“Università del Piemonte Orientale Amedeo Avogadro”

Corso di Laurea Specialistica in
Informatica dei sistemi avanzati e dei servizi di rete

Corso di Sistemi Adattativi

Intrusion Detection System
e
Data Mining

VICINO Guido

Anno Accademico
2005/2006

Indice

Introduzione....3
Che cos'è un Intrusion Detection System?....3
Data mining sui dati di audit....7
Riunione dei dati di Audit....11
Costruzione delle feature....12
  Codifica delle feature....13
  Algoritmo di costruzione delle feature....16
Ripper e creazione delle regole di rilevamento....18
Rilevamento di anomalie basate sul payload....21
  Analisi n-gram....21
  Distanza semplificata di Mahalanobis....23
  Apprendimento incrementale....24
  Ridurre la dimensione del modello tramite raggruppamento....25
  Apprendimento non supervisionato....25
Conclusioni....26
Riferimenti:....27

Introduzione

Negli ultimi anni la sicurezza informatica e la protezione dei dati hanno ricevuto l'attenzione non solo degli esperti informatici, ma anche di tutti i semplici utilizzatori di calcolatori e reti. Il numero degli attacchi informatici a servizi e macchine disponibili in rete aumenta conseguentemente alla crescita degli utenti in Internet. Questo si traduce nella richiesta di strumenti, tecniche e persone operanti nel campo della sicurezza informatica.

Le caratteristiche di una rete o di un sistema distribuito che lo rendono vulnerabile rispetto ad un sistema indipendente sono:

- Le reti solitamente forniscono più risorse.
- Le risorse ed i servizi forniti devono essere condivisibili.
- Le politiche di protezione legate a tali sistemi devono permettere le due caratteristiche precedenti.

Quello che ci interessa studiare è quando queste caratteristiche fanno nascere vulnerabilità che possono essere sfruttate da un attaccante per danneggiare la sicurezza di un sistema informatico. Un attacco alla sicurezza di un sistema avviene quando esso subisce tentativi di intrusione o tentativi di cessazione dei servizi che esso dispone.

In questa relazione vogliamo mostrare come si possano adottare tecniche di Data Mining e Machine Learning per risolvere i problemi legati alla sicurezza e alla protezione di risorse e dati.

Gli strumenti su cui intendiamo studiare queste tecniche sono conosciuti con il nome di *Intrusion Detection System*.

Che cos'è un Intrusion Detection System?

Un intrusione può essere definita come una serie di azioni che attentano alla compromissione dell'integrità, confidenzialità o disponibilità di una risorsa [HLMS90]. Per evitare questo si sviluppano tecniche di prevenzione come l'autenticazione degli utenti, il controllo degli errori di programmazione, la protezione delle informazioni tramite crittografia. Queste tecniche da sole non sono sufficienti a garantire la protezione di un sistema informatico ma forniscono una prima linea di difesa. La semplice prevenzione è insufficiente quando il sistema diventa più complesso rendendo molto difficile controllare tutte le possibili debolezze in esso soprattutto quando queste sono legate a mancanze nella progettazione, ad errori di programmazione o a tecniche di "social engineering”.

Un *Intrusion Detection System* o *IDS* è uno strumento hardware/software utilizzato per il rilevamento e l'identificazione di intrusioni ed accessi non autorizzati a computer o reti.

L'architettura tipica di un IDS è composta da tre componenti: sensori, console e motore. I *sensori* si occupano di rilevare e generare gli eventi di sicurezza. Sulla *console* gestiremo i sensori e verranno visualizzati allarmi, avvertimenti, e mostrati gli eventi. Il *motore* si occuperà di registrare gli eventi in una base di dati e analizzarli per produrre eventualmente allarmi e file di registrazione.

Gli elementi centrali del rilevamento delle intrusioni sono:

- Le *risorse* da proteggere nel sistema in esame, come gli account utenti, i file system, i kernel del sistema operativo.
- I *modelli* che caratterizzano un comportamento "normale" o "legittimo" di tali risorse.
- Le *tecniche* che confrontano le attività attuali del sistema con i modelli stabiliti e determinano quello che può essere un comportamento "anomalo" o "intrusivo".

Nella creazione di tali sistemi si devono fare assunzioni a priori su cosa siano comportamenti "normali" ed "anomali", ipotizzando inoltre che dato un insieme sufficiente di caratteristiche del sistema modellate e misurate si possa determinare con buona accuratezza l'ambiente in cui ci troviamo.

Le tecniche di rilevamento intrusione possono essere divise in *misuse detection*, che usano pattern di attacchi ben conosciuti o di punti deboli del sistema per identificare le intrusioni, ed in *anomaly detection*, che cercano di determinare una possibile deviazione dai pattern stabiliti di utilizzazione normale del sistema.

I *misuse detection system* o *signature based intrusion detection system*, codificano e confrontano una serie di segni caratteristici (s*ignature action*) delle varie tipologie di scenari d'intrusione conosciute. Queste caratteristiche possono ad esempio essere i cambi di proprietà di un file, determinate stringhe di caratteri inviate ad un server e così via. I principali svantaggi di tali sistemi sono che i pattern di intrusione conosciuti richiedono normalmente di essere inseriti manualmente nel sistema, ma il loro svantaggio è soprattutto di non essere in grado di rilevare qualsiasi futura (quindi sconosciuta) tipologia di intrusione se essa non è presente nel sistema. Il grande beneficio che invece hanno è quello di generare un numero relativamente basso di falsi positivi, e di essere adeguatamente affidabili e veloci.

Gli *anomaly detection systems* fanno uso di profili (pattern) dell'utilizzo normale del sistema ricavati da misure statistiche ed euristiche sulle caratteristiche dello stesso, per esempio, la CPU utilizzata e le attività di I/O di un particolare utente o programma. Le maggiori problematiche legate a tali sistemi sono principalmente legate alla selezione delle caratteristiche del sistema da adottare, queste possono variare enormemente a seconda dei vari ambienti di calcolo; inoltre alcune intrusioni possono essere soltanto rilevate studiando le relazioni che intercorrono tra gli eventi

perché l'evento singolo potrebbe rientrare correttamente nei profili. Tali sistemi richiedono l'applicazione di algoritmi e tecniche prese dall'intelligenza artificiale e dal data mining, in quanto il sistema dev'essere essenzialmente autonomo ed apprendere da situazioni nuove, cercando di imparare dai propri errori.

Un'altra possibile divisione che possiamo adottare per classificare i sistemi di rilevamento è se vanno ad analizzare la rete, la macchina ospite, o entrambe:

- Un *Network Intrusion Detection System* processa i dati relativi al traffico di rete al fine di rilevare intrusioni. Si mette quindi in ascolto nel tentativo di rilevare traffico anomalo; per fare questo i suoi sensori vengono posizionati in punti critici dell'architettura di rete (ad esempio nei punti di accesso ad una rete locale o ad una DMZ).
- Un *Host based Intrusion Detection System* rileva i tentativi di intrusione effettuati sui servizi e le risorse offerte su un sistema locale, quindi ad esempio sul sistema operativo e sul software localizzati sulla macchina ospite. Ad esempio questi sistemi dovranno rilevare modifiche al file degli utenti e delle password, alla gestione dei privilegi, alle basi dati locali etc.
- Un *Hybrid Intrusion Detection System* fornisce le funzionalità dei due sistemi precedenti, andando ad integrare i dati relativi al traffico di rete e quelli forniti dal software in esecuzione sulla macchina locale.

L'utilizzo di tecniche di Data Mining e Machine Learning in questi sistemi serve ad eliminare, il più possibile, gli interventi manuali o specifici dai processi di analisi di un sistema di rilevamento intrusione.

L'accuratezza di un sistema simile è legata a quanto bene riusciamo a rilevare attacchi alla sicurezza del sistema, più precisamente questo si può misurare tramite due parametri:

- *Detection rate:* indicante il rapporto tra attacchi rivelati dall'IDS su attacchi subiti dal sistema su cui è in esecuzione.
- *False positive rate:* indicante il tasso di dati che il sistema erroneamente considera essere di natura intrusiva quando invece sono prodotti da eventi relativi al normale utilizzo.

Questi due parametri vengono normalmente misurati tramite sperimentazioni reali o come vedremo in seguito su dati diversi da quelli usati per addestrare il sistema (cioè diversi da quelli di *training set)*.

E' importante notare come l'accuratezza non possa essere da sola sufficiente, in quanto i sistemi di rilevamento intrusione devono lavorare in *real time*, cioè devono determinare sul momento la

presenza di eventuali intrusi o di violazioni del sistema. Questo complica le cose dal punto di vista dell'efficienza e dell'utilizzabilità specialmente quando andiamo a parlare di *anomaly detection systems* perché l'analisi sui dati e sui data set dev'essere fatta in maniera molto veloce. Questo non è richiesto quando ad esempio si vanno a fare studi di *analisi forense*, cioè quando ormai è noto che il sistema o la rete d'interesse è già stata violata e si vuole sapere a posteriori come questo è avvenuto.

Il *Data mining* riferisce generalmente al processo automatico di estrazione di modelli da larghi archivi di di dati. I recenti sviluppi in questo campo hanno reso disponibili una grande varietà di algoritmi, che traggono le loro basi dai campi della statistica, del riconoscimento di pattern, dell'apprendimento automatico e delle basi di dati. Le tipologie di algoritmi utili per questi sistemi sono principalmente tre:

- *Algoritmi di classificazione:* presi dei dati vanno a determinare a seconda delle loro caratteristiche se appartengono ad una classe o ad un'altra. Questi algoritmi forniscono come risultato dei classificatori. Presi degli *audit data* cioè dei dati d'ascolto ricavati dai sensori del sistema, si applicherà poi un algoritmo di classificazione per istruire un classificatore che servirà in futuro a distinguere tra la classe di comportamenti normali e la classe di comportamenti anormali.
- *Analisi di relazioni e collegamenti:* questi algoritmi permettono di determinare relazioni tra i vari campi di una base di dati, essi trovano correlazioni nei dati di audit e provvedono a selezionare l'insieme corretto di caratteristiche del sistema per il rilevamento di intrusioni.
- *Analisi di sequenze:* servono a fornire informazioni per la comprensione di una sequenza temporale di eventi rilevati dal sistema, permettendoci ad esempio di creare dei profili di comportamento per un utente o un programma.

Per completezza specifichiamo l'ultimo aspetto che può caratterizzare questi sistemi, e dove sono possibili molte applicazioni delle tecniche di apprendimento automatico, infatti si può ulteriormente differenziare tra:

- *Sistemi passivi* che si limitano a notificare sulla console dell'operatore la presenza di eventi sospetti o di reali intrusioni.
- *Sistemi attivi* che subito dopo aver rilevato un intrusione attivano meccanismi automatici che mettono in atto delle opportune contromisure per eliminare o comunque isolare la violazione del sistema o della rete. Per fare questo ad esempio si vanno a modificare automaticamente le liste di accesso di un sistema o le regole di un firewall.

Concludiamo questa introduzione specificando che i sistemi attuali in commercio sono principalmente basati su *signature* in quanto questi sistemi sono attualmente di più facile

realizzazione e di una maggiore velocità computazionale. Presenteremo alcuni approcci presi dalla letteratura relativa ai sistemi adattativi che mirano a modificare questa visione, cercando di mostrare le tecniche che rendono questi oggetti più indipendenti dall'intervento umano. L'operatore umano rimane comunque essenziale nel campo della sicurezza informatica, in quanto è proprio degli attacchi andare a sfruttare l'anatomia e le debolezze proprie dei sistemi, rendendo inutili anche le previsioni statistiche e i rilevatori più evoluti.

Data mining sui dati di audit

Andremo a presentare alcune proposte sono state discusse in letteratura sullo sviluppo di strutture per la costruzione di modelli di intrusion detection [LS98]. Fondamentalmente questi sistemi si basano sui dati di audit riguardanti pattern relativi ai comportamenti di utenti e programmi, presi questi dati si ha un insieme di caratteristiche rilevanti del sistema che verranno in seguito usate per ricavare(tramite apprendimento induttivo) classificatori che possano riconoscere tali anomalie ed intrusioni.

Un modo proposto per ricavare informazioni da questi dati di audit è quello di generare *regole associative* significative tra i vari elementi di una base di dati(tramite variazioni dell'algoritmo Apriori [AS94]). Queste regole forniscono "correlazioni" tra i vari eventi che accadono nel sistema, permettendoci di avere una visione di quello che succede all'interno di esso. Andiamo ora a spiegare i concetti base di questa tecnica in maniera più formale.

Sia A un insieme di attributi, ed I un insieme di valori per A chiamati *item*. Ogni sotto insieme di I è chiamato un *item set*. Il numero di item in un item set definisce la *lunghezza* dello stesso. Sia D una base di dati con n attributi(o colonne).

Una *regola di associazione* è l'espressione:

$$X \rightarrow Y, \textit{confidenza, supporto}$$

Siano qui X ed Y item set, e sia la loro intersezione vuota. Il *supporto* sia la percentuale di record in D che contengono l'unione dei due item $X \cup Y$, e sia la *confidenza* la percentuale di record che contengono X e che contengono anche Y.

Ad esempio, data la regola di associazione che lega la storia dei comandi (history) di una shell da riga di comando di un utente (cioè i programmi lanciati e i loro argomenti):

$$ls \rightarrow /home/john, 0.3, 0.1$$

si saprà che il *30%* delle volte che l'utente chiama il comando *ls* lo farà per visualizzare i file contenuti nella sua home directory per un 10% delle attività registrate .

Tali algoritmi esaminano la correlazione sui valori di attributi differenti e i dati (pre-processati) di audit hanno di solito attributi multipli ed un largo numero di valori possibili. Per questo motivo si potrà scegliere di non memorizzare i dati in una base di dati binaria. Tale decisione ovviamente porterà ad un aumento di velocità a danno della memoria occupata. La struttura dati per un item set frequente è data da un *vettore riga* (di bit) che registra i record in cui l'item set è contenuto. La base di dati verrà scorsa solo una volta per generare la lista degli item set frequenti di lunghezza *1*. Quando un item set c_k di lunghezza *k* viene generato dalla congiunzione di due *item set frequenti* di lunghezza *k-1* chiamati l_{k-1}^1 e l_{k-1}^2, il vettore riga di c_k è semplicemente l'AND bit a bit dei due vettori riga l_{k-1}^1 e l_{k-1}^2. Il *supporto* per c_k può essere facilmente calcolato contando gli uni nel suo vettore riga.

Data una base di dati *D* dove a ciascuna transazione (record) è associata un *timestamp.* Sia l'intervallo $[t_1, t_2]$ la sequenza delle transazioni che iniziano con il timestamp t_1 e finiscono con il timestamp t_2. La durata dell'intervallo è definita da $t_2 - t_1$. Dato un item set *A* in *D*, un intervallo è una minima occorrenza di *A* se esso contiene *A* e nessuno dei suoi sotto-intervalli contiene *A*.

Si dice *frequent episode rule* l'espressione:

$$X,Y \rightarrow Z,\ confidenza,\ supporto,\ finestra.$$

Qui *X*, *Y* e *Z* sono item set in D, il *supporto* è la percentuale di occorrenze minime dell'unione di *X, Y ,Z* in *D*(che è il rapporto tra il numero di occorrenze e il numeri di record in D), mentre la confidenza è la percentuale di occorrenze minime che contengono l'unione di *X* e *Y* e che contengono anche *Z*. La durata di ogni occorrenza dev'essere minore della *finestra*.

Ad esempio con:

$$(servizio = http, flag = SO), (servizio = http, flag = SO) \rightarrow (servizio = http, flag = SO)$$
$$[0.93,0.03,2]$$

intendiamo che il *93%* delle volte, quando il sistema riceve due connessioni *http* con la flag *SO,* dopo *2* secondi da esse si vede arrivare una terza connessione, e questo pattern si verifica nel *3%* dei dati. Un episodio detto *seriale* ha come vincolo aggiuntivo che *X, Y* e *Z* devono occorrere in transizioni in ordine temporale, questo significa che vorremo che *Z* segua *Y* ed *Y* segua *X*.

Gli algoritmi base che lavorano sulle regole di associazioni non considerano nessun dominio di conoscenza, e questo porta alla generazione di regole di associazioni irrilevanti, che dovranno essere poi esaminate in seguito per eliminare quelle inutili. Si farà quindi uso di uno o più di *axis attribute*, cioè quegli attributi ritenuti rilevanti nella transizione scelti tra quelli “essenziali”(come

nel caso di una connessione possono essere l'indirizzo sorgente, destinatario, il servizio etc). Si andranno quindi a scartare le correlazioni tra gli altri attributi ritenuti non utili.

tempo	*durata*	*servizio*	*src_bytes*	*dst_bytes*	*flag*
1.1	10	telnet	100	2000	SF
2.0	2	ftp	200	300	SF
2.3	1	smtp	250	300	SF
3.4	60	telnet	200	12100	SF
3.7	1	smtp	200	300	SF
3.8	1	smtp	200	300	SF
5.2	1	http	200	0	REJ
3.7	2	smtp	300	200	SF

Tabella 1: Network Connection Records

Ad esempio guardiamo la Tabella 1. L'algoritmo base per la ricerca di regole di associazione, senza l'uso di un attributo axis, può generare regole quali:

$$src\_bytes = 200 \rightarrow flag = SF$$

Regole come questa sono inutili e a volte potrebbero sviare e causare falsi positivi, in quanto non c'è alcuna connessione tra il numero di byte provenienti dalla sorgente e lo stato normale della connessione. In una connessione per esempio conta molto il tipo di *servizio* che eroga, possiamo decidere di affidare a lui il ruolo di axis. Questo significherà che le regole seguenti descriveranno solo pattern che comprendono quest'attributo.

Diventa ancora più importante l'utilizzo di questa tecnica per vincolare la generazione di episodi frequenti. Ad esempio l'algoritmo classico potrebbe generare *frequent episode rule* come la seguente:

$$src\_bytes = 200,\ src\_bytes = 200 \rightarrow dst\_bytes = 300,\ src\_bytes = 200$$

dove ogni valore degli attributi, come *src_bytes=200*, proviene da record di connessione separati e quindi senza alcuna validità ne significatività.

L'algoritmo modificato con l'attributo axis troverà quindi regole di associazione “utili” come la seguente:

$$(servizio = smtp,\ src\_bytes = 200,\ dst\_bytes = 300,\ flag = SF)\ ,$$
$$(servizio = telnet,\ flag = SF) \rightarrow\ (servizio = http,\ src\_bytes = 200)$$

In tal maniera si sono combinate le associazioni (sugli attributi) e i pattern sequenziali(sui record) in una regola unica, tale formalismo non solo elimina le regole irrilevanti, ma provvede anche a fornire informazioni proficue sui dati di audit.

A volte diventa importante scoprire i pattern con frequenza bassa, perché alcuni servizi possono generare un traffico molto basso rispetto a quello totale(ad esempio *snmp*); in questi casi si richiede comunque che vengano inseriti nei nostri profili, anche se statisticamente risulterebbero meno importanti rispetto ad altri servizi quali ad esempio l'*smtp*. Si effettuerà quindi una ricerca tramite un algoritmo detto di *level-wise*:

***Input**: il supporto finale minimo s_0 il supporto iniziale minimo s_i e l'attributo asse*
***Output:** le regole di episodio frequente Rules*

```
Begin
(1) R_restricted = 0;
(2) scandire il database per formare L = {1 - item set che incontrano s_0};
(3) s = s_i;
(4) while (s ≥ s_0) do begin
(5)  trovare gli episodi seriali da L: ciascun episodio deve contenere
almeno un valore dell'attributo axis value che non è in R_restricted;
(6)  aggiungere i nuovi valori di attributo axis a R_restricted;
(7)  aggiungere le episode rules all'insieme delle regole risultanti
Rules;
(8)  s = s/2;
    end while
end
```

L'idea è quella di trovare prima gli episodi relativi ai valori di attributi axis con frequenza alta, per esempio:

(servizio = smtp, src_bytes = 200), (servizio = smtp, src_bytes = 200)

→ (servizio = smtp, dst_bytes = 300)

Poi iterativamente abbassiamo la soglia del supporto per trovare gli episodi relativi ai valori di attributi axis con frequenza bassa restringendo la partecipazione dei valori di axis "vecchi" che sono già usciti. Più specificatamente, quando un episodio è generato, deve contenere almeno un valore axis "nuovo" cioè con bassa frequenza. Per esempio, nella seconda iterazione (dove *smtp* ora è un valore axis vecchio), noi otteniamo la regola:

(servizio = smtp, src_bytes = 200), (servizio = http, src_bytes = 200)
→ (servizio = smtp, src_bytes = 300).

La procedura termina quando si è raggiunto un valore molto basso di supporto, questo si può decidere che sia la più bassa frequenza di tutti i valori axis.

Si noti che per un valore axis con alta frequenza, si vanno ad omettere i suoi episodi con frequenza molto bassa(generati nei cicli con bassi valori di supporto) perché essi non sono ritenuti d'interesse o rappresentativi. Tale procedura risulta quindi un mining "approssimativo". Si continuerà però ad includere tutti i vecchi valori axis con alta frequenza per formare gli episodi con nuovi valori di axis

perché comunque risulta importante catturare la sequenzialità dei contenuti di questi. Ad esempio, anche se usato in maniera poca frequente, *auth* occorre normalmente con alti servizi quali *smtp* e *login*. E' quindi d'obbligo includere questi servizi a frequenza alta nelle regole riguardanti *auth*.

Riunione dei dati di Audit

Dopo aver raccolto e scoperto i pattern presenti nei dati di audit di un obbiettivo da proteggere(sia questo una macchina, un rete, un utente etc) abbiamo le informazioni necessarie per capirne il comportamento.

Quando riuniamo i dati di audit raccolti sul nostro obbiettivo, andiamo a calcolare i pattern da ciascun nuovo data set, e andiamo a fondere le nuove regole con l'attuale sistema di regole aggregate. L'aggiunta di nuove regole rappresenta nuove variazioni sul comportamento normale del sistema in esame. Se l'insieme di regole aggregate si stabilizza, ad esempio nel caso in cui non sia possibile aggiungere nuove regole dai nuovi dati di audit, si può fermare la raccolta dei dati in quanto l'insieme ha coperto un numero sufficiente di variazioni.

Un possibile approccio per la fusione delle regole si deve basare sul fatto che anche lo stesso tipo di comportamento potrà avere leggere differenze attraverso i vari data set. Non ci si deve quindi aspettare un confronto esatto dei pattern recuperati. Invece si richiede di combinare i pattern simili in altri più generali.

Ad esempio si possono fondere due regole r_1 e r_2 in una regola r se:

1. Le loro parti destra e sinistra sono esattamente le stesse, oppure le loro parti destra possono essere *combinate* ed anche le loro parti sinistre.
2. I valori corrispettivi di *supporto* e di *confidenza* sono vicini, ad esempio con una soglia definita dall'utente.

Ipotizziamo di combinare le parti sinistre e che la parte sinistra di r_1 ha solo un item set:

$$(a\,x_1 = vx_1,\ a_1 = v_1)$$

dove ax_1 è un attributo axis. La parte sinistra di r_2 deve avere anch'essa un solo item set:

$$(a\,x_2 = vx_2,\ a_2 = v_2)$$

inoltre si richiede che valgano i seguenti vincoli:

$$ax_1 = ax_2,\ vx_1 = vx$$

$$a_1 = a_2$$

dato che le parti sinistre devono coprire lo stesso insieme di attributi allora i loro valori axis devono essere gli stessi. Affinché le parti sinistra possano essere combinate, v_1 e v_2 devono essere valori adiacenti. La parte sinistra della regola risultante *r* assumendo che v_2 sia il valore più largo risulta:

$$ax_1 = vx_1 \text{ con } v_1 \leq a_1 \leq v_2$$

Ad esempio dati:

(servizio = smtp, src_bytes = 200) e *(servizio = smtp, src_bytes = 300)*

queste regole possono essere combinate nella seguente:

(servizio = smtp, 200 ≤ src_bytes ≤ 300)

E' possibile introdurre un altro livello di analisi e al fine di poter permettere che alcune feature essenziali possano referenziarne altre. Queste *reference feature* normalmente si usano per fornire informazioni relativamente ad un certo “soggetto”, mentre altre caratteristiche descriveranno le “azioni” che si riferiscono a quel soggetto.

Ad esempio se noi vogliamo studiare i pattern sequenziali relativi alle connessioni verso lo stesso host di destinazione, stabiliremo che *dst_host* è il “soggetto” e che *servizio* sia la ”azione”. In questo caso scegliamo *dst_host* come reference feature. Nella creazione di un episodio l'algoritmo andrà a controllare, all'interno delle occorrenze minime dello stesso, che gli eventi registrati riguardino gli item set costituenti ed aventi lo stesso valore di referenza.

Costruzione delle feature

La raccolta dei dati relativi agli episodi frequenti, la creazione di regole di associazioni risultano strumenti e tecniche inutili se non vengono usate come linee guida per la costruzione di caratteristiche statistiche temporali per la costruzione di modelli di classificazione. Questo processo coinvolge prima di tutto il riconoscimento dei pattern caratteristici di comportamenti unici di un intrusione. Mostreremo qui un approccio [LS00] per la costruzione delle *caratteristiche* o *feature* da usare poi per la classificazione.

Per identificare i pattern relativi ad una intrusione, si andranno a ricavare due data set, uno in cui il sistema svolge il suo ruolo normale, ed uno in cui ha subito un intrusione. Si andranno poi a confrontare le frequent episode rule al fine di far emergere i pattern caratteristici dall'intrusione, questi vengono conosciuti con il nome di “intrusion-only”.

Un algoritmo di confronto procederà ad esaminare i vari pattern. Bisogna tenere conto che il numero dei pattern può essere molto largo e che raramente ci sono pattern riconosciuti esattamente

dai due data set, si useranno quindi algoritmi euristici per identificare automaticamente solo i pattern d'intrusione. L'idea consiste nel convertire i pattern in numeri in maniera tale che a pattern "simili" corrispondano numeri "vicini". In seguito il confronto tra pattern e l'identificazione sono realizzati attraverso il confronto dei numeri ed un ordinamento gerarchico dei risultati.

Inoltre come abbiamo detto l'aggregazione dei dati relativi al comportamento normale del sistema richiede una grande mole di spazio di memorizzazione. L'uso di una codifica per convertire ciascun pattern frequente in un un numero riduce questi spazi.

Codifica delle feature

Come detto si vuole che la codifica faccia corrispondere alle associazioni più simili per struttura e sintassi numeri "vicini" tra loro. Si definisca quindi una misura di similarità e per prima cosa si stabilisca un ordine parziale delle associazioni scoperte. Assumendo che i record abbiano n attributi, si vada a chiamare un associazione $(A_1 = v_1 , A_2 = v_2 , ..., A_k = v_k)$ "completa ed ordinata" se $k = n$ e gli attributi $A_1 , A_2,..., A_k$ sono in un qualche *ordine di importanza* definito dall'utente (ad esempio nel caso più semplice in ordine alfabetico).

Un'associazione scoperta può essere sempre convertita nella corrispondente forma "completa ed ordinata" inserendo inizialmente $A_i = null$ per ogni valore mancante dall'attributo A_i, e poi ordinando gli attributi per importanza.

Per due associazioni "complete ed ordinate" si dice che

$$(A_1 = v_1, A_2 = v_2, ..., A_n = v_n) < (A_1 = u_1 ,A_2 = u_2 , ...,A_n = u_n)$$

$$\text{se } v_j = u_j \text{ per } j = 1 , 2 , ..., i-1 \text{ e } v_i < u_i.$$

Allora diciamo che l'associazione X_i è più "simile" a X_j che a X_k se vale $X_i < X_j < X_k$ (oppure che $X_k < X_j < X_i$).

Dato un insieme di associazioni, si usa il seguente algoritmo per calcolare le codifiche:

- Convertire ciascuna associazione nella corrispettiva forma "completa ed ordinata".
- La codifica di un'associazione $(A_1 = v_1, A_2 = v_2, ..., A_n = v_n)$ è un numero $e_{v1}e_{v2} \ldots e_{vn,}$ dove l'ordine delle cifre va' dalla più significante a quella meno significante, in "ordine d'importanza" decrescente degli attributi.
- Ciascun e_{vi} è:

  - *0* se v_i è *null*, ad esempio quando l'attributo *Ai* è mancante dall'associazione originale;
  - l'*ordine di comparsa* di *vi* fra i valori di *Ai* esaminati(processati) finora nel processo di codifica(altre forme di ordinamento possono essere incorporate facilmente).

Quando si codificano le associazioni dai record di rete,si usa il seguente “ordine d'importanza” decrescente: *flag*, *attributo axis, attributo di referenza*, *il resto degli attributi essenziali in ordine alfabetico*, *tutti i rimanenti attributi in ordine alfabetico*. L'attributo *flag* è ritenuto il più importante ed interessante dato che il suo valore è un sommario di come la connessione si sta comportando in accordo con i protocolli di rete. Ogni altro valore diverso da *SF*(corrispondente alla instaurazione normale della connessione e alla terminazione) è di grande interesse per il rilevamento d'intrusione.

Timestamp	Durata	Servizio	src_host	dst_host	src_bytes	dst_bytes	Flag
1.1	0	http	spoofed_1	vittima	0	0	S0
1.1	0	http	spoofed_2	vittima	0	0	S0
1.1	0	http	spoofed_3	vittima	0	0	S0
1.1	0	http	spoofed_4	vittima	0	0	S0
1.1	0	http	spoofed_5	vittima	0	0	S0
1.1	0	http	spoofed_6	vittima	0	0	S0
1.1	0	http	spoofed_7	vittima	0	0	S0
.....	...	...	...	...	...	...	...
10.1	2	ftp	A	B	200	300	SF
12.3	1	smtp	B	D	250	300	SF
13.4	60	telnet	A	D	200	12100	SF
13.7	1	smtp	B	C	200	300	SF
15.2	1	http	D	A	200	0	REJ
...	...	...	...	...	...	...	...

Tabella 2: SYN Flood Attack

Consideriamo come esempio l'attacco SYN flood mostrato nella Tabella 2. L'attaccante ha utilizzato diversi indirizzi sorgente *spoofed* per mandare molte connessioni(dove tra l'altro viene inviato solo il pacchetto di SYN) ad una porta(nel nostro caso http) della macchina vittima in un lasso di tempo decisamente basso(con marca temporale 1.1).

Nella seguente Tabella 3 possiamo vedere una regola associativa ottenuta per fornire un pattern relativo ad un comportamento tipico di un intrusione:

Episodio frequente	Significato
(flag = S0, servizio = http, dst_host = vittima), (flag = S0, servizio = http, dst_host = vittima) → (flag = S0, servizio = http, dst_host = vittima) [0.93, 0.03, 2]	Il 93% del tempo, dopo due connessioni *http* con flag S0 ricevute dall'host *vittima*, entro due secondi dalla prima di queste due, una terza connessione viene fatta. E questo pattern occorre nel 3% dei dati.

Tabella 3: Un Intrusion-only pattern

La Tabella 4, invece, mostra alcuni esempi di codifica delle regole di associazioni computate:

associazione	*codifica*
(flag = SF, servizio = http, src_bytes) = 200	*11001*
(servizio = icmp_echo, dst_host = B)	*02100*
(flag = S0, servizio = http, src_host = A)	*21010*
(servizio = user_app, src_host = A	*03010*
(flag = SF, servizio = icmp_echo, dst_host = B, src_host = C)	*12120*
...	*...*

Tabella 4: Esempio delle codifiche usate per le associazioni

Qui l'attributo axis è il *servizio*, mentre *dst_host* è l'attributo di referenza. Le associazioni sono codificate nell'ordine della loro posizione nella tavola (prima la prima riga), si noti che le cifre sono solo cinque, in quanto il sesto attributo *dst_bytes* non compare in nessuna associazione. Un vantaggio di questo schema di codifica è che usa operazioni aritmetiche molto semplici per controllare il livello di dettaglio richiesto per l'analisi ed il confronto delle associazioni.

Possiamo ora far corrispondere ad un regola episodica $X,Y \rightarrow Z$, dove X, Y e Z sono associazioni(item set), una tale rappresentazione dei dati: $codifica_X$, $codifica_Y$, $codifica_Z$.

Per ridurre le difficoltà di manipolazione nel trattare rappresentazioni di dimensioni maggiori a *3,* per regole più "lunghe" $L_1, L_2, ..., L_i \rightarrow R_1, R_2, ..., R_j$ $(i, j \geq 1)$,si usa una forma riassunta $L_1, L_2 \rightarrow R_j$. Se L_2 è mancante, semplicemente specifichiamo $codifica_{L2} = 0$.

Per il confronto tra pattern, prima andiamo a convertire la codifica tridimensionale di un episodio in un valore ad una dimensione. Assumendo $codifica_X = X_1X_2...X_n$, $codifica_Y = Y_1Y_2...Y_n$ e $codifica_Z = Z_1Z_2...Z_n$, allora la sua rappresentazione unitaria sarà $X_1Z_1Y_1X_2Z_2Y_2...X_nZ_nY_n$. Questa rappresentazione preserva l'ordine d'importanza degli attributi nell'associazione codificata mantenendo la struttura tipica dell'episodio. Qui due episodi che hanno simili il "corpo"(es: X) e la "testa"(es: Z) corrisponderanno a numeri vicini tra loro. Come esempio, guardando le associazioni codificate nella Tabella 4(ed "ignorando" *dst_host*, *src_host* e *src_bytes),* il pattern del SYN Flood di Tabella 3 è codificato come 222111. In maniera simile un pattern "normale" come

(flag = SF, service = http), (flag = SF, service = icmp echo) → (flag = SF, service = http)

è codificato come 111112.

Quando si confrontano due episodi utilizzando i loro numeri ad una dimensione, un semplice confronto *digit-wise* (senza riporto) viene effettuato, ottenendo il punteggio di differenza *diff:*

$$d_{x1}d_{z1}d_{y1}d_{x2}d_{z2}d_{y2} \ldots d_{xn}d_{zn}d_{yn}$$

dove ciascuna cifra d_{xi} è il valore della differenza assoluta delle cifre corrispondenti X_i dei due episodi. Per esempio quando andiamo a confrontare il pattern "syn flood" con il pattern "normale",

il punteggio di differenza è *111001*.

Dati i pattern del comportamento normale e i pattern relativi ad un intrusione i data set sono calcolati utilizzando la stessa scelta di attributi axis, di attributi di referenza, supporto, confidenza e requisiti di finestra, noi possiamo identificare i pattern "intrusion only" utilizzando la seguente procedura:

- Codificare tutti i pattern;
- Per ciascun pattern dal data set dell'intrusione, calcolare il suo punteggio di differenza con ciascun pattern normali, tenendo il suo più basso *diff score* come il punteggio d'intrusione per questo pattern;
- Restituire tutti pattern che hanno un punteggio d'intrusione differente da *0*. Per esempio dato che non c'è un pattern "normale" con *flag=S0* in tutti i suoi item set, il punteggio di differenza per il "syn-flood" pattern, ad esempio 111001, sarà molto alto e quindi il pattern verrà selezionato.

Questa procedura considera un pattern da un data set d'intrusione come "normale" finché esso ha una corrispondenza con uno dei pattern "normali".

Algoritmo di costruzione delle feature

Ogni pattern d'intrusione è utilizzato per costruire feature aggiuntive all'interno dei record di connessione, utilizzando il seguente algoritmo:

Input: *un episodio frequente e l'insieme delle feature esistenti nei record di connessione F*
Output: *l'insieme F aggiornato*

```
Begin
(1)   Sia F0 (ad esempio dst_host) l'attributo di referenza usato per
      l'estrazione dell'episodio;
(2)   Sia w, in secondi, la larghezza minima dell'episodio;

      /* tutte le feature seguenti considerano solo le connessioni
       * nei passati w secondi che condividono lo stesso valore in F0
       * come connessione corrente.
       */

(3)   Sia count_same_F0 il numero di queste connessioni;
(4)   F = F U {count_same_F0};
(5)   for ciascun "attributo essenziale" F1 diverso da F0 do begin
(6)        if lo stesso valore F1 è in tutti gli item set then begin
(7)         Sia percent_same_F1 la percentuale delle connessioni che
            condividono lo stesso valore F1 come connessione corrente;
(8)         F = F U {percent_same_F1 };
           end else
              /* ci sono valori diversi di F1 o non ci sono affatto */
(9)           Sia percent_diff_F1 la percentuale di valori F1 differenti
                  nella connessione
(10)          F = F U {percent_diff_F1 };
```

```
            end
        end
(11)    for ciascun valore V2 di un attributo "non essenziale" F2 do begin
(12)        if V2 è presete in tutti gli item set then begin
(13)         Sia percent_same_V2 la percentuale di connessione che
             condividono lo stesso valore V2 come connessione corrente;
(14)         F = F U {percent_same_V2};
(15)        end else if F2 è un attributo numerico then begin
(16)                    Sia average_F2 la media dei valori F2 delle
                        connessioni;
(17)                    F = F U {average_F2};
            end
        end
end
```

Questa procedura analizza un episodio frequente ed utilizza tre operatori *count, percent* ed *average* per costruire feature statistiche. Queste feature sono anche temporali dato che esse misurano solo le connessioni che rientrano nella finestra temporale *w* e condividono gli stessi valori degli attributi di referenza.

L'idea dietro l'algoritmo di costruzione viene dall'interpretazione diretta di un episodio frequente. Per esempio, se lo stesso valore di un attributo appare in tutti gli item set di un episodio, allora c'è una larga percentuale di record(nei dati originali) che hanno lo stesso valore. Si va' a trattare questi attributi "essenziali" e "non essenziali" in maniera differente. Gli attributi "essenziali" descrivono l'anatomia di un'intrusione, per esempio, "lo stesso servizio, la stessa porta" usata per un attacco. I valori attuali, come ad esempio l'*http* sono spesso non importanti perché lo stesso metodo di attacco può essere applicato a target differenti, come ad esempio l'*ftp*. D'altra parte, i valori degli attributi correnti "non essenziali", come *S0*, spesso indicano una caratteristica immutabile dell'intrusione perché riassumono il comportamento della connessione di rete.

Basandosi sulle precedenti osservazioni, si possono processare in seguito i pattern per eliminare gli *hostname* specifici e i nomi dei servizi, prima dei passi di codifica e confronto. Brevemente, per ciascun pattern si useranno src_0, src_1, ..., *etc* e dst_0, dst_1, ...,*etc* e srv_0, srv_1, ...,*etc* al fine di rimpiazzare gli host sorgenti, destinatari e i servizi nel pattern corrente.

Ad esempio la costruzione per il pattern "syn flood" risulterà nelle seguenti feature aggiuntive: un conteggio delle connessioni verso lo stesso *dst_host* negli ultimi *2* secondi, e fra queste connessioni, la percentuale di quelle che hanno lo stesso servizio di quella corrente, e la percentuale di quelle che hanno la *flag S0*.

Un problema aperto è decidere che valore assegnare alla finestra *w*. Intuitivamente un buon requisito per una corretta dimensione della finestra è che l'insieme dei pattern sequenziali sia stabile, cioè che vengano catturati pattern sufficienti e che il rumore sia piccolo.

Il processo della costruzione delle feature ci fornisce degli strumenti per discriminare tra i

comportamenti normali e quelli intrusivi, diventa poi facile generare le regole di rilevamento tramite l'intervento umano, quindi inserendole a mano nel sistema, oppure tramite sistemi che sfruttano tecniche di apprendimento automatico.

Le tecniche di data mining per il rilevamento di intrusione sono state sviluppate principalmente da Salvatore Stolfo e Wenke Lee della Columbia University, nelle loro sperimentazioni pratiche per la costruzione delle regole di rilevamento utilizzarono il sistema RIPPER, un apprenditore induttivo di regole. E' possibile utilizzare altri sistemi per la creazione delle regole, in questa breve trattazione descriverò questo sistema ed il suo funzionamento.

Ripper e creazione delle regole di rilevamento

Il Rule Induction è una delle più comuni forme di Knowledge discovery. Questa tecnica mira a scoprire dai dati una serie di regole "*if / then*" per la classificazione di casi differenti. Ogni regola è accompagnata da una misura di accuratezza che rende disponibile la selezione delle sole regole interessanti ai fini delle analisi. Ripper si presenta come un algoritmo di apprendimento di regole proposizionali operante in maniera efficiente su grossi data set dove vi è una forte presenza di rumore.

Ripper richiede come input un esempio rappresentato da un vettore di valori reali o di feature simboliche, seguite da una classe, permettendo anche feature basate su insiemi di valori. Questo algoritmo impara tramite “induzione” le regole di classificazione sulla base di un insieme di esempi pre-classificati.

Prima della fase di apprendimento, Ripper ordina tramite euristiche le classi; l'approccio usuale è quello di ordinarle per frequenza crescente. Ripper segue una strategia tipica degli apprenditori basati su alberi di decisioni, e sfrutta una strategia *overfit-and-simplify*. In una strategia di questo tipo un ipotesi è formata facendo prima “crescere” un albero complesso che però tende a soffrire di overfit sui dati, poi in seguito si procede a semplificarlo o “potarlo”. RIPPER si basa sull'algoritmo IREP (Incremental Reduced Error Pruning), con il quale condivide il processo di apprendimento delle regole, che ora andiamo a descrivere.

Inizialmente i dati di training disponibili vengono divisi in due insiemi:

- un *growing set* (corrispondente a 2/3 dei dati per Ripper)
- un *pruning set* (il resto dei dati).

L'apprendimento delle regole inizia con una clausola vuota. Il primo passo nella crescita di una regola è valutare tutte le condizioni della forma $A_n == v$, $A_c \leq \theta$ o $A_c \geq \theta$ dove A_n rappresenta uno degli attributi nominali dati a Ripper, e v è uno dei valori validi per l'attributo da prendere, mentre

A_c è una variabile continua e θ è un qualche valore per A_c che occorre nei dati di training. Il sistema procede facendo "crescere" le regole, cioè producendole, con un approccio *greedy* aggiungendo una condizione alla volta.

L'algoritmo iterativamente forma le regole sotto forma di clausole congiuntive che riguardano parte delle istanze positive, escludendo tutte quelle negative. Alla prossima iterazione, tutte le istanze positive prese sono rimosse dal training set, ed una nuova clausola congiuntiva è formata anche questa riguardante solo gli esempi positivi ed escludendo quelli negativi.

In tal maniera durante la fase di crescita, le varie condizioni sono ripetutamente aggiunte alla regola. L'apprendimento delle regole si ferma quando le regole non riguardano più nessuno degli esmpi negativi provenienti dal growing set. Questo conduce tuttavia ad un insieme di regole che soffrono di *overfitting* sui dati. Si richiede quindi una fase di *pruning* dove le condizioni delle regole sono ripetutamente cancellate fino all'abbassamento del tasso di errore. Ciascuna regola è immediatamente potata cancellando ogni sequenza finale delle condizioni nelle regole che massimizzano la funzione seguente:

$$v*(Rule, PrunePos, PruneNeg) = \frac{p-n}{p+n}$$

dove *PrunePos* è l'insieme degli esempi positivi nel pruning set e *PruneNeg* è l'insieme di esempi negativi nel pruning set. Il numero degli esempi riguardanti la regola *Rule* in *PrunePos* è *p* mentre *n* è il numero di esempi riguardanti la regola in *PruneNeg.* Una volta che la regola è creata, gli esempi che la riguardano sono rimossi dai dati di training. Ripper continua ad imparare regole fin quando la condizione di arresto non è raggiunta.

Come condizione di arresto si usa la *Minimum Description Length* o *MDL-based heuristic.* La description length è ottenuta sommando il numero dei bit richiesti per descrivere l'ipotesi di classificazione al numero di bit richiesti per descrivere le eccezioni a questa ipotesi. Il principio della minimum description length mira a minimizzare questa misura, questo spinge l'apprenditore verso regole più compatte.

La total description length di un insieme di regole è calcolata dopo l'aggiunta di ciascuna di esse. L'algoritmo si fermerà nell'aggiungere regole quando questa description length è più larga di *d* bit rispetto alla la description length più piccola trovata finora. Questo parametro *d* è normalmente settato a 64.

Ripper include inoltre ottimizzazioni sull'insieme di regole. Si considerino due alternative, il "rimpiazzamento" e la "revisione". Nel caso del rimpiazzamento, una nuova regola è appresa per ciascuna regola nell'insieme di regole, ma questa volta il pruning è effettuato per minimizzare il

tasso di errore dell'intero insieme di regole dei dati di pruning. La revisione consiste invece nel riesaminare le regole esistenti facendole crescere e poi tornando indietro facendo il pruning. L'MDL viene adottato anche per decidere se usare la regola originale, la regola di rimpiazzamento, o la regola rivista.

Il sistema restituisce come risultato dell'apprendimento un insieme di regole *if-then* riguardanti i casi particolari, ed una regola di default "vera" per le classi rimanenti. Ciascuna di queste regole prodotte da Ripper hanno una determinata informazione relativa alla "confidenza": il numero di esempi abbinati(cioè quelle istanze che si conformano alla regola) e il numero di esempi non abbinati(le istanze che sono in conflitto con la regola) all'interno dei dati di training.

Ripper presenta diverse opzioni che possono andare ad influenzare pesantemente sull'apprendimento:

- *Ordinamento delle classi*: prima della fase di apprendimento, Ripper decide secondo una qualche statistica di ordinare le classi. Ci sono tre metodi di ordinamento: *+freq, -freq* e *mdl.* L'opzione di default è *+freq*. Con *+freq* le classi vengono ordinate per frequenza crescente, con *-freq* invece per frequenza decrescente. Nel caso si scelga *MDL* le classi vengono ordinate a seconda della *description length* dell'insieme di regole.
- *Test negativi*: questa opzione è usata nella costruzione delle regole. L'opzione è specificata a *-!n* se l'utente desidera permettere test negativi nelle condizioni delle regole.
- *Semplificazione delle ipotesi:* si permette all'utente di semplificare le ipotesi di Ripper, che sono espresse da un insieme di regole *if-then.*
- *Trattamento degli esempi:* tramite questa opzione l'utente può determinare il numero minimo di esempi che devono essere trattati da una regola.
- *Rapporto delle perdite*: si vuole specificare il rapporto tra il costo di un falso negativo e il costo di un falso positivo. Si possono specificare valori di *0.5*, *1* e *2*.
- *Passi di ottimizzazione*: con questa opzione specificata l'utente controlla il numero di passi di ottimizzazione nell'apprendimento delle regole, settando opportunamente le iterazioni.

Nel nostro caso specifico Ripper verrà utilizzato per formare regole di rilevamento a partire dalle feature estratte, un esempio di tali regole potrà essere simile alla seguente:

```
if per 2 secondi trascorsi il numero delle connessioni allo stesso
dst_host sono maggiori di 4;
      and   la percentuale delle connessioni sullo stesso servizio è
maggiore del 75%;
```

and la *percentuale di connessioni che hanno la flag S0* è maggiore del 75%;

then siamo in presenza di un attaco *syn_flood*.

L'utilizzo di RIPPER per la creazione delle regole di rilevamento a partire dalle feature estratte con i metodi precedenti si è rilevata dagli esperimenti condotti in letteratura una decisione efficiente, purtroppo non sono riuscito a trovare esempi in cui si fossero provati altri algoritmi per la creazione delle regole di rilevamento che si siano mostrati migliori.

Rilevamento di anomalie basate sul payload

Un'altra idea interessante per la costruzione di un rilevatore d'intrusione anomaly based è quella di andare a studiare statisticamente il comportamento normale del payload del traffico di rete di un host costruendo un modello statistico, dove le variazioni rispetto a questo possono segnalare l'eventuale presenza di un'intrusione. Questo modello statistico è stato presentato e discusso in letteratura sotto il nome di PAYL nell'articolo [WS04] di K. Wang e S.J. Stolfo.

Il payload di una rete è semplicemente un flusso di byte. A differenza delle intestazioni dei pacchetti di una rete i payload non hanno un formato fisso ed ogni carattere o byte può apparire in qualsiasi posizione del flusso dei pacchetti.

Di solito i servizi di rete hanno delle porte con valori fissi assegnati, ciascuna applicazione ha il suo protocollo predefinito, e la sua tipologia di payload. Ad esempio il servizio SSH ha come porta predefinita la 22 e ha un payload che è crittografato e che appare con una distribuzione uniforme di byte, mentre il servizio TELNET usa la porta 23 e ha un payload principalmente caratterizzato da caratteri stampabili inseriti dall'utente tramite tastiera. La lunghezza di un payload varia molto a seconda della porta e del protocollo, generalmente la maggior parte dei comuni pacchetti TCP/IP ha una lunghezza dei payload che varia da 0 a 1460 byte. Le varie lunghezze portano spesso informazioni sul tipo di dati trasmessi, ad esempio i payload più grossi sono quelli che trasportano informazioni binarie su formati media, file binari, video clip e così via.

L'idea è quella di calcolare un modello di payload per ciascuna differente gamma di lunghezza del payload, per ciascuna porta, per ciascun servizio e a seconda della direzione del flusso di dati entrante o uscente dall'host.

Analisi n-gram

Al fine di avere un modello semplice e veloce da calcolare si è proposto di modellare il payload utilizzando una analisi *n-gram* con il valore *n* pari ad uno. Una *n-gram* è una sequenza di *n* byte adiacenti all'interno di un payload. Una sliding window con larghezza *n* è fatta scorrere sopra

l'intero payload e andando a contare le occorrenze di ogni singolo *n-gram*.

Si userà come *feature vector* per i payload la frequenza relativa a ciascun *n-gram* che è calcolata dividendo il numero di occorrenze di ciascun *n-gram* per il numero totale di essi. Il caso più semplice di un *1-gram* calcola la frequenza media di ciascun carattere ASCII a 0-255. Possono risultare all'interno della stessa frequenza media molte varianti più o meno stabili sulle frequenze dei caratteri, che devono essere quindi caratterizzate in maniera molto differente nel modello. Per questo si va' a calcolare, in aggiunta alla *media*, la *varianza* e la *deviazione standard* di ciascuna frequenza come altre caratteristiche aggiuntive. Così per un payload di lunghezza fissa di una determinata porta, si vanno a studiare la frequenza relativa a ciascun carattere come una variabile e si calcolano la loro media e deviazione standard come modello del payload.

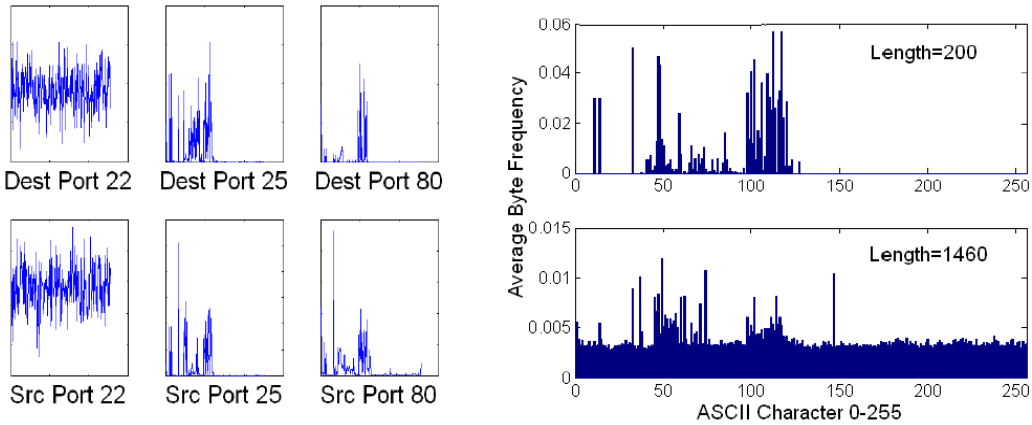

Figura 1: Esempi di di distribuzioni di caratteri per diverse porte e per la porta 80 con differenti lunghezze del payload [WS04].

Nella Figura 1 possiamo vedere come ad ogni porta corrisponda un grafico di distribuzione differente per ogni flusso di byte/caratteri e si può notare come per la distribuzione SSH(con flusso crittografato) le distribuzioni in entrata ed in uscita non presentino pattern particolarmente differenti rispetto alle altre.

Dato un insieme di dati di allenamento, si calcoli un insieme di modelli M_{ij}. Questo memorizza la frequenza media e e la deviazione standard relative alla frequenza di ciascun byte, questo viene fatto per ciascuna lunghezza osservata *i* e per ogni porta *j*. La costruzione di questo modello a partire dai dati di training rappresenta il comportamento normale del sistema. Durante il rilevamento ciascun payload in arrivo è preso in esame ed il suo valore di distribuzione è calcolato. Questa nuova distribuzione sul payload è poi confrontata con il modello M_{ij}. Se la distribuzione del nuovo pacchetto è significativamente differente dal modello normale, il rilevatore etichetta il pacchetto come anomalo e genera un avvertimento.

Distanza semplificata di Mahalanobis

La distanza di Mahalanobis è una distanza standard per confrontare due distribuzioni statistiche. Può essere un ottimo strumento per misurare la similarità tra il nuovo campione di payload e quello precedentemente calcolato. Questo viene fatto calcolando la distanza tra le distribuzioni dei byte del nuovo payload osservato rispetto al profilo del modello calcolato, relativamente alle gamme di lunghezza corrispondenti. Maggiore è il punteggio di distanza calcolato e maggiormente il payload del nuovo pacchetto è ritenuto anomalo.

La formula della distanza di Mahalanobis è la seguente:

$$d^2(x, \overline{y}) = (x - \overline{y})^T C^{-1} (x - \overline{y})$$

dove x e $\overline{y}$ sono due vettori di feature e ciascun elemento del vettore è una variabile. Il vettore di feature della nuova osservazione è x mentre $\overline{y}$ è il vettore feature medio calcolato dagli esempi di training. Sia C^{-1} l'inverso della matrice di covarianza data da $C_{ij} = Cov(y_i, y_j)$ dove y_i e y_j sono l'*i*-esimo elemento ed il *j*-esimo elemento del vettore di training.

Il vantaggio portato dall'utilizzo della distanza di Mahalanobis è che essa prende in esame non solo il valore medio ma anche la varianza e la covarianza delle variabili misurate. Invece di calcolare semplicemente la distanza dai valori medi, questa distanza pesa ciascuna variabile per la sua deviazione standard e per la sua covarianza, in tal maniera il valore calcolato fornisce una misura statistica di quanto bene i nuovi esempi sono consistenti con i campioni di allenamento.

Per rilassare il problema si può fare l'assunzione che ciascun byte sia indipendente dagli altri, quindi la matrice di covarianza *C* diventa diagonale e gli elementi lungo la diagonale sono solo la varianza di ciascun byte. Al fine di semplificare e velocizzare il processi di calcolo della distanza di Mahalanobis, si può usare una versione semplificata di quest'ultima detta *distanza di Mahalanobis semplificata:*

$$d(x, \overline{y}) = \sum_{i=0}^{n-1} (|x_i - \overline{y_i}| / \overline{\sigma_i})$$

dove la varianza è rimpiazzata con la deviazione standard ed *n* è fissato a *256* in quanto si usa il modello *1-gram*(dove quindi ci sono 256 valori possibili per ogni byte). L'utilizzo di questa funzione fornisce un rilevatore estremamente veloce, rendendo la computazione lineare nella lunghezza del payload. Utilizzando la distanza di Mahalanobis semplificata c'è la possibilità che la deviazione standard $\overline{\sigma_i}$ sia uguale a zero e che quindi la distanza diventi infinita. Questo succede quando un carattere o un byte non compare mai nei campioni di allenamento, oppure quando esso appare esattamente con la stessa frequenza in ciascun campione. Per evitare questa situazione si utilizza un fattore α rappresentante la confidenza statistica sul campione di training:

$$d(x, \overline{y}) = \sum_{i=0}^{n-1} (|x_i - \overline{y_i}| / (\overline{\sigma}_i + \alpha))$$

Più largo è il valore di α , meno le confidenze dei campioni sono realmente rappresentative dell'attuale distribuzione, e quindi la distribuzione dei byte può essere più variabile. Questo valore α può essere decrementato automaticamente con l'aumento del numero di campioni di training.

Apprendimento incrementale

Implementare un modello di *1-gram* incrementale è relativamente facile e richiede di aumentare di poco le informazioni memorizzate nel modello. Questo approccio presenta diversi vantaggi. Innanzitutto un modello simile non avrà bisogno di manutenzione specifica e migliorerà l'accuratezza man mano che il tempo di utilizzo aumenta e più dati vengono campionati. Inoltre una versione incrementale "online" può anche permette di far "scadere" i dati "vecchi" dal modello mantenendo una vista più accurata contenente solo i payload recenti relativi ad un servizio. Si può ad esempio stabilire un parametro di decadimento dei dati, ed enfatizzare le distribuzioni di frequenza che appaiono nei nuovi campioni. Questo modello ha un'accuratezza migliore rispetto ad uno "statico".

Per realizzare e calcolare una versione incrementale della distanza di Mahalanobis, si richiede di calcolare la media e la deviazione standard di ciascun carattere ASCII per ciascun nuovo campione osservato. Per la frequenza media di un carattere, viene calcolato:

$$\overline{x} = \sum_{i=1}^{N} x_i / N$$

dai campioni di training. Se noi andiamo anche a memorizzare il numero di campioni processati N, si può aggiornare la media quando si incontri un nuovo campione x_{N+1}:

$$\overline{x} = \frac{\overline{x} \times N + x_{N+1}}{N+1} = \overline{x} + \frac{x_{N+1} - \overline{x}}{N+1}$$

Siccome la deviazione standard è la radice quadrata della varianza, il calcolo della varianza può essere riscritto utilizzando il seguente valore atteso E come:

$$Var(X) = E(X - EX)^2 = E(X^2) - (EX)^2$$

è possibile quindi aggiornare la deviazione standard in maniera simile andando a memorizzare la media delle x^2_i nel modello. Questo richiede di mantenere in più un array di 256 elementi in ciascun modello che memorizzi la media delle x^2_i e il numero di osservazioni totali N.

Ridurre la dimensione del modello tramite raggruppamento

Nella descrizione appena fatta, si va' a calcolare un modello M_{ij} per ciascuna lunghezza osservata *i* dei payload spediti alla porta *j*. Questo schema può portare a diversi problemi, innanzi tutto la dimensione totale del modello può diventare molto larga. Questo a causa del fatto che le lunghezze dei payload sono associate con dei file media che possono essere misurati anche in gigabyte). Inoltre, per tali motivi, la distribuzione dei byte per i payload di lunghezza *i* può essere molto simile a quella dei payload di lunghezza *i-1* e *i+1,* differenti solo di un byte. Memorizzare un modello per ciascuna lunghezza può diventare quindi ridondante e causare spreco di spazio. Infine per determinate lunghezze ci possono non essere abbastanza campioni. Questo problema di sparsità porterà i dati a generare una distribuzione empirica che sarà una stima inaccurata della distribuzione vera portando ad avere un rilevatore difettoso.

Sono state presentate due possibili soluzioni a questi problemi. Una soluzione per il problema della sparsità è quella di rilassare i modelli assegnando un fattore α più alto alle deviazioni standard permettendo una maggiore variabilità dei payload. L'altra soluzione è di prendere "in prestito" i dati dai modelli "vicini" per incrementare il numero dei campioni.

Si confronti due modelli vicini utilizzando la distanza semplice di Mahalanobis per misurare la similarità delle loro distribuzioni di frequenza medie dei byte. Se la loro distanza è più piccola di una determinata soglia *t*, si fondono questi due modelli. Questa fusione si può fare come descritto tramite apprendimento induttivo, aggiornando le medie e le varianze dei due modelli per produrre una nuova distribuzione aggiornata.

Vi è ancora però la possibilità che la lunghezza dei dati di test sia fuori dalla gamma di tutti i modelli calcolati. Per questi dati di test, si useranno modelli la cui lunghezza della gamma è più vicina ad essi.

Apprendimento non supervisionato

Il modello utilizzante la distanza di Mahalanobis può essere anche usato con un algoritmo di apprendimento non supervisionato. Quindi diventa possibile il training dei modelli anche se è presente rumore nei dati di training. L'assunzione si basa sul fatto che i payload anomali sono una minoranza nei campioni di training e la loro distribuzione è differente da quella di un payload "normale". Questi payload anomali possono essere identificati nel training set e le loro distribuzioni rimosse dal modello. Per far questo si applicheranno i modelli appresi dal training set al fine di rilevare i valori errati. Questi payload anomali hanno una distanza molto più larga rispetto al profilo dei campioni "medi" normali, e quindi appariranno statisticamente errati. Dopo avere identificato le anomalie possiamo rimuoverle e aggiustare i modelli, aggiornando le distribuzioni di frequenza dei

modelli calcolati e rimuovendo i conteggi relative alle frequenze dei byte che sono apparsi nei dati di training anomali.

Conclusioni

Si sono presentati due metodi diversi per la costruzione di *anomaly based intrusion detection system*, queste due metodologie sono ancora in fase di sperimentazione, ma dai risultati ottenuti sembrano poter fornire una buona base per la costruzione di futuri sistemi di rilevamento autonomi. Gli obiettivi attuali della ricerca sembrano mirare ad una riduzione dei costi computazionali di queste tecniche al fine di poterle rendere più adatte ad approcci real-time e ad un aumento dell'accuratezza ed efficienza nel rilevamento. Sebbene questi sistemi a livello sperimentale sembrano fornire buoni risultati, mi interrogo se questo corrisponderà ad un effettivo miglioramento della sicurezza, in quanto caratteristica propria degli attacchi informatici è l'adattarsi alla tecnologia di difesa al fine di apparire come un comportamento normale e lecito.

Riferimenti:

[HLMS90]

R. Heady, G. Luger, A. Maccabe, and M. Servilla.

The architecture of a network level intrusion detection system.

Technical report, Computer Science Department, University of New Mexico, August 1990.

[LS98]

W. Lee, S. Stolfo, K. M. 1998.

Mining audit data to build intrusion models.

In R Agrawal, P. Stolorz, G. P.-S., ed., Proc. Fourth Intl. Conf. Knowledge Discovery and Data Mining, 66--72. AAAI Press.

[LS00]

W. Lee, S. Stolfo. 2000.

A framework for constructing features and models for intrusion detection systems.

In ACM Transactions on Information and System Security (TISSEC) Vol. 3 , Issue 4 (November 2000) Pages 227 - 261 .

[LSM99]

W. Lee, S. Stolfo, K. W. Mok, 1999.

Mining in a Data-flow Environment: Experience in Network Intrusion Detection

Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press.

[WS04]

Ke Wang, Salvatore J. Stolfo

Anomalous Payload-based Network Intrusion Detection

RAID 2004, pages. 203-222

[AIS93]

R. Agrawal, T. Imielinski, and A. Swami.

Mining association rules between sets of items in large databases.

In Proceedings of the ACM SIGMOD Conference on Management of Data, pages 207-216, 1993.

[AS94]

R. Agrawal and R. Srikant.

Fast algorithms for mining association rules.

In Proceedings of the 20th VLDB Conference, Santiago, Chile, 1994.