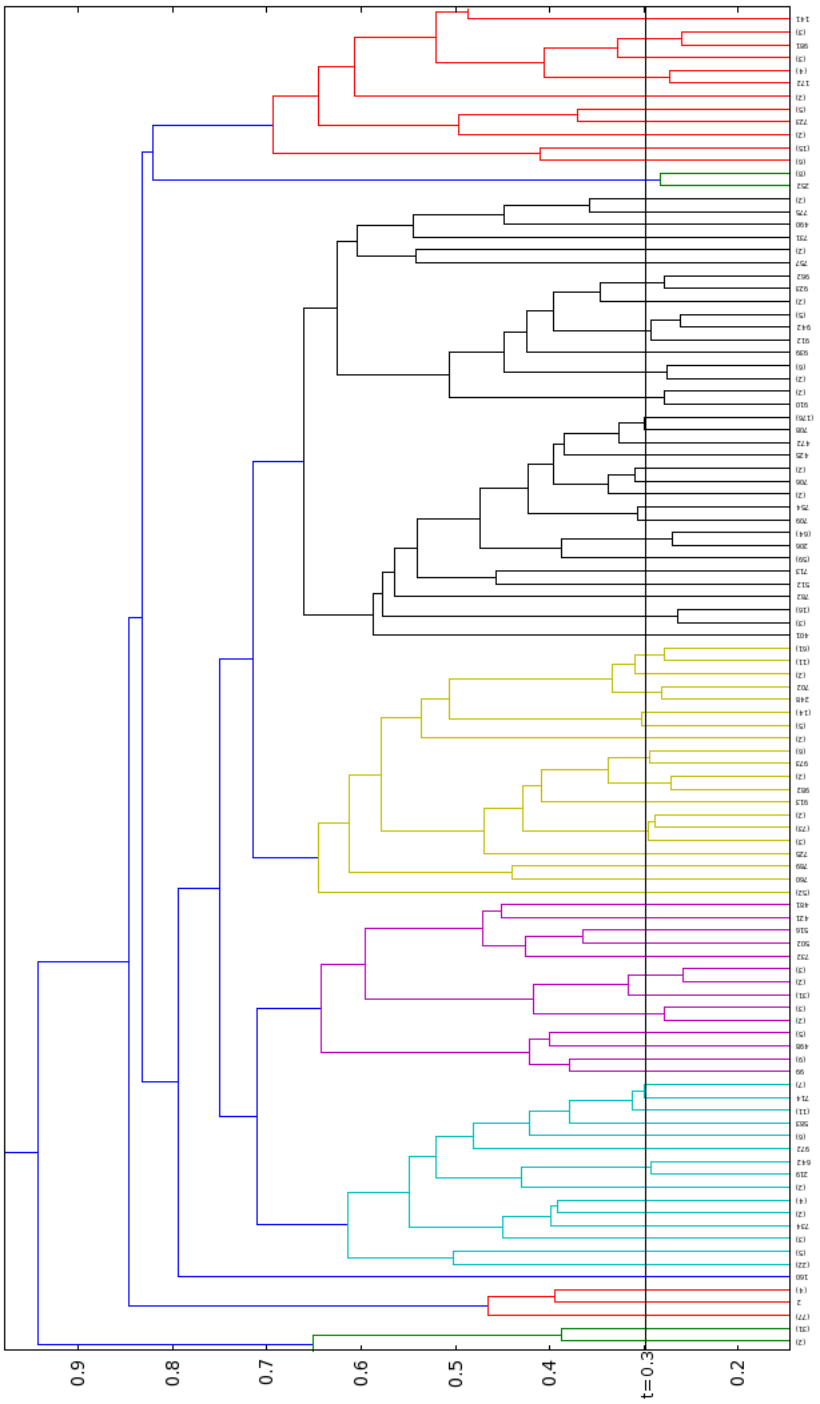
4. caso particolare 4: L'eseguibile modifica la chiave di registro relativa all'esecuzione automatica di programmi all'avvio di Windows; copia se stesso nell'eseguibile che verrà poi lanciato effettivamente e che contiene il vero codice maligno.

5. caso particolare 5: L'eseguibile si lancia mediante il servizio *svchost.exe* di Windows.

Si noti che LSA ha individuato la similitudine tra i campioni relativi a queste problematiche perchè agli occhi dell'analisi sono campioni perfettamente identici o molto simili, e comunque sono stati isolati dal resto dei campioni analizzati.

6.3.6 Dendrogramma SC

Figura 6.4: Dendogramma SC



Il dendrogramma SC infine, è del tutto simile a quello prodotto da LSA. Anche qui sono ben individuabili le famiglie di malware ad un livello di similitudine sufficientemente alto (distanza: $t = 0.3$). Le differenze con l'algoritmo LSH sono del tutto simili a quelle già evidenziate con LSA. Risulta pertanto evidente come il cluster formato dagli algoritmi che prediligono la precisione nel trade-off tenda a costruire poche classi raggruppando il più possibile i campioni appartenenti alla stessa famiglia della ground truth. L'algoritmo SC, come prevedeibile, non ha aggiunto molto al cluster già formato con LSA per il fatto che quest'ultimo ha costruito una buona approssimazione della matrice di partenza e quindi non ha rivoluzionato, contrariamente ad LSH, il cluster formato.

Capitolo 7

Direzioni future di ricerca e conclusioni

In questo capitolo finale si riassumono le conclusioni a cui si è giunti con lo studio e si forniscono degli spunti per proseguire il lavoro svolto, prendendo in considerazione i possibili miglioramenti apportabili alla sandbox utilizzate e ad alcuni aspetti di analisi.

7.1 Conclusioni

7.1.1 Lavoro svolto

Lo studio ha consentito di quantificare precisamente le perdite derivate dall'utilizzo di un algoritmo efficiente nell'ambito del clustering di malware. Un algoritmo di clustering di malware è composto da tre fasi: la generazione di report comportamentali da parte dell'esecuzione del dataset in una sandbox, il processamento delle attività del malware precedente al calcolo delle similitudini ed il clustering vero e proprio. Si valutano infine i risultati confrontandoli con quanto evidenziato da molteplici prodotti antivirus concordanti nell'assegnare un'etichetta al malware, la ground truth. Lo studio ha posto risalto sulla qualità del cluster prodotto a parità di misura di distanza utilizzata, la distanza coseno, di dataset e di ground truth. Il motivo per cui è stata effettuata questa scelta, è la necessità di non alterare i risultati dell'analisi cambiando i parametri suddetti, i quali avrebbero sicuramente impattato sull'oggetto di analisi: il processamento delle attività del malware precedente al calcolo delle similitudini. Si è voluto verificare come l'utilizzo di algoritmi che effettuano approssimazioni in questa fase abbia dei risvolti

sulla qualità finale del clustering. Gli algoritmi implementabili in ambito di clustering di malware, risultano più o meno efficienti in base alle scelte effettuate in questa fase e di conseguenza si pongono in ben determinati punti nella curva che rappresenta il trade-off tra tempo di esecuzione e precisione nel calcolo delle distanze.
Sono stati quindi sviluppati tre algoritmi, LSA, LSH e SC che effettuano una scelta differente nel trade-off. SC è l'algoritmo che non effettua nessuna approssimazione nel calcolo delle distanze coseno tra tutti i campioni di malware, LSH esegue una funzione di hash sull'insieme delle caratteristiche di un malware creandone una firma, LSA crea l'approssimazione della matrice di partenza che mantiene le dimensioni a varianza più grande rimuovendo le altre. LSA è di conseguenza un algoritmo che risulta in un'approssimazione meno invasiva nel calcolo delle misure di distanze rispetto a LSH (dato che si è scelto un valore del parametro z tale per cui l'errore di approssimazione dell'algoritmo è minimo).
Si è visto come una precisa scelta nel trade-off vada a discapito di alcune metriche che sono state analizzate nello studio al fine di produrre un'analisi qualitativa del clustering: la precisione, il richiamo, la consistenza ed il numero di falsi negativi. I risultati ottenuti dall'analisi di queste metriche hanno messo in risalto la tendenza, da parte di LSH, ad aumentare fortemente il numero di falsi negativi ed il richiamo scegliendo dei valori dei parametri dell'algoritmo che ne migliorano la precisione ed il numero di falsi positivi (i parametri l e k). Questo ha delle ricadute, oltre che sul numero di falsi negativi, sulla consistenza. L'alto livello di consistenza dimostrato da LSH, crea una forte segmentazione nella composizione del cluster rispetto alle famiglie derivate dalla ground truth. Questa caratteristica non è stata invece evidenziata nell'esecuzione degli algoritmi meno efficienti LSA ed SC.

7.1.2 Deduzioni derivate dallo studio

Un algoritmo efficiente è progettato in modo tale da risultare il più possibile scalabile, data la necessità di avere dei metodi veloci per effettuare una clusterizzazione in breve tempo su un quantitativo enorme di campioni di malware che devono essere processati ogni giorno. Per poter far ciò è necessario tralasciare nell'analisi determinate componenti, di cui lo studio ha evidenziato l'esistenza, tra cui quella di maggior impatto, il numero di campioni individuati. Un algoritmo che effettua un'analisi più precisa riesce sicuramente a meglio individuare ed a clusterizzare campioni di malware. Si possono scegliere soluzioni che si adagiano in una via di mezzo tra i due estremi, precisione massima ed efficienza massima. Lo studio ha permesso

di individuare una soluzione che si discosta dall'estremo di precisione massima di un quantitativo definito a meno di un parametro, l'algoritmo LSA. Risulta quindi possibile scegliere soluzioni, frutto di un compromesso tra i due estremi, che permettano di ottenere risultati più o meno precisi in più o meno tempo a seconda delle esigenze.
L'analisi dinamica di malware infine è sicuramente un ambito che ha risolto e superato i problemi presenti nell'analisi statica; tuttavia richiede attenzione nel valutare in maniera razionale e scientifica i risultati prodotti. L'ambito dell'analisi di malware è infatti un terreno difficile per gli analisti che si devono necessariamente scontrare con le evoluzioni apportate dai programmatori di malware. Anche l'analisi dinamica infatti risente di determinate problematiche sulle quali bisogna sicuramente porre attenzione e sforzi di ricerca. Con questo studio si è dato contributo in merito ad alcuni aspetti importanti che hanno ricadute sui cluster prodotti da un algoritmo efficiente.

7.2 Sviluppi futuri

Le possibili ottimizzazioni sulla clusterizzazione ottenuta devono necessariamente passare da un miglioramento di Anubis al fine di eliminare le problematiche derivate dall'utilizzo della sandbox, di cui si è parlato nel capitolo 2. Infatti, i report prodotti dalla sandbox in alcuni casi risultano incompleti ed in altri non è possibile eseguire l'analisi. Si potrebbe pensare di inglobare nella sandbox la possibilità di esecuzione multipath, così come descritta in [39]. Un'alternativa è l'utilizzo di altre sandbox, che produrrebbero dei report comportamentali sicuramente almeno parzialmente differenti rispetto a quelli prodotti da Anubis. Infine, è possibile preprocessare le attività dei malware presenti nei report in modo tale da filtrare elementi quali l'utilizzo di nomi random, che qualificano come differenti due attività che potenzialmente sarebbero uguali, e processare il traffico di rete anche con altri strumenti che effetuino un'analisi approfondita sui pacchetti scambiati.
Riguardo il sistema operativo utilizzato per l'analisi, si è detto che Anubis, come altre sandbox, utilizza il sistema operativo Windows XP service pack 2 per condurre l'analisi. Si è già commentato in precedenza come questo risulti in un forte vincolo, dal momento che determinati campioni di malware sfruttano ben determinate vulnerabilità di sistema operativo. Si pone dunque il problema che l'utilizzo di una versione di sistema operativo già aggiornata rende impossibile un'analisi precisa del malware. Si potrebbe pensare di utilizzare diverse macchine virtuali dotate di aggiornamenti differenti per condurre un'analisi che dia maggiore probabilità di successo nel far eseguire al malware determinati path.

Un altro aspetto su cui sono sicuramente possibili ottimizzazioni è l'utilizzo di molteplici misure di distanza nell'analisi; alcuni studi sono stati fatti in merito alla questione [14, 26], mettendo in evidenza i pregi ed i difetti delle stesse. L'inesistenza di una misura di distanza ottima per tutte le casistiche rende infatti necessario vagliare quale si vuole utilizzare nell'analisi in base a molteplici metriche tra le quali: complessità computazionale, sensibilità alla variazione dell'ordine delle attività del malware, appropriatezza della misura. Risulta dunque chiaro che l'utilizzo di più metriche produca dei risultati migliori rispetto all'utilizzo di una singola misura di distanza.

Bibliografia

[1] Anubis. http://anubis.seclab.tuwien.ac.at.

[2] Bit blaze. http://bitblaze.cs.berkeley.edu/.

[3] Conficker analysis. http://web17.webbpro.de/index.php?page=analysis-of-conficker.

[4] Cwsandbox. http://mwanalysis.org/.

[5] Gensim. http://nlp.fi.muni.cz/projekty/gensim/.

[6] Norman sandbox. http://www.norman.com/security_center/security_tools/.

[7] Numpy. http://numpy.scipy.org/.

[8] Scipy. http://www.scipy.org/.

[9] Stanford. http://nlp.stanford.edu/IR-book/pdf/17hier.pdf.

[10] Svdlibc. http://tedlab.mit.edu/~dr/SVDLIBC.

[11] Threat expert. http://www.threatexpert.com/submit.aspx.

[12] Virus total. http://www.virustotal.com.

[13] Broder A., Charikar M., Frieze A., and Mitzzenmacher M. Min-wise independent permutations. *STOC*, 1998.

[14] M. Apel, C. Bockermann, and M. Meier. Measuring similarity of malware behavior. In *LCN*. IEEE Computer Society, 2009.

[15] Bradford R. B. An empirical study of required dimensionality for large scale latent semantic indexing applications. In *CIKM*. ACM, 2008.

[16] M. Bailey, J.Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J.Nazario. Automated classification and analysis of internet malware. In *Recent Advances in Intrusion Detection, 10th International Symposium.* Springer, 2007.

[17] Michael W. Berry. Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, 1992.

[18] Stone-Gross Brett, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, 2009.

[19] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *In Proceedings of the 12th USENIX Security Symposium*, 2003.

[20] M. Christodorescu, S. Jha, S. A. Seshia, D. X. Song, and R. E. Bryant. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2005.

[21] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *journal of the american society for information science*, 1990.

[22] P. Ferrie. Anti-unpacker tricks 2 part seven. *Virus Bulletin*, 2009.

[23] A. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1996.

[24] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins Univ. Press, 3 edition, 1996.

[25] D. Gryaznov and J. Telafici. What a waste - the anti-virus industrie dos-ing itself. In *Proceedings of Virus Bulletin Conference.*, 2007.

[26] I. Gurrutxaga, O. Arbelaitz, J. Perez, J. Muguerza, I. J. Martin, and I. Perona. Evaluation of malware clustering based on its dynamic behaviour. In *Seventh Australasian Data Mining Conference (AusDM 2008)*. ACS, 2008.

[27] Haveliwala T. H., Gionis A., and Indyk P. Scalable techniques for clustering the web. In *WebDB (informal proceedings)*, 2000.

[28] Aswani K. and Srinivas S. Latent semantic indexing using eigenvalue analysis for efficient information retrieval. *International journal of applied mathematics and computer science*, 2006.

[29] J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 2006.

[30] R. Konrad, T. Philipp, W. Carsten, and H. Thorsten. A malware instruction set for behavior-based analysis. In *Sicherheit*, 2010.

[31] T. K. Landauer and S. T. Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge., 1997.

[32] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 1998.

[33] Todd A. Letsche and Michael W. Berry. Large-scale information retrieval with latent semantic indexing. *Inf. Sci.*, 1997.

[34] C. Linn and S. Debray. Resistance to static disassembly. In *SIGSAC: 10th ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2003.

[35] R. W. Lo, K. N. Levitt, and R. A. Olsson. Mcf: a malicious code filter. *Computers & Security*, 1995.

[36] C. F. Van Loan. Generalizing the singular value decomposition. *SIAM J. Numer. Anal.*, 1976.

[37] Brand M. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 2006.

[38] L. Martignoni, M. Christodeorescu, and S. Jha. Omniunpack: Automating the hidden-code extraction of unpack-executing malware. In *In Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2007.

[39] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007.

[40] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *ACSAC*. IEEE Computer Society, 2007.

[41] Halko N., Martinsson P. G., and Tropp J. A. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions., 2009.

[42] Indyk P. A small minwise independent family of hash functions. *SODA*, 1999.

[43] B. N. Parlett and D. S. Scott. The lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 1979.

[44] L. Peng, L. Limin, G. Debin, and K. R. Michael. On challenges in evaluating malware clustering. In *Recent Advances in Intrusion Detection, 13th International Symposium.* Springer, 2010.

[45] R. Perdisci, A. Lanzi, and W. Lee. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *ACSAC*. IEEE Computer Society, 2008.

[46] R. Perdisci, W. Leea, and N. Feamstera. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*. USENIX Association, 2010.

[47] I. V. Popov, S. K. Debray, and G. R. Andrews. Binary obfuscation using signals. In *SS'07 Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium.*, 2007.

[48] M. D. Preda, M. Christodorescu, S. Jha, and S. Debray. A semantics-based approach to malware detection. *ACM Transactions on Programming Languages and Systems*, 2008.

[49] A. Quarteroni, R. Sacco, and F. Saleri. *Matematica numerica*. Springer, 2 edition, 2000.

[50] Konrad R., Thorsten H., Carsten W., Patrick D., and Pavel L. Learning and classification of malware behavior. In *DIMVA*. Springer, 2008.

[51] Rehurek R. Fast and faster: a comparison of two streamed matrix decomposition algorithms. *CoRR*, 2010.

[52] Rehurek R. Subspace tracking for latent semantic analysis. In *Proceedings of the 33rd European Conference on Information Retrieval (ECIR)*, 2010.

[53] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. Technical report, Berlin Institute of Technology, 2009.

[54] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *ACSAC*. IEEE Computer Society, 2006.

[55] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy*, 2001.

[56] M. Sharif, A. Lanzi, J. Giffin, and W. Lee. Automatic reverse engineering of malware emulators. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009.

[57] S. J. Stolfo, K. Wang, and W. J. Li. Towards stealthymalware detection. *Malware Detection*, 2007.

[58] Symantec. Understanding virus behavior under windows nt.

[59] P. Szor. *The art of computer virus research and defense.* Addison-Wesley Professional, 2005.

[60] Lee T. and Mody J. J. Behavioral classification. In *EICAR*, 2006.

[61] Yetiser T. Polymorphic viruses - implementation, detection, and protection. `http://vx.netlux.org/lib/ayt01.html`, 1993.

[62] Bayer U., Moser A., Kruegel C., and Kirda E. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2006.

[63] Bayer U., Kruegel C., and Kirda E. Ttanalyze: a tool for analyzing malware., 2005.

[64] Bayer U., Habibi I. P., Balzarotti D., Kruegel C., and Kirda E. A view on current malware behaviors. In *LEET'09: 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats.*, 2009.

[65] Bayer U., Milani Comparetti P., Hlauscheck C., Kruegel C., and Kirda E. Scalable, behavior-based, malware clustering. In *NDSS*. IEEE Computer Society, 2009.

[66] G. Wicherski. pehash: A novel approach to fast malware clustering. In *LEET'09 Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, 2009.

Appendice A

Il manuale utente

Nell'appendice A si mostrano i comandi da eseguire sugli script che compongono l'architettura del sistema. Qualsiasi comando lanciato senza parametri mostra l'elenco delle opzioni disponibili.
Utilizzo degli script di estrapolazione delle feature dai report xml generati da Anubis:
input: Report xml, output:file di testo contenente l'elenco di tutte le attività di tutti i campioni

python FeaturesCollection.py -d O1 -w O2 P1

dove l'opzione O1 indica la directory contenente i report xml da parsare, *O*2 la directory di lavoro e P1 l'elenco degli hash MD5 dei campioni di malware
input: file di testo output dello script precedente output: Matrix.mtx, matrice Malware-Caratteristiche

python PaseHashedFeaturesNoCount.py -d O1 -w O2 -f O3 P1

dove l'opzione O1 indica la directory contenente i report xml da parsare, *O*2 la directory di lavoro, *O*3 il nome del file di testo in input e P1 l'elenco degli hash MD5 dei campioni di malware
Utilizzo dello script di dell'analisi semantica latente e calcolo delle distanze:
input: matrice caratteristiche-malware output: matrice triangolare superiore delle distanze coseno

python Gensim.py

Utilizzo dello script di clusterizzazione:
input: matrice triangolare superiore output dello script precedente output:cluster gerarchico completo e troncato al valore di similitudine t e dendrogramma

python Linkage.py P1 P2

dove il parametro P1 è il file contenente la matrice delle distanze, P2 il valore di soglia di similitudine desiderato per il taglio.

Appendice B

Listato

In appendice B, vengono mostrati stralci di codice particolarmente significativi.

```
import xml.dom as xd
import xml.dom.minidom as md

Features=set()

def esaminaFigli(figli,filename):
  for i in range(len(figli)):
    if(figli[i].nodeType==1):
       attributi=figli[i].attributes
       prefix=figli[i].tagName
       for t in range(len(ListaFeature)):
         if(pre==ListaFeature[t][0]):
           for g in range(len(ListaFeature[t])):
             postfix=figli[i].getAttribute(ListaFeature[t][g])
             prefix=prefix+' '+postfix
           Features.add(str(prefix+'\n'))
       esaminaFigli(figli[i].childNodes,filename)


Doc=md.parse('Filename.xml')
root=Doc.documentElement
childs=root.childNodes
esaminaFigli(childs,'Filename.xml')
```

Il listato presentato è relativo alla parte dell'algoritmo che effettua il parsing dei file xml generati da Anubis. Il tipo di parser utilizzato è una versione DOM alleggerita. La funzione esaminaFigli è una procedura ricorsiva che esamina tutti gli elementi dell'albero estrapolando l'insieme di comportamenti che abbiamo designato e costruindo l'unità base dell'analisi semantica latente, ovvero una stringa per ogni attività rappresentativa svolta dal

malware. Per un rapido processamento delle stringhe, esse verrano successivamente hashate e verranno utilizzate per poter calcolare la matrice Malware-Caratteristiche.

```
from gensim import corpora, models, similarities
import scipy as sp
from scipy import io,sparse
import numpy as np

corpusm = corpora.MmCorpus('Matrix.mtx')
lsi = models.LsiModel(corpusm, numTopics = 250)
corpusm = lsi[corpusm]
sms = similarities.docsim.SparseMatrixSimilarity(corpusm)
temp=[]
i=0;
for doc in corpusm:
  sims = sms[doc]
  temp.append(sims[i+1:])
  i=i+1
temp=np.array(temp,dtype=object)
np.save('LSA.npy',temp)
```

Il secondo stralcio di codice mostra come viene utilizzata la libreria *Gensim* per effettuare la decomposizione ai valori singolari. La matrice sparsa Malware-Caratteristiche (Matrix.mtx) viene importata ed utilizzata per generare il corpus. Da questo corpus viene effettuata la decomposizione ai valori singolari, dove notiamo il numero di dimensioni latenti, 250, che rappresentano la dimensione dello spazio vettoriale dei concetti. Come discusso, non abbiamo sfruttato la possibilità dell'esecuzione in cluster. Dopo aver calcolato la decomposizione, si procede al calcolo delle similitudini fra tutti i vettori dei campioni nello spazio vettoriale ridotto. Infine, ricaviamo i valori delle distanze presentando una query per stabilire la similitudine tra un campione e tutti gli altri. Eseguiamo questa operazione in un ciclo per estrapolare tutti i valori delle distanze tra tutti i campioni e memorizziamo i valori in una matrice codificata come un array *Numpy*.

```
import numpy as np
import scipy as sp
from scipy.cluster import hierarchy
from scipy.spatial import distance
import matplotlib

Z=sp.cluster.hierarchy.linkage(G,method='average',metric='cosine')
T=sp.cluster.hierarchy.fcluster(Z,t=0.3,criterion='distance')
matplotlib.pylab.figure()
sp.cluster.hierarchy.dendrogram(Z)
matplotlib.pylab.show()
```

L'ultima parte di codice è relativa alla clusterizzazione. Viene mostrata la funzione relativa al calcolo del cluster gerarchico, specificando la misura di distanza della matrice delle distanze in ingresso, ed il criterio di aggregazione. La seconda funzione calcola il flat cluster al valore di soglia $t = 0.3$. Infine, si mostrano le funzioni di calcolo del dendrogramma e di plottaggio dello stesso in *matplotlib*.