POLITECNICO DI MILANO
V Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

UN SISTEMA DI INTRUSION DETECTION BASATO SU TECNICHE DI APPRENDIMENTO NON SUPERVISIONATO

Relatore: prof. Sergio M. Savaresi

Correlatore: prof. Giuseppe Serazzi

Tesi di laurea di:

Stefano Zanero

Matr. 634200

Anno accademico 2001/2002

In memoria di mio padre,
che mi ha insegnato tutto,
e dedicata a mia madre,
che mi ha donato tutto.

RINGRAZIAMENTI

È difficile pensare di ringraziare tutti coloro che in qualche misura hanno contribuito alla nascita e al completamento di questo lavoro. Un primissimo ringraziamento va al professor Sergio Savaresi, che ha incoraggiato questa idea fin dalla sua nascita, che ormai risale a molti mesi fa, proponendomi di trasformarla in una tesi, e che mi ha seguito costantemente durante il lavoro, insieme al professor Giuseppe Serazzi, che ha suggerito idee, consigli e correzioni.

Devo anche ringraziare il professor Marco Colombetti, che mi ha consentito di studiare la problematica degli IDS (pur sotto un diverso approccio) come progetto per il corso di Ingegneria della Conoscenza e dei Sistemi Esperti. Alcuni dettagli su tale ricerca sono riportati come complemento in una appendice di questo lavoro di tesi. È d'obbligo ringraziare, per i loro contributi riguardo l'etologia, la dottoressa Marzia Possenti, ed il professor George W. Barlow del Department of Integrative Biology all'università della California a Berkeley.

Ringrazio Andrea Cocito e Orlando Bassotto, rispettivamente direttore e assistente del dipartimento di bioinformatica dell'Istituto di Oncologia Europeo, per i numerosi suggerimenti relativi agli algoritmi di clustering e alla loro performance.

È doveroso ringraziare (per le sue circostanziate critiche) il dottor Marcus J. Ranum, che in uno stadio molto iniziale di questa ricerca mi ha aiutato a focalizzarmi sui problemi irrisolti. Un ringraziamento va anche al dottor Diego Zamboni, in precedenza del CERIAS dell'università Purdue, e attualmente ricercatore presso il laboratorio IBM Research di Zurigo, per aver ascoltato pazientemente un riassunto di questo lavoro e avere offerto utili suggerimenti ed incoraggiamenti. Un ringraziamento anche al dottor Luca Deri dell'università di Pisa per le informazioni su NTOP e per aver voluto leggere una bozza di questo lavoro.

Un ringraziamento particolare a Norberto Carnelli, per aver letto il lavoro di tesi e aver contribuito a indicare punti oscuri, suggerendo formulazioni alternative.

A tutti coloro (e sono veramente tanti) che a questo lavoro hanno contribuito un'idea, un suggerimento, o che si sono simpaticamente prestati come ascoltatori di teorie a tratti incomprensibili anche per me, o che mi hanno sopportato (e non deve essere stata impresa da poco) in questi mesi di lavoro e concentrazione, va il mio sincero ringraziamento.

SOMMARIO

Questa tesi presenta una architettura originale per la realizzazione di un sistema di Intrusion Detection (rilevamento di intrusione) basato sulla rilevazione di anomalie nel traffico di una rete informatica.

Viene analizzato innanzitutto il problema della sicurezza dell'informazione, e degli attacchi contro i sistemi informatici. A partire da questa analisi, viene introdotto il tema della realizzazione degli Intrusion Detection System, apparati rivelatori di intrusione nelle reti informatiche. Viene studiato lo stato dell'arte di questi sistemi, e vengono mostrate tutte le problematiche che essi devono affrontare.

A partire da queste osservazioni viene introdotto il concetto di sistemi IDS basati sulla rilevazione delle anomalie, e vengono analizzati sinteticamente gli approcci alla loro realizzazione. Sulla base di tali osservazioni viene proposta poi una architettura innovativa per la realizzazione di un sistema IDS basato su algoritmi di apprendimento non supervisionato, operante sul flusso di traffico di una rete TCP/IP. Di tale architettura viene presentata una visione d'insieme, e vengono studiati i due stadi di cui essa è composta: uno stadio di clustering per il contenuto dei pacchetti, e uno stadio di analisi del flusso di traffico.

Per il problema di clustering vengono analizzati gli algoritmi disponibili in letteratura, e vengono proposte prove di carattere sperimentale, relative all'applicabilità dell'approccio e alle problematiche di carattere computazionale. I risultati ottenuti dimostrano l'applicabilità di questo approccio originale per estrarre informazioni dal payload dei pacchetti IP in transito su una rete.

Per il problema di correlazione, vengono studiati i prototipi e gli studi già realizzati, e viene mostrato come nessuno di essi consideri il contenuto dei pacchetti. Vengono proposte prove, di carattere sperimentale e preliminare, dell'utilità di integrare questi algoritmi con le informazioni ricavate dal primo stadio.

INDICE

INTRODUZIONE 1

1.1 OBIETTIVI 1
1.2 CONTRIBUZIONI ORIGINALI 1
1.3 SCHEMA DEL LAVORO 2

SICUREZZA DELL'INFORMAZIONE 5

2.1 SICUREZZA DELL'INFORMAZIONE 5
2.2 IL PARADIGMA C. I. D. 5
2.3 IMPLEMENTAZIONE TRADIZIONALE 7
2.4 IMPERFEZIONI DEL PARADIGMA 10
2.5 UNA TASSONOMIA DELLE MINACCE 14
2.5.1 L'INCIDENTE INFORMATICO COME PROCESSO 15
2.5.2 AGGRESSORI, BERSAGLI, OBIETTIVI 15
2.5.3 L'ATTACCO, GLI STRUMENTI E I RISULTATI 19
2.5.4 OSSERVAZIONI CONCLUSIVE 26

IDS: CONCETTO E STATO DELL'ARTE 29

3.1 COS'È UN INTRUSION DETECTION SYSTEM ? 29
3.2 TASSONOMIA DEGLI INTRUSION DETECTION SYSTEM 31
3.2.1 I DUE APPROCCI PRINCIPALI 31
3.2.2 HOST BASED, NETWORK BASED 32
3.2.3 ON-LINE (IN-LINE, REAL-TIME), OFF-LINE 34
3.2.4 SISTEMI CENTRALIZZATI E DISTRIBUITI, AGENTI 34
3.3 CARATTERISTICHE DESIDERABILI PER UN IDS 35
3.3.1 REATTIVITÀ 35
3.3.2 SICUREZZA INTRINSECA: SURVIVABILITY, ROBUSTEZZA 36
3.3.3 INTERAZIONE CON IL PERSONALE E FLESSIBILITÀ 37
3.3.4 ADATTABILITÀ AD ATTACCHI NUOVI 37
3.3.5 SCALABILITÀ E CONCORRENZA 38
3.4 PROBLEMI TIPICI DI UN IDS 39
3.4.1 INDIVIDUAZIONE DI ATTACCHI NUOVI O MODIFICATI 39

3.4.2 ERRORI: FALSI POSITIVI, FALSI NEGATIVI 41
3.4.3 IL FATTORE TEMPO: SEQUENZIALITÀ E CORRELAZIONE 42
3.4.4 RICOSTRUZIONE DEL TRAFFICO E AMBIGUITÀ 43
3.5 SISTEMI ED ARCHITETTURE RILEVANTI.....44
3.5.1 IDES, NIDES, EMERALD 45
3.5.2 ISOA: INFORMATION SECURITY OFFICER ASSISTANT 45
3.5.3 MIDAS E WISDOM&SENSE 46
3.5.4 DIDS: DISTRIBUTED IDS 46
3.5.5 SNORT 47

UN IDS BASATO SULL'APPRENDIMENTO NON SUPERVISIONATO 51

4.1 OBIETTIVO DEL PROGETTO 51
4.2 RICERCHE PRECEDENTI 52
4.3 FORMULAZIONE DEL PROBLEMA 53
4.4 ARCHITETTURA PROPOSTA.....55
4.4.1 ARCHITETTURA DI MASSIMA 55
4.4.2 IL PRIMO STADIO 56
4.4.3 IL SECONDO STADIO 57
4.5 ADDESTRAMENTO E VALUTAZIONE 58
4.5.1 IL PROBLEMA DELLA VALUTAZIONE 58
4.5.2 SET DI DATI PER LA VALUTAZIONE E L'ADDESTRAMENTO 61
4.5.3 ADDESTRAMENTO 62

PROGETTO DEL PRIMO STADIO 65

5.1 DEFINIZIONE DEL SOTTOPROBLEMA DA ANALIZZARE.....65
5.2 CLASSE DI ALGORITMI DA ANALIZZARE 66
5.2.1 ALGORITMI DI CLUSTERING NON SUPERVISIONATO 66
5.2.2 ALCUNI REQUISITI PER UN ALGORITMO CANDIDATO 67
5.3 ANALISI SINTETICA DEGLI ALGORITMI DI CLUSTERING 69
5.3.1 PROPRIETÀ DEGLI ALGORITMI DI CLUSTERING 69
5.3.2 TASSONOMIA DEGLI ALGORITMI DI CLUSTERING 71
5.3.3 ANALISI SINTETICA DEGLI ALGORITMI DI CLUSTERING 72
5.3.4 ALGORITMI DI RIDUZIONE DIMENSIONALE 93
5.4 ESPERIMENTI DI IMPLEMENTAZIONE.....95
5.4.1 METODI E OBIETTIVI 95
5.4.2 SCELTE EFFETTUATE 96
5.4.3 PRIMO TEST: ANALISI DI UN LOG DI APACHE 97

5.4.4 INTERLUDIO: DECODIFICA E ANALISI DI DATI TCP 102
5.4.5 ALGORITMO S.O.M. 109
5.4.6 ALGORITMO PRINCIPAL DIRECTION DIVISIVE PARTITIONING 119
5.4.7 ALGORITMO K-MEANS 126
5.5 CONCLUSIONI 130

PROGETTO DEL SECONDO STADIO 131

6.1 DEFINIZIONE DEL SOTTOPROBLEMA DA ANALIZZARE 131
6.2 ANALISI DEI POSSIBILI ALGORITMI 132
6.2.1 TIPI DI ALGORITMI E LORO APPLICABILITÀ 132
6.2.2 INSTANCE-BASED LEARNING 133
6.2.3 S.O.M. 133
6.2.4 P.H.A.D. 135
6.2.5 METODI STATISTICI 135
6.2.6 PERCETTRONE MULTISTRATO 136
6.3 OSSERVAZIONI, PROPOSTE ED ESPERIMENTI 137
6.3.1 OSSERVAZIONI SUGLI ALGORITMI ESAMINATI 137
6.3.2 LA NOSTRA PROPOSTA 137
6.3.3 ESPERIMENTI PROOF-OF-CONCEPT 139
6.3.4 IMPLEMENTAZIONE PROOF-OF-CONCEPT 143
6.3.5 ALCUNE OSSERVAZIONI SULL'ADDESTRAMENTO 148
6.4 CONCLUSIONI 149

CONCLUSIONI 151

APPENDICE - UN IDS COMPORTAMENTALE 153

INTRODUZIONE 153
SPUNTI DALL'ETOLOGIA E DALLA PSICOLOGIA 153
TERMINOLOGIA COMPORTAMENTI, PATTERN, DEVIAZIONI 153
LE MOTIVAZIONI DELL'AZIONE 154
LA SCELTA DELL'AZIONE 155
FIXED ACTION PATTERN 156
DISPLAY 157
INDIVIDUAZIONE DEI COMPORTAMENTI 158
DEFINIZIONE DEL PROBLEMA 158
UNA PROPOSTA METODOLOGICA 159
PASSO UNO: COSA OSSERVARE ? 160

PASSO DUE: LE CATEGORIE 160
PASSO TRE: CONOSCENZE DI BASE SUL DOMINIO 161
ETOGRAMMI 161
CATENE COMPORTAMENTALI E MODELLI DI MARKOV 161
CONCLUSIONI: UNA PROPOSTA DI MODELLO.....165

BIBLIOGRAFIA 167

1

INTRODUZIONE

1.1 OBIETTIVI

L'obiettivo di questo lavoro di tesi è studiare concretamente la fattibilità dell'utilizzo di algoritmi di apprendimento non supervisionato per la realizzazione di un sistema di intrusion detection.

Il concetto di "Intrusion Detection System" è stato sviluppato da James P. Anderson in un suo rapporto tecnico del 15 aprile 1980 [1]. L'idea alla base di un intrusion detection system (o IDS) è che sia possibile rilevare i segnali caratteristici di un'intrusione in un sistema informatico allo scopo di dare l'allarme.

Nel seguito si mostrerà come l'avvento dell'era dell'informazione ed il moltiplicarsi dei sistemi critici e vulnerabili, connesso all'accelerazione nello sviluppo di nuovi attacchi, abbia rapidamente reso inadeguati gli approcci basati sull'uso di conoscenza umana. Per questo motivo molti sono stati i tentativi di realizzare sistemi innovativi in questo campo, senza grande successo. Saranno presentati i principali approcci seguiti ed i loro difetti, e saranno presentate chiaramente le motivazioni che ci hanno spinto a studiare gli algoritmi di apprendimento non supervisionato come nuovo tipo di approccio. Verrà proposta una architettura innovativa per la realizzazione di un IDS basato sull'individuazione di anomalie, network-based, sfruttando algoritmi di apprendimento non supervisionato. Verranno altresì proposte varie verifiche sperimentali a sostegno della fattibilità di tale approccio. Verranno infine delineati spunti per ulteriori ricerche possibili.

1.2 CONTRIBUZIONI ORIGINALI

Analizzando gli articoli scientifici scritti sul tema è possibile notare che, nonostante la fiducia quasi unanimemente espressa dagli esperti nella possibilità teorica di realizzare sistemi di apprendimento non supervisionato per il problema

dell'intrusion detection, nella pratica ben pochi di questi esperimenti sono stati realizzati concretamente.

In aggiunta, molti degli articoli sembrano concentrarsi sulla applicazione di un particolare tipo di algoritmo (ad esempio le reti neurali, oppure gli algoritmi di pruning) piuttosto che sulla selezione di una classe di algoritmi di soluzione, con un successivo confronto pratico per stabilire quale sia l'algoritmo più adatto, tralasciando in qualche modo il fine e prestando attenzione esagerata al mezzo.

Per questo motivo, abbiamo deciso di presentare un modello completo di IDS basato su algoritmi di apprendimento non supervisionato, e di focalizzare la nostra attenzione su una indagine pragmatica relativa alla realizzabilità e all'efficienza di un simile concetto. Non siamo partiti perciò con l'intenzione di dimostrare la validità di un approccio, o di realizzare un prototipo di immediata applicabilità, ma con la curiosità di studiare in modo aperto l'ampio spettro di soluzioni disponibili.

Abbiamo inoltre deciso, contrariamente alla maggioranza degli autori da noi consultati, di utilizzare dati realistici e completi, ritenendo che dati simulati o falsi non possono essere ritenuti indicativi, in quanto le nostre aspettative di progettisti si rifletterebbero inevitabilmente nel modello utilizzato per generare i dati, falsando quasi inconsciamente l'esperimento.

Questo approccio sperimentale ci ha consentito di analizzare approfonditamente il tema, e di proporre e realizzare l'uso di tecniche di clustering non supervisionato per la classificazione del payload dei pacchetti IP. Non abbiamo trovato, in letteratura, precedenti analisi di questa possibilità. Questa proposta, e lo studio sulla sua efficacia, costituiscono il nucleo innovativo fondamentale di questo lavoro di tesi.

1.3 SCHEMA DEL LAVORO

Lo schema di questo lavoro ricalca fedelmente le varie fasi del suo sviluppo. Iniziamo (capitolo 2) con un'analisi generale del problema della sicurezza dell'informazione e delle minacce che minano alla base tale sicurezza. Da questa analisi prendiamo spunto (capitolo 3) per una disamina approfondita del problema dell'intrusion detection, con una descrizione per quanto possibile completa dello stato dell'arte, cercando di evitare un elenco di prodotti e di articoli ma piuttosto proponendo una tassonomia dei differenti approcci e dei loro immancabili problemi.

Proporremo poi (capitolo 4) l'utilizzo di algoritmi di apprendimento non supervisionato per la realizzazione di un sistema di intrusion detection ad individuazione di anomalie. Analizzeremo con attenzione i precedenti approcci al

problema della anomaly detection, le loro proposte innovative e le loro limitazioni. Proporremo poi lo schema concettuale di quella che secondo noi può essere una architettura funzionale per applicare algoritmi di questo tipo al problema. Giustificheremo con motivi qualitativi ed intuitivi le nostre scelte, abbastanza originali.

In seguito (capitoli 5 e 6) descriveremo le funzioni, i requisiti e gli aspetti fondamentali di entrambi gli stadi di cui è composta l'architettura che proponiamo. Per il primo stadio in particolare (nel capitolo 5) esamineremo una serie di algoritmi candidati, e proporremo i risultati di una serie di prove effettuate su ciascuno di questi algoritmi e su dei dati reali. Proporremo quindi le nostre scelte e le nostre valutazioni.

Per il secondo stadio esamineremo una serie di approcci già realizzati (capitolo 6) e i modi in cui il primo stadio da noi progettato può integrare utilmente tali approcci migliorando le possibilità di individuare eventi anomali all'interno di un sistema di rete. Proporremo alcune prove sperimentali di carattere preliminare relative ai miglioramenti che la nostra architettura a doppio stadio potrebbe apportare rispetto alle architetture originali.

In conclusione (capitolo 7) cercheremo di valutare il potenziale di questa architettura, offrendo, speriamo, interessanti spunti per ricerche successive.

In appendice, proponiamo come punto di vista complementare una ricerca relativa alla possibile struttura di un sistema IDS di anomaly detection realizzato mediante tecniche di analisi comportamentale.

2

SICUREZZA DELL'INFORMAZIONE

2.1 SICUREZZA DELL'INFORMAZIONE

Non è certo necessario dimostrare che nel nostro mondo una parte sempre maggiore di ciò che facciamo, di ciò che gestiamo, o che trasmettiamo, viene realizzata, controllata, o coadiuvata dall'utilizzo di sistemi informatici. Per questa ragione è particolarmente importante capire quanto possiamo fidarci di questi sistemi. Questo problema diviene sempre più critico, mano a mano che il valore delle informazioni scambiate in forma digitale cresce.

La sicurezza dell'informazione ha attraversato un lungo periodo di gestazione in cui veniva considerata un'arte più che una scienza, da apprendere tramite l'esperienza e non tramite lo studio. Solo pochi studi pionieristici stendevano le basi dei metodi formali che negli ultimi anni sono stati sviluppati. Questo, unito all'intrinseca difficoltà della materia, spiega il motivo per cui gran parte degli approcci al tema della sicurezza dell'informazione sono basati su esperienza e conoscenza euristica più che su tecniche raffinate. L'unico settore della sicurezza che vanta una lunga tradizione nell'uso dei metodi formali è il campo della crittografia, che, non sorprendentemente, è molto più avanzato e stabile degli altri.

Se questo metodo pragmatico di procedere ha funzionato per anni, lo sviluppo esplosivo dei sistemi collegati in rete e di dispositivi con potenza e versatilità francamente inimmaginabili solo pochi anni fa ha reso necessario, urgente, lo sviluppo di un'ingegneria della sicurezza informatica, dove il termine "ingegneria" va inteso nel suo senso tradizionale di scienza applicata, con definizione formale di concetti, metodi e tecniche conosciute, esplorate da una ampia letteratura scientifica, e globalmente diffuse.

2.2 IL PARADIGMA C. I. D.

Sia nella letteratura accademica che nella pratica, gli obiettivi della sicurezza dell'informazione sono normalmente descritti in termini di confidenzialità, integrità, e disponibilità dell'informazione stessa (paradigma C. I. D. o, usando i termini inglesi ***confidentiality***, ***integrity*** e ***availability*** , paradigma C.I.A.).

La **confidenzialità** dell'informazione può essere definita in vari modi, ma è sostanzialmente la capacità di un sistema di rendere le proprie risorse accessibili soltanto a coloro che sono autorizzati ad accedervi. In modo formale, possiamo dire che la confidenzialità costruisce una relazione tra ogni risorsa del sistema e uno o più soggetti autorizzati all' accesso in lettura.

Per fare un parallelo col mondo reale, si può pensare ai livelli utilizzati dalla NATO per i documenti "classified": Confidential, Secret, e Top Secret. Ogni persona ha (o non ha) una "clearance" per un determinato livello. Inoltre, per ogni documento si decide il "need to know", ovvero la necessità per una determinata persona di accedere alle informazioni contenute. Questo tipo di modello è stato trasportato nell'informatica mediante i Trusted Computer System Evaluation Criteria (TCSEC) [2].

L'**integrità** è invece definita come la capacità di un sistema di rendere possibile solo alle persone autorizzate la modifica delle sue risorse dei suoi dati, e anche a queste solo in modo autorizzato onde garantire la consistenza di questi dati con le funzioni gestite dal sistema. Anche questa definizione implica la creazione di una relazione tra ogni risorsa e un insieme di utenti autorizzati, tuttavia ci si rende immediatamente conto della difficoltà di definire, e ancor più di garantire, che ogni utente modifichi in modo corretto soltanto i dati che la sua posizione gli da autorità di modificare.

Si può inoltre notare come la creazione di una relazione tra ogni risorsa e gli utenti che gli hanno accesso è perfettamente equivalente alla costruzione di una relazione tra ogni utente e l'insieme di risorse a cui può accedere. Sarebbe umanamente più comprensibile utilizzare il secondo formato, ma per ragioni storiche ed implementative i sistemi reali utilizzano controlli d'accesso che riflettono la prima forma. È proprio questo semplice problema di prospettive algebriche a creare molti dei problemi di misconfigurazione di cui discuteremo nel seguito.

La **disponibilità** è in qualche modo un obiettivo in conflitto con gli altri due, in quanto impone che le relazioni precedenti non siano semplicemente dei " solo se ", ma dei "se e solo se"; ovvero, il sistema deve garantire accesso soltanto le persone autorizzate, ma non deve mai negare l'autorizzazione a un accesso corretto. Inoltre, l'accesso deve essere consentito tempestivamente, dove la definizione esatta di

"tempestivamente" dipende dal dominio del problema. Esiste in letteratura [3] il concetto di Maximum Waiting Time proprio introdotto a questo proposito.

Spesso vengono utilizzati i termini "exposure" per indicare il fallimento nella condizione di confidenzialità, "compromise"per il fallimento nella relazione di integrità, e "denial of service" per la mancanza di disponibilità, tuttavia su questa terminologia non c'è lo stesso accordo che sulla precedente [4].

Altri testi definiscono, in aggiunta alle tre condizioni elencate, ulteriori obiettivi quali "privacy", o "identificazione", ma bastano poche e semplici valutazioni per comprendere come, in generale, si tratti di semplici sottocasi di questi obiettivi più generici, oppure di tecniche utili a raggiungere tali obiettivi. Per esempio la privacy non è altro che la confidenzialità dei dati sensibili di l'utente, mentre il "non ripudio" è la capacità di un sistema di provare oltre ogni dubbio che una transazione è avvenuta. Esempi di tecniche spesso confuse con i fini della sicurezza sono l'identificazione e l'autenticazione degli utenti, oppure l'accounting, la possibilità per l'amministratore di sistema di ricostruire a posteriori gli eventi e le responsabilità.

È in ogni caso necessario notare che il concetto di "sicurezza dell'informazione" si estende ben oltre l'informatica in senso ristretto, ovvero l'hardware, il software e le reti di computer, andando a interessarsi del sistema informativo nel suo complesso, e quindi dell'intero processo di gestione dei dati e delle informazioni messo in atto da un'organizzazione.

Rientrano quindi a pieno titolo nell'ingegneria della sicurezza anche considerazioni di sicurezza ambientale, gestione dell'informazione non digitale, procedure nella gestione delle risorse umane, eccetera, eccetera. In effetti, i filtri della sicurezza dell'informazione operano all'interfaccia tra le persone ed i processi del sistema informativo, e devono gestire entrambe le variabili. Come si può immaginare, si tratta di un processo complesso, costoso e prono a sviste anche gravi.

2.3 IMPLEMENTAZIONE TRADIZIONALE

L'implementazione tradizionale del paradigma C.I.D. (che si può osservare in quasi tutte le applicazioni informatiche e in buona parte dei processi di qualsiasi sistema informativo moderno) è un approccio che viene usualmente riassunto nel motto "Who are you ? What can you do?".

Siti web, sistemi operativi pensati per l'uso in rete, sistemi di accesso remoto, telefoni cellulari, ormai qualsiasi tipo di applicazione accessibile da molti utenti e con esigenze anche minime di sicurezza richiede all'utente di "effettuare un login",

ovvero in qualche modo di provare la propria identità, prima di accedere ai servizi del sistema. L'identità digitale che viene associata all'utente ha privilegi e impostazioni particolari che limitano i servizi e le operazioni ad esso disponibili.

Anche altri servizi e applicazioni tipicamente usati per garantire la sicurezza in rete (firewall, virtual private network, e simili) operano con un meccanismo simile, identificando un utente o una macchina in rete (per mezzo di un login e di una password, per mezzo dell'indirizzo IP, mediante uno scambio di token crittografici, questo è secondario rispetto alla nostra discussione), e applicando un insieme di regole alle operazioni che questa persona o macchina può eseguire.

Per effettuare questa associazione esistono diversi tipi di paradigmi, divisi in due macro-categorie: paradigmi DAC (Discretionary Access Control) e paradigmi MAC (Mandatory Access Control), con una comparsa marginale di paradigmi definiti RBAC (Role Based Access Control).

Nel caso dei DAC, la regola di associazione è semplice. Ogni oggetto ha un proprietario. Il proprietario può fare ciò che vuole dell'oggetto, delegando diritti a chi desidera all'interno del sistema su di esso, fino ad arrivare alla possibilità di trasferire ad un altro utente l'ownership dell'oggetto. Esiste in generale un amministratore di sistema che ha l'ownership degli oggetti di sistema, oltre a poter violare le restrizioni d'accesso. I sistemi DAC sono sicuramente quelli a cui siamo più abituati: i meccanismi dei permessi UNIX e le ACL degli oggetti WindowsNT sono sistemi DAC.

In un sistema MAC gli oggetti hanno un loro livello di segretezza (possono essere, in generale, più livelli in diverse categorie, ma per comprendere il meccanismo basta pensare a un singolo livello omnicomprensivo). Gli utenti hanno un loro livello di accesso. Esiste una figura che nei sistemi normali non è prevista, ovvero quella del "security officer", che è l'unico a poter amministrare i livelli d'accesso. In generale le regole che vengono usate per stabilire i permessi d'accesso sono il "no read up" (l'utente non può leggere informazioni con un livello d'accesso superiore al suo) e il "no write down" (l'utente non può scrivere informazioni verso un livello d'accesso inferiore al suo).

Il modello MAC forse più famoso, quello di Bell-LaPadula [5], descrive così le regole d'accesso:

1. Security Rule: un utente non può leggere informazioni il cui livello d'accesso sia più elevato del suo

2. *-property: un utente non può spostare informazioni da un livello di sicurezza a un livello di sicurezza più basso (ma può spostarle ad uno più alto). Questo coincidentalmente include la regola di "no write down".

In realtà, dal momento che il sistema di Bell-LaPadula prevede la possibilità di definire degli insiemi di categorie e non un singolo "livello di segretezza", il concetto di livello di sicurezza "superiore" o "inferiore" va sostituito con quello di dominanza. Ovvero, per esempio, un utente potrebbe avere come clearance: {top-secret("ricerca"), secret("finanziario", "personale")}. In tal caso, sarebbe dominante su un documento etichettato {secret("ricerca","finanziario")}, ma non su un documento etichettato {top-secret("finanziario")}. Potrebbe inserire informazioni nel secondo tipo di documento ? No, perché le sposterebbe dal suo contesto personale a un documento che, pur non essendo dominato dal suo contesto, nemmeno lo domina o lo eguaglia.

Per necessarie ragioni di praticità, il sistema di Bell-LaPadula, nella sua descrizione originaria, prevedeva la possibilità di utilizzare un sistema DAC a permessi per gli oggetti "non confidenziali", e la possibilità di definire dei "trusted subjects" (per fare un esempio: lo spooler della stampante) in grado di violare le regole d'accesso per svolgere compiti di sistema.

I sistemi di tipo RBAC, molto studiati negli ultimi anni, tendono a superare entrambi questi concetti definendo il paradigma dei "ruoli". I diritti d'accesso agli oggetti sono definiti in base a dei ruoli (per esempio, l'accesso al database "paghe" è consentito al ruolo "direzione_risorse_umane"), e uno o più ruoli sono associati agli utenti in base alle loro esigenze lavorative. Il sistema sembrerebbe essere rigoroso e flessibile. Ricerche e sperimentazioni sono in corso.

Quasi ogni sistema di sicurezza tradizionale è una implementazione più o meno evoluta del paradigma base di identificazione e gestione dei privilegi: il sistema chiede a un utente (che può essere una persona o un altro sistema), di identificarsi, e usa questa identità per attivare un insieme di regole operative. Ad esempio, i sistemi di controllo d'accesso da remoto funzionano tutti in base a questo schema; ma anche sistemi apparentemente diversi, come i firewall, controllano il traffico in ingresso e in uscita da una rete, "identificandone" la provenienza ed applicando opportune politiche di filtraggio.

Questo approccio ha successo perché è quasi del tutto ortogonale al problema: non importa quale sia nello specifico il protocollo o il servizio che si offre, uno schema di autenticazione e privilegi può esservi imposto senza troppa fatica; in generale, anche troppi software e protocolli vengono dapprima fatti funzionare sotto l'assunzione che

chiunque abbia accesso illimitato, dopodichè si prova a creare dei controlli successivi che "aggiungono" uno strato di sicurezza imposta *sul* sistema invece di essere costruita *nel* sistema. Il problema è che il paradigma onnipresente che usiamo per imporre la sicurezza è minato fin dalle fondamenta.

2.4 IMPERFEZIONI DEL PARADIGMA

Nel famoso "mondo ideale" di ogni controesempio, i requisiti semplici e lineari del paradigma C.I.D. sarebbero facilmente implementabili, progettando un insieme adeguato di soluzioni di sicurezza mediante il metodo usuale di progettazione per requisiti e specifiche, creando una "security policy" omnicomprensiva che partendo dai requisiti C.I.D. descriva le specifiche atte a garantirli. Sfortunatamente, questa soluzione semplice è ben poco realistica.

Come nota Baker in un ottimo saggio [6] "[the] philosophy of protection [...] depends upon systems to: behave predictably (they should do what we think they will do); be available when we need them; be safe (they should not do what we don't want them to do); be capable of protecting our data from unwanted disclosure,modification,and destruction; respond quickly. In other words, systems should be trustworthy". Significativamente il titolo dell'opera è "Fortezze costruite sulla sabbia".

I sistemi reali usati nel mondo di oggi sono larghe reti interconnesse di macchine diverse, che utilizzano software e sistemi operativi multipurpose diversi tra di loro, spesso non progettati per soddisfare le nostre specifiche, ma che tuttavia sono assolutamente essenziali per il corretto funzionamento dell'organizzazione: sarebbe molto bello fare a meno di qualsiasi accesso dall'esterno ai nostri sistemi database, ma in questo modo perderemmo le funzionalità consentite, per esempio, da un sito web che mostra informazioni calcolate dinamicamente. Come se ciò non bastasse, oltre al compromesso tra funzionalità e sicurezza un ulteriore punto di equilibrio va raggiunto tra requisiti hardware e software aggiuntivi (con i relativi costi, spesso molto elevati) e le risorse economiche disponibili.

Nessuna compagnia al giorno d'oggi, per quanto grande ed economicamente florida, può permettersi anche solo di progettare un sistema operativo o un sistema DBMS: il software viene per quanto possibile acquisito sotto forma COTS (Commercial, Off The Shelves), e quasi mai costruito internamente. Se da un lato questo ha economicamente senso, per la disciplina della sicurezza dell'informazione si tratta di un vero incubo in cui componenti di base del sistema informativo stesso sono sottratti a qualsiasi valutazione approfondita di qualità e rispondenza ai

requisiti. Le problematiche di progettazione della sicurezza diventano duplici: da un lato, garantire e preservare la sicurezza dei singoli componenti del sistema informativo, dall'altro integrare in modo sicuro componenti diverse e non sempre interoperabili.

Assicurare, in qualsivoglia modo, che un simile aggregato di componenti diversi e di varia qualità sia conforme a una security policy completa basata sul paradigma C.I.D. è una battaglia persa. Spesso (anche se non sempre) è possibile rilevare correttamente e completamente i requisiti di sicurezza del sistema e tradurli in appropriate specifiche all'interno di una policy, ma rimane difficile, o meglio impossibile, validare completamente il comportamento del sistema contro dette specifiche.

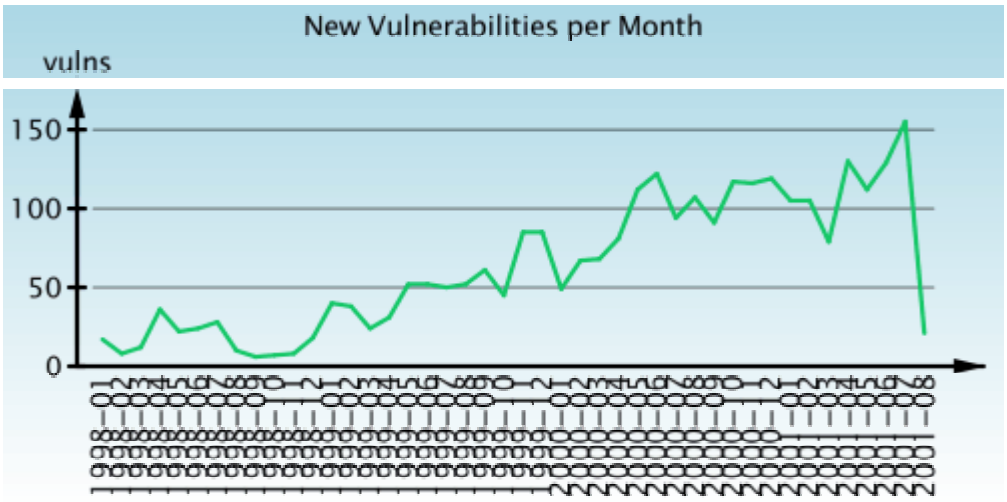
Innanzitutto, normalmente il processo di validazione del software viene effettuato presupponendo un utente "cooperativo", che magari commette degli sbagli, ma che sostanzialmente intende usare il software seguendo le regole: insomma, non si parte mai dall'assunto che un utente possa voler saltare le nostre specifiche. Idem dicasi per l'ambiente d'esecuzione: il software viene progettato, nei casi migliori, per resistere a errori nel sistema. Molto di rado il software viene messo sotto test per valutare la sua resistenza a deliberati attacchi di utenti malevoli. Non è necessario essere degli esperti per capire come questi assunti, normalmente innocenti, siano assolutamente irragionevoli nelle condizioni operative di cui si occupa la sicurezza dell'informazione: il caso "pessimo" della sicurezza è proprio la presenza di un intruso, malevolo, che cerca deliberatamente qualsiasi debolezza per penetrare il sistema, e che è spesso abile ed esperto almeno quanto il progettista delle misure di sicurezza.

In secondo luogo, come è noto, vi è comunque un problema profondo nel test delle applicazioni software. Per citare Dijkstra: "Program testing can be used to show the *presence* of bugs, but never to show their absence" [7]. Se esiste un campo in cui questa verità viene quotidianamente dimostrata, è proprio quello della sicurezza informatica. Un interessante progetto denominato FUZZ, sviluppato all'università del Wisconsin da Miller e altri [8], invia alle applicazioni l'equivalente di un ingresso a rumore bianco. I risultati possono essere definiti, alternativamente, sorprendenti o inquietanti: quasi tutte le applicazioni sottoposte al test hanno presentato problemi più o meno seri.

Alcuni autori hanno discusso [9] se abbia senso cercare di "dimostrare la sicurezza" di un sistema operativo (e, quindi, di qualsiasi altro programma). Per questo si utilizza spesso il concetto di "assurance" [10], ovvero, nell'impossibilità di

stilare metriche della sicurezza in sé si cerca di "garantire la sicurezza" mediante valutazioni legate al processo di gestione della sicurezza.

Senza bisogno di procedere a dimostrazioni accademiche, tuttavia, il semplice numero delle vulnerabilità rilevate nel software e nei sistemi operativi più diffusi (di cui riportiamo un significativo grafico *mensile* qui sotto, tratto dal noto database di vulnerabilità di SecurityFocus [11]) è indice che anche processi ben stabiliti di ingegneria del software e di testing, pur fondamentali per ridurre al massimo il numero di problemi, non sono sufficienti a garantire realisticamente la sicurezza delle applicazioni.



Numero di nuove vulnerabilità scoperte ogni mese e segnalate a SecurityFocus

Nonostante la continua evoluzione di nuove pratiche per ridurre al minimo la "window of exposure" [12], ovvero la finestra di esposizione in cui le vulnerabilità sono conosciute ma non esiste o non è applicata una soluzione appropriata, la battaglia pare ben lontana dall'essere vinta.

Nell'articolo citato si evidenziano ad esempio i vantaggi (e gli svantaggi) offerti dal concetto di "full disclosure", tanto caro al movimento open source, ovvero la pratica mediante la quale vengono rivelati alla comunità degli sviluppatori i dettagli completi delle vulnerabilità rinvenute. Da un lato questo conduce, alla lunga, ad una maggiore conoscenza dei meccanismi di vulnerabilità e ad una migliore prevenzione, dall'altro dissemina anche nelle "mani sbagliate" le conoscenze (o parte delle conoscenze) necessarie per un attacco. Per qualsiasi procedura di gestione delle vulnerabilità possiamo immaginare, l'esperienza ci insegna che esisteranno comunque delle finestre di esposizione, quando non delle vere e proprie controindicazioni. E ad ogni modo, riguardo alla reale lunghezza della finestra di esposizione, si può apprendere una terribile verità dal caso del virus Nimda, che

aggrediva una vulnerabilità di I.I.S. (Internet Information Server, il web server di Microsoft) per cui esisteva una patch da "alcuni mesi". Centinaia di migliaia di server erano ancora non protetti.

Non solo. In questi ultimi paragrafi abbiamo volutamente ristretto il nostro discorso dall'analisi della sicurezza del sistema informativo nel suo complesso a quella delle singole applicazioni e componenti che ne fanno parte. Ampliando di nuovo lo sguardo, scopriremo che stiamo trascurando quello che tutte le statistiche, oltre al consenso quasi unanime degli operatori del settore [13], indicano come l'anello più debole della catena della sicurezza: le persone che interagiscono col sistema informativo.

Ad esempio, sappiamo che il paradigma "Who are you ? What can you do ?" basa gran parte della propria sicurezza sull'identificazione corretta delle persone che accedono al sistema. Nella più classica delle tricotomie, l'identificazione può essere basata su tre possibili fattori: qualcosa che il soggetto sa (per esempio una password), qualcosa che il soggetto ha (un "token", una tessera, un badge, una chiave hardware), o qualcosa che il soggetto è (per esempio mediante un sensore biometrico).

La "progressione" di sicurezza di questi sistemi è evidente: la password può essere condivisa tra più persone (anche in modo innocente, si pensi alla tipica situazione di un laboratorio di calcolo universitario) o compromessa involontariamente; il token è uno solo, ma può essere smarrito, rubato e a volte contraffatto; la prova biometrica è la più determinante ed infatti si sta diffondendo, sia pure molto lentamente a causa dei costi non indifferenti e delle implicazioni di natura giuridica (per la legge 675 sulla privacy i dati biometrici sono altamente personali). La stragrande maggioranza delle applicazioni utilizza ancora oggi il fattore "password" per l'identificazione, che è il più insicuro: gli utenti amano parole chiavi brevi, facili da ricordare e da indovinare, e magari amano appuntarsele sulla tastiera (il dramma dei post-it con i codici di accesso incollati sui monitor è sulla scena della sicurezza da anni, e le repliche non accennano a finire). L'educazione degli utenti alla sicurezza di base è impresa improba, continua e complessa.

Tutto questo senza considerare il caso tutt'altro che raro [14] (anzi, si tratta probabilmente del più grosso problema di sicurezza in qualsiasi organizzazione !) in cui l'utente "autorizzato" si comporta in modo non confacente alla fiducia che gli è stata concessa, ovvero quello che viene indicato con l'intraducibile espressione inglese "disgruntled employee" (lavoratore scontento, approssimativamente). Non è nemmeno necessario che lo faccia con un intento criminale, in realtà: le misure di sicurezza sono spesso viste come un intralcio al proprio lavoro, e cercare di aggirare quella limitazione che proprio dà fastidio inutilmente è uno dei passatempi degli

utenti di computer in tutto il mondo. Il problema è proprio nella percezione di "utilità". Nessuno di noi si lamenta delle misure di sicurezza in aeroporto o del codice segreto della tessera ATM (Automatic Teller Machine, di cui l'esempio più noto in Italia è il Bancomat), perché li riteniamo utili a garantire la nostra sicurezza.

Infine, oltre ai casi in cui un utente volontariamente viola le misure di sicurezza (sapendo o meno di commettere un illecito molto pericoloso) vi sono i numerosi casi in cui un malintenzionato può sfruttare l'ingenuità di un utente legittimo per guadagnare accesso ad una rete. Kevin Mitnick, detto "Condor", uno dei più famosi pirati informatici americani, oggetto di una storica caccia all'uomo da parte dell'FBI [15], non era un esperto di sicurezza. Era un eccellente "social engineer", ovvero riusciva a convincere in pochi minuti una persona all'interno di qualsiasi organizzazione a dargli accesso.

Concludendo: per garantire la sicurezza con il paradigma C.I.D. dovremmo poter in qualche modo essere certi dell'attuazione delle politiche che lo descrivono. Non possiamo essere certi del funzionamento corretto dei sistemi software e hardware; ancora più difficile è controllare il corretto comportamento delle persone. Per citare uno dei più famosi esperti di sicurezza del mondo, Gene Spafford: "Secure web servers are the equivalent of heavy armored cars.The problem is, they are being used to transfer rolls of coins and checks written in crayon by people on park benches to merchants doing business in cardboard boxes from beneath highway bridges. Further, the roads are subject to random detours, anyone with a screwdriver can control the traffic lights, and there is no police around" [16].

Perciò, siamo costretti ad assumere il "worst case", ovvero che i sistemi informativi, a livello di software e di hardware, per non parlare delle persone che li utilizzano, sono inerentemente e inevitabilmente insicuri; che nessuna politica di sicurezza può essere efficacemente imposta su un sistema informativo complicato ed interconnesso; che ogni sistema verrà prima o poi compromesso, in maniera più o meno grave. Questo ci porta inevitabilmente a dover studiare un approccio alternativo e complementare a quello praticato fin qui. Tale approccio verrà proposto in dettaglio nel successivo capitolo 3.

2.5 *UNA TASSONOMIA DELLE MINACCE*

Cercheremo di dare una tassonomia e una nomenclatura per gli attacchi ai sistemi informatici e per i possibili perpetratori di questi attacchi. Premettiamo che questa non intende essere in alcun modo una guida esaustiva ai tipi di attacco (compito che si rivelerebbe improbo, visto il numero quotidiano di innovazioni in questo settore), ma piuttosto cercherà di delineare una serie di direttrici tipiche d'attacco, e di

tecniche "esemplari", universalmente note, per farne uso. Vi sono molte ricerche che propongono una tassonomia di attacchi (per esempio [17]), ma quasi tutte sono più o meno incomplete e contraddittorie. Non esistendo un vero consenso sull'argomento, ci limitiamo a proporre alcuni concetti di base che serviranno nelle discussioni che seguono.

2.5.1 L'INCIDENTE INFORMATICO COME PROCESSO

Uno dei più interessanti ed informati tentativi di descrivere un incidente informatico e dare una tassonomia delle sue componenti è stato fatto dal CERT/CC [18]. In particolare, viene descritto l'incidente informatico come un processo con componenti multipli:

L'intrusione come processo

Consideriamo questa schematizzazione tra le più sensate ed accorte. La ricerca del CERT approfondisce ognuno degli elementi dello schema, ma nel seguito ci distacchiamo da esso integrandolo o sostituendolo con il nostro punto di vista e le nostre esperienze, e coerentemente con la nostra finalità espositiva.

2.5.2 AGGRESSORI, BERSAGLI, OBIETTIVI

"Conoscere l'altro, e se stessi – cento battaglie, nessun rischio. Non conoscere l'altro, e conoscere se stessi – a volte vittoria, a volte sconfitta. Non conoscere né l'altro né se stessi – ogni battaglia è un rischio certo" [19]. Le parole del saggio Sun Tzu, maestro cinese dell'arte della guerra, ci devono fare riflettere. Per capire l'andamento di qualsiasi scontro bisogna conoscere le due parti in gioco, e nel campo di battaglia della sicurezza vi sono aggressori e aggrediti, o attaccanti e difensori. Ponendoci nell'ottica dei difensori, conoscere "noi stessi" significa conoscere i sistemi che usiamo (e su questo abbiamo già speso del tempo) ma soprattutto conoscere cosa dobbiamo proteggere. Conoscere il nemico significa essere consci delle sue capacità e dei suoi obiettivi. Per questo ci pare utile affrontare in modo coordinato l'inizio e la fine del processo dell'incidente informatico, ovvero l'aggressore e i suoi obiettivi.

La prima nota da fare è che (eccettuati rari sottocasi) i sistemi informativi non vengono aggrediti da altri sistemi, ma da *persone*. Ciò implica che l'analisi delle loro motivazioni e delle loro metodologie sia una disciplina delle scienze sociali e della psicologia più che dell'informatica. Tuttavia, come abbiamo già notato, la disciplina della sicurezza dell'informazione è una di quelle discipline che si situano all'interfaccia tra l'uomo e il sistema informativo, affrontando una miscela unica ed interdisciplinare di problemi paragonabili, ad esempio, a quelli riscontrati nello studio delle HCI (Human Computer Interfaces).

Per cominciare, possiamo fare una macrodistinzione tra minacce **esterne** e minacce **interne**. Come abbiamo già detto, infatti, una gran parte degli attacchi provengono da dentro le mura di qualsiasi organizzazione. Già definire i "confini" di una singola organizzazione diviene oggi difficile: concetti come le "extranet" che estendono il sistema informativo mediante Internet fino ad interfacciarlo con quello dei partner, dei clienti e dei fornitori, rendono molto labile il confine tra ciò che è interno ad una impresa e ciò che è esterno. Un aggressore interno ha tutti i vantaggi, in quanto molto spesso opera a scatola aperta: sa già quali sono i punti deboli, e come è strutturata l'architettura dei sistemi, tutte cose che dall'esterno vanno scoperte poco alla volta.

Tra gli aggressori interni possiamo individuare due categorie principali:

- **disgruntled employees**, sono persone che stanno coscientemente aggredendo l'organizzazione, perché hanno qualche conto in sospeso, o perché sono stati corrotti, o ancora peggio si sono fatti assumere appositamente per avere una opportunità di violarne la sicurezza;

- **trasgressori ingenui:** semplicemente cercano di abusare in qualche modo delle risorse loro concesse, per esempio violando le linee guida aziendali sull'uso responsabile di Internet, sul software da installare sulle postazioni di lavoro aziendali, o simili. Sono pericolosi perché nel loro tentativo di violare le difese aziendali potrebbero involontariamente fornire una porta d'accesso ad altri intrusi ben più pericolosi (ad esempio, scaricando ed utilizzando software di dubbia provenienza potrebbero installare un "troiano").

Tra gli aggressori esterni possiamo fare due tipi di distinzioni, la prima è una distinzione a livello di capacità, e distinguiamo:

- **esperti**: dotati di conoscenze tecniche approfondite, capaci di elaborare attacchi innovativi o di "inventare" procedimenti nuovi; molto spesso sono più esperti dei sistemi che stanno attaccando di quanto non siano le persone

incaricate di difenderli; se volessimo tracciarne un quadro psicologico li definiremmo come controllati, freddi, sul limite del paranoico, desiderosi di sentirsi "in controllo" della situazione;

- **script-kiddies**: privi di reali conoscenze tecniche, studiano ed utilizzano "exploit", attacchi preconfezionati rilasciati da esperti; il loro quadro psicologico è ben più agitato, emotivo, possono lasciarsi prendere dalla paura e agire irrazionalmente.

Tra le due categorie vi sono ovviamente infinite gradazioni intermedie.

Una seconda distinzione dei nemici si può fare secondo i loro fini:

- nemici con un **obiettivo** preciso: sul sistema informativo sono contenuti dati che vogliono rubare, oppure vogliono informazioni su una singola persona, vogliono modificare o eliminare qualche informazione scomoda, o in ogni caso mirano ad un obiettivo specifico e a lasciare quante meno tracce possibile della propria presenza;

- nemici che vogliono fare soltanto un **danno**, per rabbia, perché sono stati pagati per farlo, o perché sono l'equivalente telematico dei vandali;

- persone che stanno **giocando**: esplorano sistemi che non gli appartengono e se li vedono vulnerabili provano a bucarli come "sfida" alle proprie capacità. Non va assolutamente sottovalutata la potenzialità di questi aggressori "ludici", e nemmeno la loro pericolosità. Per quanto spesso non vogliano fare alcun danno, o addirittura sperino di "dare una mano" agli amministratori delle macchine che attaccano, un errore può sempre succedere.

Dobbiamo anche valutare quali siano i possibili bersagli di un aggressore con un obiettivo preciso:

- I **dati**: l'aggressore cerca di violare la confidenzialità delle informazioni che il sistema contiene, oppure di modificarle abusivamente. I dati possono essere copiati per essere venduti, oppure dare vita a un rapimento con riscatto (l'attaccante ottiene abusivamente i segreti industriali di una vittima, e minaccia di renderli pubblici se non gli verrà versato un cospicuo riscatto, per esempio);

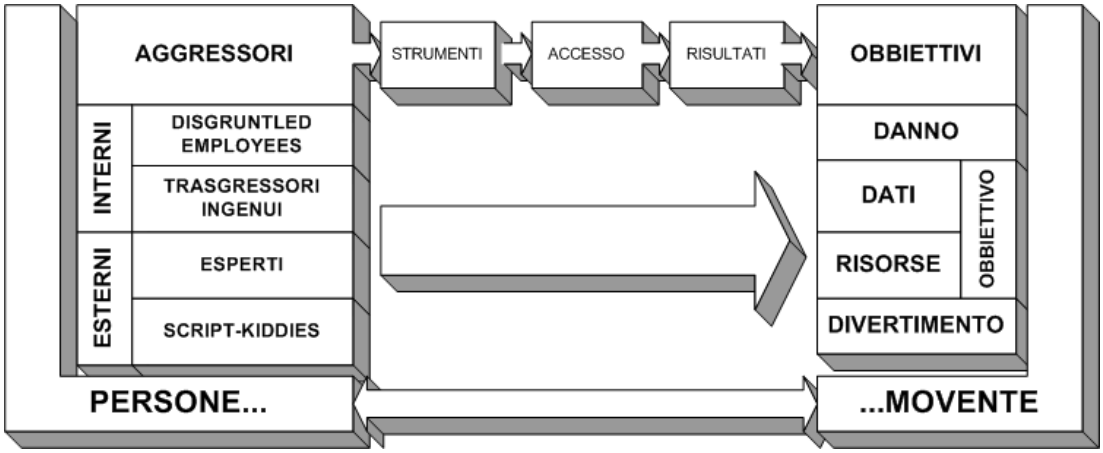
- I **sistemi**: l'aggressore mira alle risorse telematiche in sé più che al loro contenuto. Ad esempio, moltissimi script-kiddie cercano macchine facili da

penetrare che possano essere usate come "deposito" di materiale pornografico e/o piratato, o che possano essere abusate per fare spam, o per disporre di connessioni permanenti per qualsivoglia altra ragione;

Le finalità ultime di una persona possono essere, come abbiamo visto, moltissime: dal guadagno diretto al guadagno derivante dalla vendita di informazioni, dalla rappresaglia al divertimento; non ci avventuriamo in questo campo che è proprio della criminologia e su cui in realtà è tuttora in corso una fervida attività di studio, anche da parte di ricercatori italiani [20]. Tutto ciò costituisce il *movente dell'attacco informatico*, legame fondamentale tra aggressori e obiettivi, molto spesso trascurato perché si tende a trascurare che la componente umana è il cuore dell'attacco. Non lo strumento, non l'aggressione in sé, ma la molla che spinge qualcuno ad aggredire un sistema.

Ogni organizzazione deve perciò considerare che sarà valutata da ogni tipo di nemici, e che se risulterà interessante anche solo da un punto di vista sarà sicuramente attaccata ed aggredita. Bisogna mettersi nell'ottica dell'aggressore, anzi, degli aggressori, non nell'ottica dell'organizzazione: ciò che magari dal nostro punto di vista è poco importante, per qualcuno può essere molto utile. Per esempio, poche organizzazioni si rendono conto di quanto attrattivo possa essere, per un ragazzino che vuole distribuire file musicali MP3, lo spazio su disco e la banda di connessione ad alta velocità che possiedono in abbondanza.

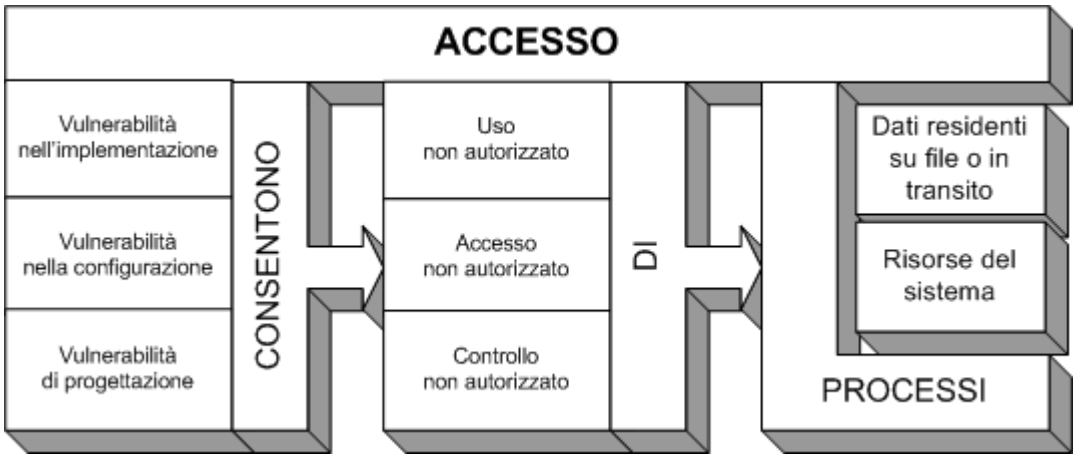
Pertanto la nostra tassonomia di attaccanti e motivazioni di attacco (che diverge completamente da quella proposta dal CERT/CC e che secondo noi si può annoverare tra i contributi originali di questa ricerca, pur non costituendone il cuore) si può schematizzare come segue:



La relazione tra aggressori ed obiettivi, mediante i concetti di persona e movente

2.5.3 L'ATTACCO, GLI STRUMENTI E I RISULTATI

Il cuore dell'attacco informatico, come abbiamo detto, è ottenere in qualche modo un accesso non desiderato al sistema: ottenere la possibilità di leggere dati confidenziali, ottenere la possibilità di modificare dati protetti, o ancora ottenere la possibilità di intralciare l'accesso legittimo ai dati da parte di altri utenti. Pertanto, il punto focale sono i *dati* e le *risorse* gestite dal sistema e le applicazioni o *processi* che il sistema svolge. L'aggressore vuole *accedere* in modo non autorizzato o comunque non previsto a questi dati. Per farlo, sfrutta necessariamente una *vulnerabilità*. Ecco lo schema del CERT, da noi integrato:



Schema dell'accesso non autorizzato a un sistema informatico

Dal momento che lo scopo è violare in qualche modo le restrizioni di confidenzialità, integrità o disponibilità imposte sui dati, l'aggressore deve, necessariamente, *attivare*, *controllare*, oppure fare *fallire* uno o più dei processi del sistema informatico, in un modo *non previsto* nel design originario del sistema.

In poche parole, l'uso o l'accesso non autorizzato ai processi del sistema consentono all'intruso di violare la sicurezza dei dati. Ma come è possibile che avvenga ciò ?

Esistono tre tipologie fondamentali di vulnerabilità in un sistema informatico:

- *vulnerabilità nell'implementazione dei programmi*, che consentono all'aggressore di far comportare il programma in maniera diversa da quella attesa. Non possiamo riportarne un elenco completo, per cui rimandiamo a testi specifici [21], ma le più comuni sono:

  - errori nella *validazione dell'input*, ovvero nel filtraggio dei dati che arrivano da fonti inaffidabili (ad esempio, gli utenti). Un esempio comune sono le pagine web che gestiscono form collegate all'esecuzione di query SQL sui database. A volte gli autori dimenticano di controllare che non vengano inseriti caratteri speciali nei campi, rendendo possibile all'utente "modificare" non solo i parametri, ma il codice SQL in sé;
  - problematiche di *buffer overflow*, ovvero la possibilità che un insieme di valori troppo lungo venga inserito in un buffer di lunghezza prefissata, andando a influire sullo stack [22], alterando il flusso di esecuzione del programma, o sullo heap del programma, corrompendone le variabili;
  - problemi nelle *chiamate e comunicazioni tra processi* e nella *gestione della memoria*: ad esempio, passaggi in chiaro di dati molto sensibili che possono essere intercettati da qualche curioso;
  - problemi di *gestione dei privilegi* di esecuzione, in particolare per i demoni di rete che hanno bisogno dei privilegi di root in alcuni punti della loro esecuzione;
  - attacchi relativi al *timing* di determinate operazioni critiche (le cosiddette "race condition");
  - problemi relativi all'implementazione degli *algoritmi criptografici*, e di utilizzo di fonti di casualità di bassa qualità.
- *vulnerabilità nella configurazione:* molti programmi per essere considerati sicuri debbono essere opportunamente configurati ed adattati alle situazioni specifiche in cui vengono inseriti. Spesso è proprio nella fase di configurazione che amministratori inesperti compiono gravi leggerezze, che rendono inutili tutte le contromisure di sicurezza. Per esempio, la condivisione di unità a disco in ambiente Windows senza alcuna forma di autenticazione è da anni costantemente nella lista delle "20 vulnerabilità più diffuse" gestita dal SANS institute [23].
- *vulnerabilità nella progettazione* di una determinata applicazione, sistema, o protocollo, che la rendano insicura per come è stata creata. Esempio tristemente noto e recente è il protocollo criptografico WEP, Wired Equivalent Privacy, progettato per rendere sicure le comunicazioni sulle reti Wireless LAN realizzate secondo lo standard IEEE 802.11b; un errore di design rende tale protocollo estremamente debole e facile da decodificare.

Abbiamo volutamente ordinato tali classi in ordine crescente di "livello concettuale", e di difficoltà di correggere l'errore una volta individuato: un buffer overflow può essere corretto mediante una patch; un errore di configurazione, una

volta scoperto, può essere sistemato; un algoritmo come WEP è irrecuperabile, e può solo essere abbandonato in favore di un altro. Le abbiamo anche scelte in modo che fossero del tutto ortogonali. Una corretta implementazione di un progetto errato, o una cattiva configurazione di un servizio concettualmente ben studiato e appropriatamente implementato, sono tutti casi egualmente possibili.

All'interno di queste macroclassi, esistono una quantità enorme di vulnerabilità più o meno note; esistono intere classi di vulnerabilità che si applicano in maniera trasversale a decine di software diversi, a volte aventi funzioni totalmente differenti. Nonostante esistano copiosi volumi di "best practices" su come scrivere un'applicazione sicura, su come progettare in modo sicuro sistemi e protocolli, le stesse, vecchie vulnerabilità e gli stessi problemi noti continuano a rispuntare, rendendo tristemente evidente che anche se fosse possibile enumerare tutti i principi della programmazione e della progettazione sicura, nondimeno il fattore umano interverrebbe a minare qualsiasi sforzo.

Per sfruttare una vulnerabilità al fine di far agire un programma in modo erroneo e guadagnare così l'agognato accesso ai dati, l'intruso interagirà con il sistema seguendo una procedura che viene comunemente denominata "exploit". A volte, per gli attacchi svolti mediante dei particolari tool, script o programmi, si usa il termine "exploit" per indicare proprio questi ultimi, chiamando "exploit techniques" invece le sequenze prolungate di azioni tese a violare un sistema.

È da notare che il punto di partenza di questi attacchi può essere vario. In generale si pone una netta distinzione tra attacchi di tipo **remoto** e attacchi di tipo **locale**. Non si intende distinguere, come è ovvio, che l'attaccante stia seduto o meno alla tastiera della macchina attaccata. Un attacco da remoto è un attacco per il quale l'aggressore non parte da un accesso a livello comandi alla macchina. Un attacco di tipo locale è invece perpetrato a partire dall'accesso a livello comandi, che tuttavia in un sistema operativo di rete può essere usualmente consentito anche da remoto (tramite telnet o SSH, per esempio).

In generale questi due attacchi corrispondono a due fasi successive, denominate **break-in** e **privilege escalation**. Break-in è l'attacco con cui un intruso partendo da remoto ottiene un accesso sufficiente per portare attacchi di tipo locale. Il termine "escalation" fa ovviamente riferimento alla salita da un livello di "ospite", a quello di utente effettivo del sistema, a quello di amministratore: in particolare, ottenere i privilegi amministrativi su un sistema viene detto "to root", dal termine che indica il super-user dei sistemi UNIX-like. Esistono però attacchi che da remoto consentono a un intruso di ottenere direttamente i privilegi amministrativi, e vengono

significativamente categorizzati come **"remote root exploits"**. Ovviamente questo tipo di attacchi sono tra i più pericolosi.

Bisogna altresì notare che alcuni attacchi (per esempio molti denial-of-service) non cercano di guadagnare accesso al sistema ma semplicemente di negarlo ad altri utenti. Per questo molti di questi attacchi possono essere eseguiti da remoto senza bisogno di un vero e proprio break-in.

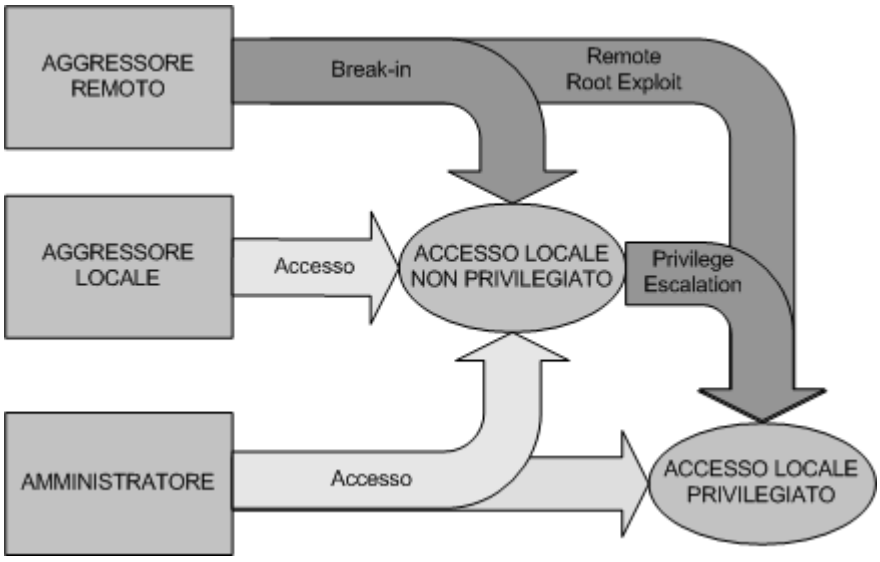


Illustrazione schematica dei cammini tipici per l'accesso a un sistema informatico; in colore più chiaro i cammini leciti, in scuro quelli illeciti

Per ottenere questo tipo di accessi "illeciti", gli aggressori usano una vasta serie di strumenti, e non crediamo certo di poterne offrire un elenco esaustivo. Tuttavia proviamo ad elencarne le tipologie principali:

- *Strumenti di packet forgery*: vasta categoria di programmi, sia generali (che consentono di costruire pacchetti del tipo desiderato dall'utente) sia molto specifici (che creano pacchetti strutturati appositamente per sfruttare una specifica vulnerabilità). Molti "remote exploit tools" ricadono in quest'ultima categoria.
- *Port-scanner* e *vulnerability scanner*: strumenti che inviano ad una o più macchine pacchetti TCP/IP per cercare di scoprire quali servizi sono disponibili. I vulnerability scanner aggiungono a questa funzione una base di conoscenze per determinare se tali servizi sono vulnerabili ad attacchi noti, sia mediante funzioni di raccolta di informazioni, che mediante veri e propri tentativi di attacco.

- *Normali client d'accesso*: non sono poche le vulnerabilità che non richiedono niente altro che un normale browser, un client telnet, o qualche altro tipo di client, e un po' di conoscenze da parte dell'aggressore per essere sfruttate. Ricadono in questa categoria, per esempio, gran parte delle vulnerabilità dovute a cattiva configurazione. A volte si usano dei client *modificati*: per esempio un client FTP da linea di comando, congiunto ad uno script, può essere utilizzato per fare molteplici tentativi di accesso tentando di "indovinare" una password.
- *Programmi* da eseguire (o far eseguire con l'inganno) su un host remoto per ottenere gli effetti più vari. Un noto detto della sicurezza informatica vuole che "una volta che qualcuno è riuscito, con l'inganno o mediante qualche tecnica, ad eseguire o a farti eseguire un programma di sua scelta sulla tua macchina, quella non è più la tua macchina".
- *Agenti software* autonomi, con varie funzioni, sono dei programmi simili ai precedenti ma sono in grado di attuare da soli un processo di compromissione del sistema.
- *Artefatti fisici*: un tempo erano molto di moda i "wiretap", piccoli morsetti applicati sui cavi di una rete di telecomunicazioni per intercettare e inserire traffico abusivamente, al punto che molte canaline di rete sicure venivano integrate con impianti ad aria compressa per rilevare eventuali forature. Caduta in disuso negli anni '90, la pratica sta rinascendo con le reti wireless, in cui l'uso di apparecchiature di intercettazione dei segnali anche molto semplici consente risultati sorprendentemente efficaci.

Strumenti e accessi si combinano in un numero sorprendente di scenari di intrusione possibili, di cui elenchiamo le variazioni più comuni ed i relativi risultati.

ATTACCHI DA REMOTO

- **Attacchi a demoni e servizi di rete:** il tipo di exploit forse più conosciuto, al punto tale da essere spesso l'unico preso in considerazione. L'idea alla base di questi attacchi è che, in assenza di un accesso diretto al livello comandi del sistema, l'aggressore cerchi di fare in modo che un processo già in esecuzione, spesso con elevati privilegi, esegua al posto suo delle operazioni sul sistema.
    - *Mediante l'invio di pacchetti artefatti:* spesso i progettisti di applicazioni di rete dimenticano che è possibile per chiunque forgiare ed iniettare pacchetti all'interno di Internet. Molti servizi si sono dimostrati vulnerabili a particolari pacchetti malformati.
    - *Mediante l'interazione con l'interfaccia pubblica:* molti attacchi vengono portati interagendo con l'applicazione, demone o servizio,

esattamente come qualsiasi utente normale farebbe, ma inviando dati o comandi malformati che provocano effetti indesiderabili. Possiamo ricordare i bug Unicode di I.I.S. (Internet Information Server) di Microsoft, il cui meccanismo di exploit era una richiesta http malformata; ma anche le vulnerabilità delle applicazioni web-based.

- *Grazie a misconfigurazioni:* la più tipica delle vulnerabilità da remoto è una misconfigurazione o un'errata concezione della sicurezza. Pagine web amministrative lasciate on-line senza password, directory FTP aperte per l'upload…
- *Bruteforcing:* l'aggressore cerca, per tentativi, di individuare i codici di accesso di un servizio, nella speranza di trovarne di deboli o corti. Contrariamente a quanto si crede, è un attacco relativamente poco usato perché facilmente individuabile.

- **Attacchi all'infrastruttura:** con questo termine ho voluto indicare attacchi che non si concentrano sul sistema aggredito, ma che cercano di sfruttare alcune debolezze dell'infrastruttura di rete che lo circonda, per esempio
  - *Sniffing:* I dati su Internet viaggiano in chiaro e vengono trasmessi da un punto all'altro rimbalzando per una serie di sistemi di cui non conosciamo l'affidabilità. Non solo, ma su una rete locale Ethernet, tranne in alcuni casi, tutte le workstation sullo stesso segmento di rete vedono passare tutto il traffico destinato anche alle altre macchine. La pratica dello sniffing consiste proprio nell'analizzare il traffico abusivamente intercettato, in generale alla ricerca dei codici d'accesso per le macchine.
  - *Hijacking:* l'hijacking delle connessioni TCP è un tipo d'attacco tanto pericoloso quanto difficile da realizzare, che consente a un aggressore di "rilevare" la connessione stabilita tra due macchine, e continuare ad usarla. Per fare un esempio, un intruso potrebbe attendere che A abbia stabilito una connessione telnet con B per poi "inserirsi" al posto di A e sfruttare il collegamento per fare ciò che vuole su B.
  - *Man-in-the-middle:* un attacco in cui l'intruso, C, riesce a interporsi in maniera perfetta tra due sistemi, A e B, che stanno comunicando via rete. In pratica A parla con C, illudendosi di parlare con B; B pure parla con C, interpretando le sue risposte come provenienti da A.
  - *Spoofing:* la pratica dello spoofing prende le mosse dall'osservazione che non c'è, in IP, un meccanismo di autenticazione del mittente: l'indirizzo IP sorgente può pertanto venire tranquillamente falsificato. A livello di TCP sono previsti alcuni meccanismi che rendono più difficile questo attacco (la presenza di un numero pseudocasuale di sincronizzazione, ad esempio) "alla cieca", ma se l'aggressore vede

tutti i pacchetti mentre passano (mediante sniffing) la difficoltà non è insormontabile.

- **Denial-of-service da remoto:** attacchi spesso meno complessi dei precedenti, in quanto in questo caso l'obiettivo dell'aggressore è "semplicemente" impedire a chiunque altro di utilizzare il servizio aggredito.
    - *Saturazione delle risorse:* sia del particolare servizio, che dell'intero sistema. Un esempio di attacco di questa classe può essere il SYN-flood, ovvero un flusso continuo di richieste di connession (SYN) che non vengono confermate con il terzo ACK dell'handshake a tre vie. La macchina bersaglio terrà traccia di ognuna di queste connessioni nell'inutile attesa dell'ACK di risposta, ed esaurirà la sua capacità di accettare nuove connessioni
    - *Utilizzo di pacchetti artefatti:* come per il caso dell'uso di pacchetti malformati per attacchi finalizzati a ottenere un accesso, determinati pacchetti malformati possono invece ottenere l'effetto di far bloccare un servizio o una applicazione, ottenendo un denial-of-service.
    - *Utilizzo dell'interfaccia pubblica del demone:* facendo richieste impossibili, malformate, o semplicemente subissando di richieste molto lunghe il servizio che si vuole bloccare, è possibile impedire l'accesso agli utenti legittimi

ATTACCHI LOCALI

- **Mediante esecuzione di comandi e script:** dall'interno di un computer ed utilizzando i comandi disponibili agli utenti si possono compiere un gran numero di possibili attacchi, cercando di ottenere privilegi "migliori", per esempio l'accesso in lettura a tutti i file, oppure i privilegi d'amministrazione.
- **Mediante la compilazione e l'esecuzione di exploit sull'host vittima:** esistono alcuni bug di servizi locali e/o del sistema operativo che possono essere sfruttati mediante l'esecuzione (in locale) di programmi di exploit.
- **Denial-of-service da locale:** lanciare un denial-of-service per un utente locale è ancora più facile: il sistema si aspetta, in genere, di dover eseguire le richieste di chi accede dall'interfaccia comandi
- **Saturazione delle risorse:** un utente locale (se non è stato limitato in qualche modo) può lanciare in esecuzione un nugolo di processi, per esempio scrivendo un piccolo programma C che lancia due "copie" di se stesso. La crescita esponenziale metterà in breve in ginocchio il sistema.
- **Altri tipi di interazione con il sistema:** quasi ogni sistema ha delle attività (per esempio, di manutenzione programmata) in grado di rallentare o bloccare l'accesso ai servizi. Inoltre, un intruso con i privilegi amministrativi può "spegnere" qualsiasi servizio gli interessi disattivare.

ATTACCHI IBRIDI

- **Utilizzo di agenti autonomi:** questi programmi agiscono da soli, in modo indipendente da chi li ha creati e lanciati, per guadagnare accesso ad altre macchine ed eseguirvi operazioni non autorizzate. L'esempio più tipico sono i virus e i worm.
    - *Virus:* sono "frammenti di codice eseguibile… che quando vengono eseguiti copiano sé stessi all'interno di altri programmi eseguibili. Quando questi ultimi vengono eseguiti, il codice virale torna in esecuzione e si diffonde sempre di più" [24]. I virus non possono essere inseriti all'interno di file di dati, però va notato che al giorno d'oggi molti programmi possono salvare istruzioni "macro" nei propri documenti, che in alcuni casi possono essere utilizzate per creare virus.
    - *Worm:* diversamente dai virus, non sono pezzi di codice che si agganciano ad altri programmi, bensì programmi che clonano se stessi, lanciandosi in esecuzione in vari modi. Spesso i worm contengono del codice "backdoor", installando sulle macchine infettate un accesso per il proprio creatore.

Per discendere dalla teoria alla pratica, consigliamo come lettura d'approfondimento i documenti del "Project Honeynet" [25], un progetto il cui scopo è studiare le tecniche e le motivazioni degli aggressori informatici, creando delle reti trappola in cui studiare i comportamenti dei pirati informatici ("honey-net" è un gioco di parole su "honey-pot", vaso di miele, riferito all'uso di un vaso di miele aperto come trappola per le mosche). Consigliamo in particolare di studiare i documenti che descrivono le sequenze di aggressioni realmente osservate, tecniche, tool e stili.

2.5.4 OSSERVAZIONI CONCLUSIVE

Per completare il quadro mancano ancora due annotazioni di carattere generale ma a nostro avviso di estrema importanza, assolutamente assenti nella tassonomia del CERT/CC.

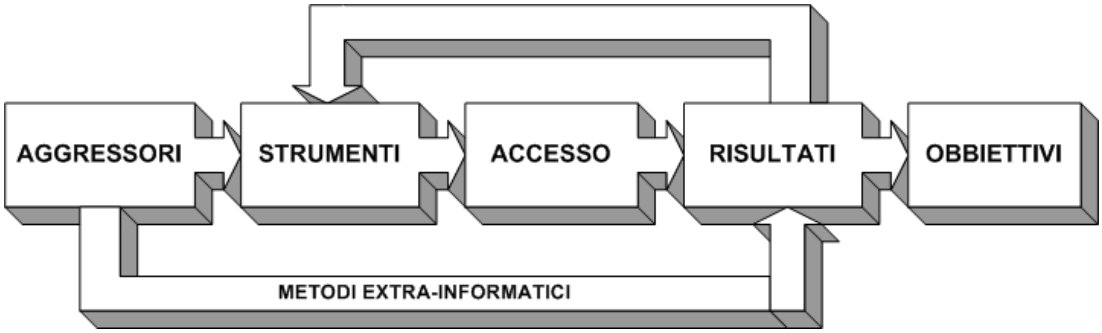
La prima: non è assolutamente detto che un attacco proceda in questa maniera lineare; anzi, sarebbe una pessima indicazione di inefficacia delle misure di sicurezza se l'attaccante potesse compiere un processo così diretto che lo porta dall'idea alla realizzazione del suo obiettivo! Solitamente la fase centrale di aggressione, ovvero l'utilizzo dei tool per guadagnare un accesso, porta a dei risultati che non sono immediatamente in linea con gli obiettivi, ma che servono per poter utilizzare altri strumenti, su altre vulnerabilità, eccetera: abbiamo già accennato il meccanismo

parlando di break-in ed escalation, ma questo ciclo aggressione-risultato-nuova aggressione è una costante delle intrusioni informatiche.

Una seconda annotazione, invece, riguarda l'utilizzo da parte degli aggressori di metodologie di attacco non strettamente informatiche: per esempio, come già accennato, l'utilizzo di metodologie di *social engineering* [26] per ottenere accessi ingannando le persone; oppure, ad esempio, il cosiddetto *dumpster diving*, la ricerca di preziosi frammenti di informazione, password, e documenti, nella spazzatura eliminata da una organizzazione; o ancora, il furto di pc portatili e palmari con memorizzate importanti chiavi d'accesso (si tratta di un fenomeno in continua crescita, secondo il rapporto CSI/FBI).

In tutti questi casi è del tutto possibile che le fasi di attacco vengano completamente oltrepassate, giungendo direttamente a dei risultati.

Illustriamo dunque in figura lo schema completo dell'incidente informatico come processo.



L'incidente informatico come processo – schema completo

3

IDS: CONCETTO E STATO DELL'ARTE

3.1 COS'È UN INTRUSION DETECTION SYSTEM ?

Quando si progetta la sicurezza di un fabbricato, si comincia a realizzare un sistema di chiusure (porte, lucchetti, chiavistelli) che consenta di "selezionare" le persone che hanno accesso alle varie aree della costruzione. Dopodichè si integra questo sistema di "difesa" con un sistema di rilevazione ("antifurto"), che consente di individuare eventuali trasgressori. Un Intrusion Detection System (IDS) è l'equivalente informatico di un antifurto.

Dopo aver introdotto il concetto con una analogia, vediamo però di esplorare più approfonditamente cosa sia e come funzioni un IDS. Abbiamo detto che, a fronte delle possibilità non secondarie di violazione della policy di sicurezza da noi prestabilita, dobbiamo trovare dei meccanismi complementari di difesa. Molto naturalmente, questi meccanismi sono quelli di individuazione degli intrusi, reazione, contenimento del danno e "chiusura" del buco nel sistema difensivo che ha reso possibile la compromissione. Ovviamente, gli IDS entrano in gioco proprio nella difficile fase di "individuazione" degli intrusi.

Se la protezione dalle intrusioni è difficile perché deve essere tenuto conto della determinazione da parte dell'intruso a violare le politiche di sicurezza, a maggior ragione l'individuazione di queste violazioni sarà doppiamente difficile. A meno di clamorosi autogol, infatti, uno dei primissimi obiettivi di qualsiasi attaccante è coprire le proprie tracce, cercare di comportarsi nella maniera più anonima possibile e soprattutto distruggere qualsiasi prova della violazione perpetrata. Il problema peggiore è che, laddove la sicurezza del sistema sia stata completamente sovvertita e l'intruso abbia raggiunto i privilegi amministrativi, improvvisamente tutti i sistemi di registrazione degli eventi (logging) e tutti i controlli inseriti nel sistema sono terribilmente inaffidabili e proni a ogni manipolazione. Proprio per questo motivo gli amministratori più preoccupati hanno l'abitudine di collegare il logging dei sistemi critici a una stampante. In questo modo, l'intruso può al massimo disabilitare il

sistema di logging, ma non cancellare selettivamente o modificare le proprie tracce. Sono però evidenti i limiti di questo tipo di artifici difensivi.

Per questo motivo è necessario che il sistema che tenta di individuare queste violazioni utilizzi un'ottica il più possibile diversa da quella utilizzata per l'implementazione del sistema di sicurezza. Laddove quest'ultimo tradizionalmente si basa, come abbiamo detto, su identificazione e privilegi ("Chi sei ? Cosa sei autorizzato a fare ?"), un IDS cerca piuttosto di capire "che cosa stai cercando di fare ?", o "perché stai operando in questo modo ?".

In altre parole, un sistema informatico ha uno scopo, per cui è stato progettato. Il paradigma C.I.D. è un metodo per controllare il sistema in maniera tale che questo scopo possa essere compiuto in modo affidabile, mediante la costruzione della relazione tra risorse e utenti. Ma la finalità del sistema informativo è proprio lo scopo originario, e il paradigma C.I.D. e la sua specifica sono soltanto dei mezzi, non delle finalità in se stesse.

Ciò che, idealmente, un IDS dovrebbe fare è "individuare quelle azioni che cercano di fare agire il il sistema in un modo diverso rispetto a quello per cui è stato progettato". Ciò significa, principalmente, cercare di individuare azioni tese a eludere le misure di sicurezza, nella speranza che le azioni degli intrusi siano sensibilmente differenti rispetto a quelle degli utenti autorizzati.

È evidente che un paradigma di questo tipo rappresenta un approccio complementare alla sicurezza rispetto a quello proposto dal paradigma C.I.D. Tornando all'esempio dell'antifurto, se il paradigma C.I.D. equivale a lucchetti e porte, e invece l'IDS è l'equivalente dell'antifurto, nessuno installerebbe mai un antifurto senza mettere anche delle buone serrature. Progettare un IDS dovrebbe essere un passo "finale" nel lungo percorso per rendere sicuro un sistema informativo, che non esime affatto dalla necessità di un'ottima sicurezza tradizionale. In effetti, i due paradigmi sono fatti per essere quanto più possibile indipendenti, nella speranza di rendere più difficile la compromissione di entrambi.

Come nota a margine, è evidente che il concetto di Intrusion Detection System da noi tracciato in questo paragrafo è paradigmatico. Non stiamo parlando di un singolo software, prodotto o meccanismo, ma dell'intera architettura di sistemi che vengono predisposti per rilevare le intrusioni. In molti casi, specialmente nell'industria, viene indicato come "Intrusion Detection System" il singolo software utilizzato per individuare le intrusioni. Questo approccio è a nostro parere limitativo, giacchè riduce una problematica e la relativa soluzione progettuale all'uso di un singolo prodotto.

Tuttavia, nel seguito di questa tesi, utilizzeremo il termine per indicare anche un singolo sistema o software che svolga compiti di intrusion detection, seguendo la convenzione tipica. Ci premeva però sottolineare questo importante concetto di IDS come architettura integrata di sistemi.

3.2 TASSONOMIA DEGLI INTRUSION DETECTION SYSTEM

In questo paragrafo cercheremo di dare una rapida panoramica dei possibili approcci al problema dell'intrusion detection. Daremo esempi concreti, ma per una più approfondita analisi dei sistemi esistenti in letteratura e sul mercato preferiamo rimandare a testi specifici [27].

3.2.1 I DUE APPROCCI PRINCIPALI

Esistono in letteratura due grandi categorie di approcci alla tematica dell'Intrusion Detection:

- **Anomaly Detection:** in questo tipo di approccio si cerca di studiare il comportamento dell'utente, confrontandolo con un profilo di comportamento "normale", modellato secondo varie tecniche. Il sistema individua in modo statistico qualsiasi deviazione "significativa", segnalandola all'amministratore come "sospetta". I sistemi di questo tipo vengono anche detti "*behavior based*"

- **Misuse Detection:** in questo tipo di approccio, viceversa, il sistema cerca di individuare direttamente un comportamento "anomalo", solitamente basandosi su qualche forma di base di conoscenza che contenga un catalogo di attacchi noti ("firme" o "signatures"). I sistemi di questo tipo vengono anche detti "*knowledge based*"

Ciascuno di questi approcci ha pregi e difetti. I sistemi basati sulla anomaly detection, per esempio, non richiedono un'immisione di conoscenza "a priori", né richiedono continui aggiornamenti delle "firme" d'attacco, essendo teoricamente in grado di rilevare i "cattivi comportamenti" sulla base di una descrizione di "normalità". Tuttavia, per costruire un modello di "comportamento normale" serve innanzitutto uno studio architetturale preciso su quale tipo di modello usare, e in secondo luogo una fase più o meno prolungata di addestramento in cui il modello viene "tarato" sullo specifico utente e sullo specifico sistema. Inoltre, questi sistemi sono drammaticamente proni ad errori e falsi positivi (daremo delle definizioni più chiare di questo argomento nel seguito).

I sistemi basati sulla misuse detection, viceversa, richiedono uno studio estensivo delle forme d'attacco per la produzione delle "firme" necessarie al loro funzionamento. Dalla qualità delle firme e dal loro aggiornamento costante dipende inevitabilmente l'efficacia del sistema (si può notare immediatamente un parallelo con i meccanismi e le problematiche tipiche del software antivirus). Stranamente, il problema dei falsi positivi ma soprattutto degli alert indesiderati affligge drammaticamente anche i sistemi di questo tipo, che dovrebbero essere meno vulnerabili. Ciò che si nota, infine, è che mantenere una base di conoscenze vasta e in continuo aggiornamento con i "pattern" degli attacchi è una impresa improba, e spesso inutile (come vedremo).

Alcuni sistemi commerciali ed accademici di intrusion detection combinano entrambi questi aspetti, in quanto ciascuno di essi possiede delle caratteristiche molto desiderabili, e sono a tutti gli effetti complementari. Vi è tuttavia un problema di metriche decisionali, e di aumento incontrollato dei falsi positivi, che ha scoraggiato fino ad ora lo sviluppo di simili soluzioni. Nella stragrande maggioranza dei casi i sistemi di Intrusion Detection commerciali sono "misuse based".

Come nota a margine vorremmo segnalare che uno degli studi esistenti sulle tassonomie degli IDS ([28], che citiamo come fonte ma che seguiamo solo in minima parte) vorrebbe inserire le categorie di "self-learning" e "pre-programmed" come ortogonali rispetto alla divisione tra "anomaly" e "misuse". Consideriamo piuttosto artificiosa questa suddivisione in quanto i sistemi di "misuse detection" non "pre-programmati" sono semplicemente inesistenti allo stato delle cose, mentre i sistemi di tipo "pre-programmato" e orientati all'anomaly detection che l'autore cita ci sembrano un po' una forzatura. Tuttavia segnaliamo l'interessante proposta dell'autore per cercare di sviluppare un sistema di misuse detection in grado di apprendere autonomamente.

Abbiamo volutamente ignorato, nella nostra tassonomia, le cosiddette "honeypot" (o trappole per mosche) od "honeynet", in quanto non le consideriamo dei veri e propri sistemi di intrusion detection (la loro finalità è differente). Tuttavia è certo che l'argomento sia in qualche modo collegato, e rimandiamo ad un volume apposito [29] per ulteriori approfondimenti.

3.2.2 HOST BASED, NETWORK BASED

Un'altra classificazione molto importante distingue tra sistemi *host-based* e *network-based*:

- I sistemi "host-based" controllano una singola macchina e a volte una singola applicazione, e dipendono dal sistema operativo (a cui sono spesso collegati strettamente, nello specifico molti moduli di intrusion detection host based sul sistema linux vengono realizzati come moduli del kernel) per tracciare chiamate di sistema, utilizzo delle risorse, comandi eseguiti e i passaggi da un livello di privilegi a un altro. Altre tipiche fonti di dati per un sistema host-based sono i log.
- I sistemi "network-based", viceversa, sono collegati in posizioni opportune ad una rete di computer, e cercano di controllare tutto il traffico che la attraversa (mediante degli sniffer di rete), cercando nel flusso di pacchetti le indicazioni di possibili attacchi.

Come sempre, entrambi gli approcci presentano vantaggi e svantaggi. Uno dei vantaggi principali di un IDS network-based è la possibilità di usare un numero di sonde relativamente piccolo per controllare anche reti di grandi dimensioni. Però vi sono alcuni tipi di intrusione che un sistema network-based non può, realisticamente, individuare: esempi tipici sono tutte le attività che coinvolgono canali crittografici. Per esempio, un IDS di rete può intercettare ed analizzare tentativi di attacco rivolte ad una web application (per esempio, mandando dati mal formattati ad un campo di una form), ma se questi ultimi viaggiano in una sessione criptata (per esempio su SSL) le "firme" dell'IDS di rete sono sostanzialmente inutili.

Uno dei potenziali vantaggi di un IDS network-based è la possibilità di operare una "passive analysis", in cui il sistema raccoglie i pacchetti di rete senza che la sua presenza sia rilevabile: operativamente questo si ottiene dotando la macchina che esegue l'IDS di una scheda di rete posta in modalità promiscua e con un indirizzo IP nullo (0.0.0.0), in maniera tale che essa non possa trasmettere alcunché ma riceva ed analizzi qualsiasi pacchetto indirizzato a qualsiasi indirizzo. In questo modo diventa molto difficile per un intruso a rilevare la presenza dello sniffer. Tuttavia una serie di ricerche [30] hanno dimostrato la possibilità di rilevare la presenza di interfacce poste in modalità promiscua.

Fino a 6-7 anni fa i sistemi più importanti erano quelli host-based. Tuttavia, l'avvento dell'era di Internet ha reso la rete uno dei vettori d'attacco più facili e probabili, dando notevole impulso allo sviluppo di sistemi network-based, il primo dei quali, storicamente, è stato Network Security Monitor, o NSM [31].

Anche questi due tipi di approcci, in realtà, non sono contrapposti ma altamente complementari. Tutte le suite per IDS più popolari al mondo, tra cui RealSecure di

ISS, o i sistemi IDS di Cisco Systems, combinano sonde di tipo host-based e network-based per cercare di coprire quanti più tipi d'attacco possibile.

Sicuramente nel futuro sarà sempre più importante un approccio misto di questo tipo, che presenta interessanti sfide per la normalizzazione dei dati derivanti da sensori diversi, per la correlazione tra essi, e per lo sviluppo di sistemi in grado di ragionare su queste viste correlate per fornire analisi di intrusione di livello più alto. Si parla di applicare al campo della intrusion detection i concetti di "multisensor data fusion" già in corso di studio nel campo, per esempio, della robotica [32].

3.2.3 ON-LINE (IN-LINE, REAL-TIME), OFF-LINE

Su questi termini ci soffermiamo perchè rappresentano spesso fonte di confusione. Vengono definiti "on-line" gli IDS che analizzano i dati mentre le azioni avvengono, mentre i sistemi di tipo "off-line" o batch analizzano i dati a posteriori. Ultimamente si tende a considerare questi ultimi più come dei tool di "computer forensics" (ovvero, di analisi a posteriori delle intrusioni, con scopi generalmente di polizia giudiziaria) che di "intrusion detection" in senso stretto, ma nella definizione che abbiamo dato di "intrusion detection system" come insieme di accorgimenti mirati a rilevare, identificare e contenere le intrusioni, è indubbio che sistemi di questo tipo siano da considerarsi totalmente inclusi.

All'interno dei sistemi on-line spesso si usa la definizione "real-time" per indicare che l'IDS è progettato per gestire un determinato flusso di dati senza perdere pacchetti e senza rimanere con un "buffer" in attesa di essere verificato.

I sistemi "in-line" invece sono una relativa novità, e spesso fanno parte della categoria dei cosiddetti "intrusion prevention systems" (vedi sotto). In sostanza, se un IDS è in-line esso è posizionato come un firewall o uno switch sulla "traiettoria" dei pacchetti, e la sua funzione più ovvia è quella di poter fungere anche da filtro oltre che da analizzatore, mentre invece gli IDS on-line di tipo tradizionale sono solitamente posizionati su una porta di rete che presenta una "copia" di tutto il traffico.

Come nota a margine, ci pare giusto ricordare che esistono sistemi come ISOA, che utilizzano un insieme limitato di trigger on-line, riservando all'operatore la richiesta di più complesse analisi off-line sui log.

3.2.4 SISTEMI CENTRALIZZATI E DISTRIBUITI, AGENTI

Un IDS può essere costituito da più processi eseguiti su varie macchine, denominati *sensori*. Un esempio notevole in campo accademico, forse uno dei primi e più documentati, è DIDS, Distributed Intrusion Detection System [33]. A volte, se tali macchine sono dedicate esclusivamente al software di intrusion detection, esse vengono chiamate *sensori* o *sonde*, e vengono spesso realizzate sotto forma di appliance integrate composte da hardware e software proprietario, ottimizzato e preinstallato.

Un IDS così strutturato può utilizzare un modello *distribuito* o *centralizzato* per l'analisi dei dati e per la loro collezione, a seconda che i vari sensori procedano per conto proprio o facciano riferimento a un server centrale. Le due cose possono essere miscelate: le sonde potrebbero analizzare ciascuna i propri dati, per poi trasmettere i risultati ad un server centrale per ulteriori rielaborazioni.

I sistemi di IDS distribuiti sono stati efficacemente descritti e progettati come sistemi basati su agenti, ad esempio nell'eccellente progetto AAFID [34] (Autonomous Agents for Intrusion Detection) sviluppato presso il gruppo COAST dell'università Purdue.

3.3 CARATTERISTICHE DESIDERABILI PER UN IDS

3.3.1 REATTIVITÀ

Una delle caratteristiche desiderate per un IDS è quella di poter in qualche modo reagire agli attacchi, ovvero cercare di bloccarli automaticamente. Questo concetto, spesso pomposamente denominato "Intrusion Prevention System", in realtà pone una serie di problemi non indifferenti.

Il primo problema è di tipo architetturale. Un sistema on-line, se non è real-time, non ha nessuna reale possibilità di reagire ad una intrusione prima che si verifichino dei danni. Se anche il sistema fosse real-time, l'unico modo efficace di consentirgli di operare preventivamente sarebbe quello di porlo in posizione "in-line", come filtro pregresso a tutto il traffico. Ciò è fattibile, tuttavia considerazioni di performance possono rendere questa soluzione irragionevole su linee ad alta capacità.

Una soluzione alternativa è un IDS on-line, real-time di tipo tradizionale, in qualche modo accoppiato con un firewall o con le ACL dei router di frontiera, in grado quindi di interrompere la comunicazione tra l'host vittima e l'attaccante. Questo sistema ha un difetto terribile: innanzitutto se del traffico legittimo fa scattare per qualsiasi motivo una firma d'attacco un utente si troverà privato di un servizio a cui ha diritto. Ma il problema diventa di gran lunga peggiore se immaginiamo un

attaccante che riesce a "forgiare" finti attacchi provenienti da un indirizzo a suo piacere: potrebbe forzare il nostro stesso IDS a chiudere fuori dal firewall indirizzi arbitrari. Inoltre, una volta che la comunicazione viene bloccata l'attacco può comunque aver avuto successo (difficilmente l'attivazione delle nuove regole sul firewall lo influenzerà). Questo può essere o meno un problema, a seconda del tipo di attacco.

Ultimo problema (in ordine di esposizione, ma non certo in ordine di importanza), come abbiamo già notato un vasto numero di attacchi avvengono dall'interno della rete, non dall'esterno. Spesso i firewall (o gli IDS in-line) sono alla frontiera della rete, e di questo genere di attacchi non si accorgerebbero mai. Viceversa un IDS piazzato sulla rete interna potrebbe individuarli, ma probabilmente non avrebbe un punto di blocco (enforcement point) su cui agire per bloccarli.

L'unica realistica prospettiva, a nostro avviso, per sistemi reattivi di intrusion detection (o sistemi di intrusion prevention che dir si voglia) è all'interno dei sensori host-based. Esistono già soluzioni commerciali che provano a proporre questo tipo di concetto [35]. Un'altra soluzione che esula dal campo che stiamo studiando ma dalle interessanti applicazioni è Angel [36], un LKM (Loadable Kernel Module) sviluppato da due studenti italiani, che cerca di impedire agli utenti di una macchina di utilizzarla per lanciare attacchi, una sorta di intrusion prevention… in uscita.

In campo militare sono stati velatamente proposti anche IDS reattivi con potenzialità di contrattacco verso l'host aggressore, ma l'idea è stata saggiamente scartata, almeno ufficialmente, per le ovvie e incontrollabili implicazioni.

3.3.2 SICUREZZA INTRINSECA: SURVIVABILITY, ROBUSTEZZA

Un IDS può essere progettato per la "survivability": dal momento che (esattamente come una scatola nera) le sue registrazioni possono essere l'unico mezzo di stabilire cos'è accaduto su una rete, diventa importante garantire che il sistema stesso sia quanto meno vulnerabile possibile. La sovversione di un sistema di difesa è infatti il più pericoloso risultato di un attacco telematico, in quanto lascia agli attaccati un falso senso di sicurezza.

Deve inoltre essere in qualche modo garantita la sicurezza della comunicazione tra le varie componenti dell'IDS, pertanto opportuni meccanismi di autenticazione, crittografia, e verifica di integrità devono essere implementati per evitare ogni sorte di attacchi di spoofing o di tipo man-in-the-middle.

È inoltre opportuno che, per ogni tipo di attacco ed aggressione (distruzione di una sonda, flood di pacchetti per disturbare l'analizzatore di rete, eccetera) il sistema presenti robustezza, ovvero non si arresti completamente ma esibisca una graduale riduzione delle performance man mano che l'attacco procede e si aggrava.

3.3.3 INTERAZIONE CON IL PERSONALE E FLESSIBILITÀ

All'interno di un NOC (Network Operations Center) o di un SOC (Security Operations Center) spesso vi sono pochi tecnici (specie durante i turni notturni) magari non eccessivamente esperti, che devono sovrintendere al controllo di numerose reti remote, che conoscono solo a grandi linee. Oltre alle ovvie problematiche connesse ai falsi positivi, i sistemi di IDS devono essere progettati per comunicare quante più informazioni possibile al personale che deve controllarli, nel modo più chiaro possibile, e possibilmente fornendo anche una guida alla soluzione di un determinato problema.

In aggiunta, deve essere possibile una totale flessibilità e adattabilità del sistema a circostanze molto specifiche di ogni singola rete. Questo può essere garantito in modo molto automatico da IDS in modalità "anomaly detection", ma deve essere opportunamente studiato per sistemi basati su firme.

3.3.4 ADATTABILITÀ AD ATTACCHI NUOVI

I possibili tipi di attacco contro una rete informatica evolvono in continuazione. Si possono distinguere una microevoluzione, processo con cui vengono ideati nuovi attacchi all'interno di categorie di attacchi già esistenti, e una macroevoluzione più lenta mediante la quale vengono scoperti dei metodi di attacco completamente nuovi.

È abbastanza evidente che la macro evoluzione può creare forme d'attacco per cui il sensore non è architetturalmente predisposto. Per fare un esempio, se il sensore controlla il traffico di rete al layer 3 ISO/OSI, e viene sviluppata una metodologia di attacco che coinvolge manipolazioni al layer 2, è molto probabile che un semplice aggiornamento della base di conoscenza o del modello con cui l'IDS opera non sia sufficiente. Si rende necessaria una revisione del motore di analisi in sé. Viceversa, in linea di principio, se una forma di attacco è già nota, le sue successive microevoluzioni dovrebbero essere relativamente semplici da inserire nella base di conoscenze o nel modello dell'IDS.

Potrebbe sembrare che il problema di aggiornamento colpisca soprattutto i motori di misuse detection, che in effetti operano su una base di firme che tende a diventare

sempre meno adeguata con il tempo (esattamente come accade per il software antivirus, che è sicuramente un esempio più familiare). È tuttavia importante notare come anche i sistemi di anomaly detection possono soffrire di questo problema. Come verrà ampiamente discusso in seguito, il primo passo per costruire un sistema di questo genere è la scelta delle metriche e delle variabili da misurare, ed il secondo passo l'implementazione di un modello (certo, da tarare) che è fisso. La scarsa diffusione questi sistemi ha reso fino ad ora poco studiate le possibili tecniche di evasione, ma è certo possibile, per alcuni versi persino facile, immaginare delle forme di attacco che sfruttino gli "angoli morti" di queste soluzioni, ovvero tecniche di aggressione i cui unici sintomi ricadano in quelle variabili che sono state scartate come indicatori in fase di progettazione. È abbastanza evidente come in questo caso l' unica soluzione sarebbe quella di ristudiare il modello da capo.

3.3.5 SCALABILITÀ E CONCORRENZA

Un buon sistema IDS deve essere in grado di scalare seguendo la crescita della rete. Per questo le soluzioni network-based hanno spesso sofferto rispetto a soluzioni host-based, in quanto analizzare gli eventi in modo distribuito è sicuramente più semplice che cercare di capire qualcosa analizzando tutto il traffico della rete aggregato. Tuttavia va notato che a livello commerciale esistono prodotti ed appliance in grado di supportare volumi di traffico "da gigabit", e che esistono studi accademici relativi alle problematiche di implementazione di sistemi di misuse detection ad altissime prestazioni, quale ad esempio BRO [37].

Tipicamente, quando un IDS di rete supera la propria capacità di analizzare i pacchetti inizia a lavorare in una modalità "a campione", scartando parte del buffer con una tecnica simile a quella utilizzata dai router. Il problema è che, nel caso dei router, i meccanismi di ritrasmissione del TCP evitano la perdita di pacchetti, mentre nel caso dell'IDS, a meno di fortunate coincidenze, ciò che è perso è perso, e potrebbe trattarsi proprio dei pacchetti interessanti.

Se tale coincidenza pare troppo azzardata e teorica, proviamo a guardare la cosa dal punto di vista dell'attaccante: generare una marea di traffico di nessun interesse è relativamente semplice; se ciò significa mettere in crisi l'IDS ancora prima che l'attacco sia cominciato, si può essere sicuri che qualsiasi attaccante con un minimo di esperienza adotterà questa tecnica, nella speranza che ad essere scartati siano proprio i pacchetti con le firme d'attacco. Questo tipo di attacco viene spesso citato dai venditori di strumenti in-line come uno dei motivi per preferire la loro soluzione. Questo può essere vero, ma non va trascurata la possibilità che questa "feature" si trasformi in un comodo mezzo per perpetrare un attacco di tipo denial-of-service.

In aggiunta ai problemi di scalabilità, il sistema deve essere in grado di gestire attacchi multipli concorrenti. Molto raramente un intruso proverà un singolo attacco contro le nostre macchine: spesso utilizzerà uno strumento che prova attacchi multipli, e vogliamo essere in grado di seguirli tutti. Non solo: seguendo le stesse considerazioni fatte poco fa, se l'attaccante sapesse che il nostro IDS è in grado di seguire soltanto un attacco alla volta farebbe in modo di generare un finto attacco proveniente da chissà dove per "distrarre" il sistema.

3.4 PROBLEMI TIPICI DI UN IDS

3.4.1 INDIVIDUAZIONE DI ATTACCHI NUOVI O MODIFICATI

Ci siamo occupati della adattabilità di un IDS a forme di attacco nuove, presupponendo l'intervento umano nell'aggiornamento di una eventuale base di conoscenza di un sensore. Uno dei problemi fondamentali che abbiamo volutamente trascurato è la possibilità, del tutto realistica, che nuove forme di attacco colpiscano i sistemi *prima* che gli esperti abbiano occasione di analizzarle. Nella tassonomia degli attaccanti abbiamo infatti distinto coloro che si limitano ad utilizzare strumenti di attacco ben conosciuti e diffusi (gli *script kiddie*) da coloro che viceversa sono in grado di creare nuovi attacchi. Ovviamente, nel primo caso, è prevedibile che sia disponibile una "firma" per quel particolare tool d'attacco, mentre nel secondo caso non è così certo.

Pertanto, dobbiamo valutare se l'IDS sia o meno in grado di far fronte autonomamente all'evoluzione degli attacchi, o se dipenda totalmente dall'aggiornamento manuale della sua base di conoscenze.

Come premessa, possiamo presupporre che le nuove forme di attacco sottoposte a un IDS facciano parte della microevoluzione. Nuove macro categorie di attacchi sono rare e generalmente vengono scoperte e portate alla luce molto prima che strumenti di attacco effettivi ed efficaci vengano sviluppati nell'underground.

Ciò premesso, è abbastanza evidente che un IDS che dipende da una base di conoscenze sia molto meno resistente (da un punto di vista teorico, quanto meno) alla microevoluzione degli attacchi rispetto ad un sistema basato sull'individuazione di anomalie.

Esiste poi un problema sostanzialmente inaffrontabile, che è quello del *polimorfismo* degli attacchi. In molti casi è possibile raggiungere un determinato scopo seguendo una pluralità di metodi, ed è corrispondentemente più difficile sviluppare delle firme appropriate. Per fare due esempi classici, tutti i tipi di bug

connessi all'uso di caratteri "unicode" sono inerentemente polimorfi, in quanto esistono codifiche multiple per ogni singolo carattere: ciò significa una crescita combinatoria nel numero di firme necessarie ad intercettare tutte le possibili variazioni di questi attacchi.

Un secondo esempio, ben più inquietante, è dato dallo strumento ADMutate [38], sviluppato da "K2" (un hacker canadese, il nome reale è ignoto): si tratta di uno strumento in grado di criptare lo shellcode di un attacco di buffer overflow destinato allo stack-smashing. Come si è già accennato, questo tipo di bug consiste nella possibilità per l'aggressore di scrivere in un buffer limitato una stringa di lunghezza arbitraria, andando a sovrascrivere lo stack del programma ed, in pratica, assumendo il controllo del flusso di esecuzione. Una stringa di buffer overflow, in buona sostanza, consiste in una sequenza lunghissima di caratteri a cui segue una serie di istruzioni in codice macchina (il cosiddetto "shell code"). L'idea è che la stringa riempia il buffer, debordi, e che si arrivi a sovrascrivere il puntatore alla prossima istruzione presente sullo stack, facendolo puntare proprio allo shell code e consentendo così l'esecuzione di comandi arbitrari.

Siccome molto spesso lo shell code contiene delle porzioni di codice "comuni", vi sono firme "generiche" sugli IDS misuse-based che si propongono di intercettarle. ADMutate cripta lo shellcode, anteponendo ad esso la routine di decriptazione. In tal modo è possibile ad ogni esecuzione dell'attacco variare la forma (ma non l'efficacia) del codice macchina utilizzato, mandando in confusione un discreto numero di sistemi IDS network-based commerciali.

Anche se ora praticamente tutti gli IDS contengono una apposita firma per riconoscere ADMutate, ciò non toglie che lo stesso principio possa essere in linea teorica ripetuto all'infinito. Il polimorfismo è un avversario strutturale di qualsiasi sistema IDS basato sulla misuse detection, in quanto mina alla base l'assunto che sia possibile elencare le forme d'attacco.

Molto spesso la risposta dei sostenitori della misuse detection a questa obiezione è che si possono trovare firme più intelligenti per gli attacchi. Si tratta però di un eterno gioco di scacchi in cui ad ogni mossa corrisponde una contromossa uguale e contraria. Gli scacchisti dovrebbero sapere che i "mirror game" in cui ad ogni mossa si risponde con la mossa speculare sono destinati alla vittoria inevitabile di chi ha mosso per primo.

Fuor di metafora, è inutile continuare in questa direzione. Per dare un altro esempio concreto, utilizzando di nuovo gli attacchi di tipo "buffer overflow", molti IDS hanno anche una firma che riconosce lunghe sequenze di 0x90, che è il codice

esadecimale dell'operazione NOP su processori Intel X86. Questo codice viene tipicamente usato nei buffer overflow come "riempitivo" in quanto è spesso difficile determinare il punto esatto dal quale tale codice verrà eseguito. Utilizzando delle NOP, il processore le interpreterà tranquillamente passando all'operazione successiva. Per evitare di essere riconosciuto in questo modo, un attaccante più furbo può per esempio utilizzare una istruzione JUMP all'indirizzo successivo (0xeb 0x00) al posto della NOP, con il problema aggiunto di utilizzare un codice a 2 byte (è quindi possibile che l'esecuzione inizi "a metà" di un'istruzione fallendo, ma in tal caso si rimedia facilmente spostando di uno l'offset del codice). Se anche questa firma viene aggiunta all'IDS, si può iniziare a giocare con i salti, in un'eterna rincorsa che i "buoni" sono destinati a perdere.

3.4.2 ERRORI: FALSI POSITIVI, FALSI NEGATIVI

Gli errori di un IDS vengono tipicamente suddivisi in due categorie. Si parla di "falso positivo" quando l'IDS segnala come anomala un'azione che è legittima o innocua. Si parla di "falso negativo" quando viceversa un IDS non suona l'allarme in presenza di una azione evidentemente maliziosa.

I "falsi positivi" possono suonare innocui: in fondo, meglio essere avvertiti una volta di troppo che una volta di meno. La famosa favola del ragazzino che gridava "Al lupo!" ci ricorda che questo non è sempre vero: un IDS che segnala in continuazione come "pericolose" azioni perfettamente normali dopo un po' verrà ignorato. Eliminare i falsi positivi è dunque uno dei focus della ricerca sugli IDS, in particolare per sistemi di anomaly detection, in quanto a meno di errori nelle firme è molto difficile che un sistema di misuse detection fornisca dei falsi positivi.

Va infatti notato che spesso quelli che gli utenti finali, in particolare di sistemi di misuse detection, indicano come "falsi positivi" sono in realtà le indicazioni di attacchi, veri e reali, contro piattaforme software o hardware che essi non hanno. Ciò che l'IDS sta segnalando, in quel caso, non è un falso positivo, è un verissimo attacco che qualcuno, alla cieca, ha rivolto contro le macchine sbagliate. Sta all'interesse dell'amministratore in tal caso scegliere se essere avvertito di tali attacchi (che, per quanto innocui, non sono comunque un buon sintomo) o ignorarli direttamente mediante il tuning del sistema.

I falsi negativi sono invece un dramma aperto: si tratta del mancato riconoscimento di una forma d'attacco. In generale, in un sistema basato sulla misuse detection, un falso negativo è associato con la mancanza di una firma per l'attacco effettuato (e ricadiamo quindi nel problema dell'individuazione di nuovi attacchi segnalato al paragrafo precedente). In un sistema basato sull'individuazione di

anomalie invece è più probabile che si abbia un falso negativo magari anche su un attacco che in precedenza era stato riconosciuto, proprio a causa della natura statistica di questo tipo di sistemi.

3.4.3 IL FATTORE TEMPO: SEQUENZIALITÀ E CORRELAZIONE

Il fattore tempo presenta le più grosse incognite per un progettista di IDS. È abbastanza evidente, ad esempio, il problema di ricostruire la sequenza tra eventi di qualsiasi tipo che siano stati rilevati da sensori dispersi su una grande rete. Il tempo di propagazione dei messaggi e la natura "best effort" delle reti TCP rende impossibile basarsi esclusivamente sull'ordine di arrivo di questi a un server centrale, sempre ammesso che un server centrale esista. Sono ben noti i problemi che si trovano nel cercare di dotare di un tempo unico un gran numero di macchine in rete (anche utilizzando protocolli ben studiati come NTP). Tuttavia questa è solo la punta dell'iceberg dei problemi che il fattore tempo impone al progettista di un IDS.

Innanzitutto, non è detto che un attacco sia costituito da una azione atomica: in realtà moltissimi attacchi sono costituiti da più eventi indipendenti, ciascuno dei quali preso singolarmente è innocente o quantomeno innocuo. Non è nemmeno detto che questi eventi debbano avvenire entro un determinato intorno di tempo l'uno dall'altro, o che non possano essere inframmezzati da un numero arbitrario di altri eventi assolutamente indipendenti (problema dell'interleaving).

Non solo: è ampiamente possibile che tutto questo si aggiungano problemi di unificazione, ovvero la presenza all'interno di una sequenza di attacco di un elemento variabile che può assumere una infinità di valori. Per finire, non è assolutamente detto che tali sequenze siano strettamente ordinate, anzi, molto spesso si tratta di un ordinamento parziale.

Tutto ciò avviene usualmente riassunto nel concetto di "uncertain reasoning", ovvero l'incapacità di determinare algebricamente se è possibile "chiudere" l'analisi o se è necessario rimanere in attesa del completamento di un attacco. Per esemplificare, se sappiamo che un attacco è formato dalle fasi A, seguita da B e da C in un ordine qualsiasi, e conclusa da D, con qualsiasi intermissione di eventi, l'osservazione della sequenza A-x-x-B-x-x-x non ci dice nulla sul fatto che stiamo osservando un attacco parziale (di cui dobbiamo aspettare la conclusione) oppure una sequenza assolutamente innocente di comandi (sempre nell'ipotesi che A e B non siano di per sé delle violazioni). Il sistema deve lasciare questa sequenza candidata "appesa", attendendo che avvenga un evento che dimostra che essa non potrà mai soddisfare la regola, o viceversa attendendo che essa sia completata.

In un sistema realistico, questo significa che prima o poi dovrà esserci un timeout o qualche altro meccanismo che consideri scadute osservazioni molto vecchie per fare spazio a quelle nuove, altrimenti offriremmo a un attaccante molto intelligente un metodo per disabilitare l'IDS intasando la memoria con sequenze appese. Anche il timeout offre una scappatoia all'attaccante: attendere oppure provocare il "flush" della sua sequenza d'attacco prima di completarla impunemente.

Ovviamente, tutti questi problemi sono tipici dei sistemi on-line, che soffrono contemporaneamente di "incertezza del futuro" e di una finestra limitata del passato, per ragioni ovvie di performance. In questo caso sono enormemente avvantaggiati i sistemi off-line, che possono gestirsi una finestra illimitata nel passato, e che conoscono già l'intero arco temporale "futuro".

3.4.4 RICOSTRUZIONE DEL TRAFFICO E AMBIGUITÀ

Un sistema IDS network-based deve affrontare un ulteriore insieme di problemi derivanti strettamente dalla sua fonte di informazione, ovvero il traffico di rete. Per discutere di queste problematiche facciamo riferimento alle reti basate su TCP/IP, ma il discorso è molto generale e si estende ad altri tipi di rete.

In linea teorica, osservando tutto il traffico di rete dovrebbe essere possibile, per un sistema arbitrariamente potente, tenendo conto della topologia della rete e di tutte le particolarità delle macchine che la compongono, stabilire esattamente quale sia l'effetto di ogni pacchetto IP che viaggia sulla rete. Tuttavia questa visione è inguaribilmente ottimistica: è computazionalmente proibitivo pensare di ricostruire su una singola macchina gli esatti effetti che il traffico di rete produce su ciascuna delle altre macchine connesse; inevitabilmente, un IDS network-based deve cercare altri tipi di approcci per determinare cosa sia un attacco e cosa no.

Alcuni IDS per esempio si limitano a fare un pattern matching pacchetto per pacchetto, e questa è senza dubbio una soluzione banale che perde gran parte della relazione temporale e di connessione tra i dati; altri, più evoluti, offrono una analisi di tipo "stateful", ovvero ottengono traccia delle connessioni stabilite e analizzano i pacchetti basandosi su queste informazioni.

Tuttavia, questa soluzione non è certo la panacea di tutti i mali: come è stato dimostrato da numerosi ricercatori [39] può essere molto difficile se non impossibile stabilire in un punto della rete esattamente quali dei pacchetti visti raggiungeranno un altro punto. Su questo principio si basano i due attacchi speculari di inserzione ed evasione dei pacchetti. Il primo tipo di attacco sfrutta quest'idea: se l'IDS è tratto per

dare l'allarme quando osserva, per esempio, la stringa “ATTACCO”, una delle possibili soluzioni è quella di spezzettare questa stringa di modo che all'IDS arrivi qualcosa che per esempio si legge “ATTrACCO”, facendo in modo che il pacchetto contenente la lettera r arrivi soltanto all'IDS e non al sistema attaccato. Questo si può ottenere, ovviamente in casi particolari, giocando con le flag del TCP. Per esempio, è molto difficile che gli IDS di rete perdano tempo a controllare i checksum dei pacchetti, e un attaccante potrebbe prendere vantaggio di questo. Altre ambiguità su cui si potrebbe giocare solo il campo TTL (che potrebbe essere grande a sufficienza per raggiungere l'IDS ma non il sistema bersaglio), utilizzare un pacchetto troppo grande con attivato il flag “don’t fragment”, sfruttare campi di opzione o timestamp strani o malformati per sfruttare le ambiguità che esistono nelle implementazione degli stack TCP su vari sistemi operativi. L’esatto contrario darebbe origine ad attacchi di “evasion”, in cui invece di vedere “ATTACCO”, per esempio, l’IDS vede solo “TACCO”. I meccanismi che possono essere sfruttati per ottenere questo tipo di ambiguità sono gli stessi considerati prima, con un curioso e perverso legame: più si cerca di rendere un sensore di rete resistente all’evasione, più si accettano pacchetti “anomali”, dando spazio ad attacchi di inserzione, e viceversa.

Un’ulteriore problematica deriva dalla possibilità che gli attacchi vengano dispersi in più pacchetti mediante l'uso della frammentazione. Ormai tutti i principali IDS di rete hanno affrontato questo problema, ma la ricostruzione di pacchetti frammentati si è rivelata un problema più difficile del previsto, oltre a richiedere un dispendio di risorse poco compatibile con le moderne esigenze di performance. Infatti si sono notate numerose incongruità nel modo in cui i vari stack TCP/IP affrontano i problemi di frammenti non perfettamente rispondenti alle specifiche.

È insomma impensabile cercare di indovinare tutte le possibili interpretazioni del flusso di pacchetti basandosi su informazioni variabili quali la topologia della rete e tipi di sistemi in uso.

3.5 SISTEMI ED ARCHITETTURE RILEVANTI

In questo paragrafo si vuole proporre una breve carrellata di sistemi di Intrusion Detection che hanno goduto di una certa fortuna, o che hanno proposto idee interessanti ed innovative. Non vuole essere un elenco in nessun modo completo (suggeriamo per questo di consultare apposite risorse in rete [40], costantemente aggiornate), solo un tentativo di chiarire meglio quanti differenti approcci si possano provare per un problema tanto complesso. Se pare curiosa l’assenza di pacchetti software commerciali nell’elenco, essa è dovuta al fatto che nessuno dei pacchetti

oggi disponibili dimostra l'innovatività che caratterizza le architetture da noi elencate.

3.5.1 IDES, NIDES, EMERALD

IDES (Intrusion Detection Expert System, [41]), sviluppato da SRI, è uno dei sistemi di Intrusion Detection host-based "accademici" più ampiamente citati, per il suo approccio molto razionale ed efficace. IDES si compone di un modulo di ragionamento costituito da un sistema esperto realizzato in P-BEST. Tramite questo sistema vengono modellate le regole che segnalano un'intrusione, secondo un approccio di "misuse detection". Ma nella fact list del sistema esperto non affluiscono solo i rilevamenti degli "eventi" di rete: affluiscono anche gli eventi segnalati dal sottosistema STAT.

STAT analizza in modo statistico la audit trail del sistema, rendendo IDES uno dei primi sistemi a cercare di integrare anomaly detection e misuse detection. STAT, di per sé, non è particolarmente illuminante: rileva le anomalie basandosi sulla misurazione di tredici "variabili d'intrusione", di cui calcola la covarianza. Un'alta covarianza implica una buona probabilità che qualcosa non vada per il verso giusto. Come si può capire, il problema di base è che STAT non propone un modello su cui fare rilevamenti di anomalia, ma cerca di definire a priori delle metriche. È un errore molto comune.

In seguito, IDES è stato sviluppato e migliorato per dare vita a NIDES e ad EMERALD, entrambi progetti degni di nota, raggiungibili dal sito di SRI.

3.5.2 ISOA: INFORMATION SECURITY OFFICER ASSISTANT

Prototipo del 1989, ISOA [42] fu progettato dalla Planning Research Corporation per analizzare dati provenienti da SunOS e da Xenix su una workstation di monitoraggio. La cosa interessante è che questo sistema, oltre ad essere uno dei primi pensati in maniera distribuita con una console centrale di analisi e correlazione (qualcosa che ancora al giorno d'oggi viene percepito come un bisogno insoddisfatto) è uno dei pochi ad esibire una miscela di comportamento on-line (con dei trigger che scattano in presenza di eventi inequivocabilmente pericolosi) e off-line (consentendo all'amministratore di richiedere un'analisi più approfondita degli eventi di un determinato periodo di tempo).

3.5.3 MIDAS E WISDOM&SENSE

Questi due sistemi sono interessanti per un motivo particolare. In genere, gran parte dei sistemi di intrusion detection basati su misuse detection usano sistemi a regole o sistemi esperti di qualche tipo come loro motore di correlazione.

MIDAS (Multics Intrusion Detection Alerting System [43]), un sistema on line realizzato dal National Computer Security Center, è un primo controesempio. È basato su un sistema di regole scritte in LISP, strutturate su quattro livelli in forward chain. Le azioni "anomale" vengono individuate e riassunte nella base dei fatti da apposite regole.

Wisdom&Sense [44], invece, è ancora più interessante. Sviluppato al Los Alamos National Lab, ricava la sua base di regole dallo "storico" delle operazioni normali del sistema. La struttura delle regole è molto particolare, si parla di numerosissimi (10000 o più) frammenti, non più lunghi di 10 byte, tra cui la ricerca dura 50 millisecondi, e la generazione meno di un'ora (e i dati sono riferiti alla potenza computazionale del 1994). L'algoritmo usa delle classiche tecniche di espansione e pruning, e probabilmente potrebbe essere migliorato e potenziato dall'uso di tecniche moderne come IREP.

3.5.4 DIDS: DISTRIBUTED IDS

DIDS, Distributed Intrusion Detection System, è stato sviluppato come estensione di un sistema precedente, noto come NSM, Network Security Monitor, ed è uno dei primi esempi compiuti di IDS distribuiti, con una console centrale, e "agenti" di tipi diversi.

NSM era un sistema anomaly-based e network-based, che categorizzava ogni connessione assegnandole un fattore di rischio, sulla base di vettori multipli di parametri relativi alla connessione (durata, rete di provenienza, host e porta di destinazione…).

In DIDS vennero aggiunti dei sensori distribuiti sugli host per intercettare anche intrusi che non accedessero via rete. A raccogliere e integrare questi dati venne posto un modulo denominato DIDS Director. Una delle caratteristiche più sorprendenti di questa architettura è tuttavia il modulo denominato NID (Network-user ID), che tenta di tenere traccia dell'utente mano a mano che si muove tra account multipli sulla rete.

È al contempo un primo segno che il sistema di autenticazione basato su user e password su ogni host non scala alle esigenze di controllo e monitoraggio di una grande rete, e una novità nel campo della correlazione, di cui ancora oggi si percepisce la mancanza.

Il DIDS Director è meno innovativo, utilizza una classica struttura knowledge based realizzata con CLIPS, e un sistema a regole gerarchiche denominato Intrusion Detection Model (IDM), che descrive la trasformazione dei dati in ipotesi d'alto livello sui tentativi d'intrusione. Le classi della gerarchia sono molto interessanti, perché sono poi gli stessi livelli a cui si incontrano difficoltà nei moderni sistemi di IDS distribuito:

1. Dati specifici di piattaforma
2. Descrizione platform-independant di eventi
3. Intervento del NID a correlare gli eventi
4. Inserimento di spazializzazione (provenienza) e temporalizzazione (assoluta e relativa)
5. Categorizzazione in attacks, misuses, e suspicious acts
6. Produzione di un livello overall di "pericolo"

3.5.5 SNORT

Snort [45] è un IDS network-based, basato sostanzialmente sulla misuse detection. Lo citiamo come esempio di questa categoria di sistemi, in quanto è un progetto sviluppato sotto GPL (GNU public license), distribuito gratuitamente su Internet in formato open source e aperto ai contributi della comunità, l'ideale come riferimento per un lavoro scientifico. Inoltre, Snort è uno dei progetti open source più famosi del mondo, e uno dei pochissimi esempi di IDS non commerciali che godano di un supporto e di una base installata tali da renderlo un "concorrente" di fascia bassa di molti sistemi commerciali.

Snort è un sistema multipiattaforma che utilizza le librerie di cattura pacchetti libpcap (le librerie standard "de facto" per la cattura di pacchetti di rete, di cui riparleremo nel capitolo quarto di questa tesi). L'architettura di Snort comprende tre sottosistemi primari: un packet sniffer/decoder altamente performante; un motore a regole basato sul pattern matching; e un sottosistema per il logging, la creazione di report, e l'invio di avvisi all'amministratore di rete.

Concentriamoci sul componente di analisi. Il linguaggio con cui vengono espresse le regole di Snort è molto semplice ed efficace. Ecco un esempio di regola:

```
log tcp any any -> 10.0.0.0/24 79
```

Questa regola scatta quando viene rilevato del traffico TCP, da qualsiasi host, qualsiasi porta sorgente (tcp any any), verso un host della classe C 10.0.0.*, porta 79 (ma possiamo anche scegliere un range di porte, ad esempio 1:1024), e richiede a snort di effettuare il logging del pacchetto. Altre azioni possibili sono pass (ignora) e alert (avvisa l'amministratore usando il metodo da lui prescelto).

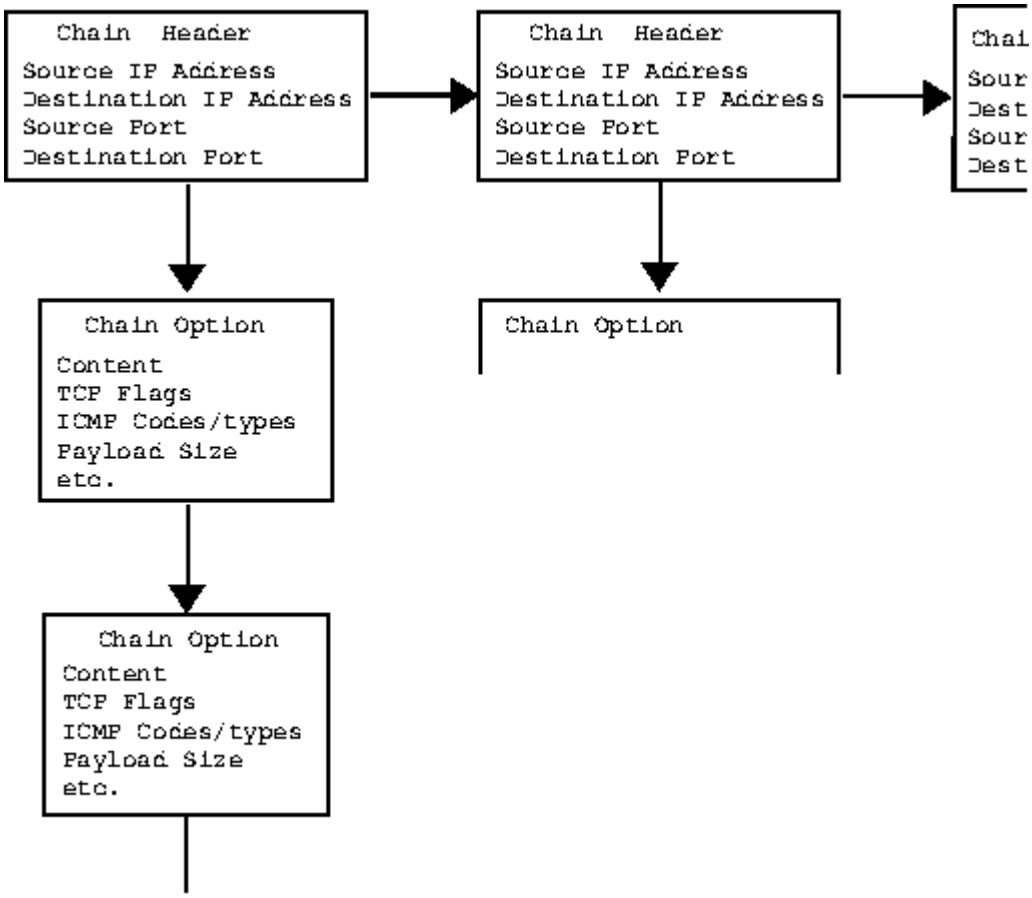
Ovviamente, specificare solo provenienza e destinazione sarà raramente utile. Per specificare il contenuto e la tipologia dei pacchetti per cui far scattare l'allarme dovremo inserire una porzione aggiuntiva della regola, come nell'esempio che segue:

```
alert tcp any any -> 10.0.0.0/24 80 (content: "/cgi-bin/phf"; msg:
"Probe del PHF!";)
```

Questa regola va ad individuare nel traffico diretto alla porta 80 delle nostre macchine il probe per la vecchissima vulnerabilità del PHF, ormai corretta da anni, ma che viene usata per tradizione negli esempi. Se volessimo individuare del traffico in formato binario potremmo per esempio scrivere una regola con l'operatore || di questo tipo:

```
alert tcp any any -> 10.10.10.0/24 111 (content: "|00 01 86 a5|";
msg: "Accesso al mountd";)
```

Nelle regole si possono anche specificare altre condizioni (minfrag, per la frammentazione dei pacchetti; ttl, id, dsize per i campi omonimi; le varie flag del TCP, eccetera). Snort conserva le sue regole di detection in una lista linkata bidimensionale per applicare un algoritmo di pattern matching estremamente efficiente, di cui spieghiamo il funzionamento aiutandoci con una figura. Riprendendo il concetto di "catena" utilizzato nel packet filtering del kernel di linux, i pacchetti vengono fatti "combaciare" dapprima in orizzontale sulla base dell'header, ed in seguito in verticale per le opzioni di payload solo in caso di match degli header, in modo da ottimizzare il più possibile il tempo di analisi. Basandosi su questo approccio, alcuni autori [46] hanno proposto degli strumenti di clustering automatizzato per sviluppare il più efficiente albero di decisione possibile a partire da una base di regole di Snort.



La struttura della doppia lista linkata di Snort

4

UN IDS BASATO SULL'APPRENDIMENTO NON SUPERVISIONATO

4.1 OBIETTIVO DEL PROGETTO

Ci poniamo l'obiettivo di studiare la realizzabilità di un sistema di intrusion detection di tipo network based con un approccio di anomaly detection. Questa scelta è motivata da svariate ragioni:

- l'approccio "misuse detection" è stato studiato approfonditamente ed implementato in molti modi, con risultati del tutto apprezzabili. Tuttavia, il paradigma su cui si basa sta sempre più mostrando i propri limiti, ed è nostra opinione che nel futuro prossimo la ricerca si orienterà sempre di più verso sistemi complementari di individuazione dell'anomalia
- Sistemi di anomaly detection molto evoluti di tipo host based sono già stati studiati in modo estensivo, mentre sono stati in qualche modo trascurati i sistemi di anomaly detection network based per una serie di problemi che esporremo.

Abbiamo cercato di affrontare il problema non con un tradizionale approccio statistico, ma cercando di analizzarlo nei termini tipici dei sistemi di apprendimento non supervisionato. I nostri obiettivi sono:

- Cercare di proporre una architettura realizzabile, computazionalmente efficiente, e strutturata pensando ad obiettivi realistici
- Cercare di dimostrare e di verificare, passo per passo, le nostre intuizioni ed asserzioni. Laddove stiamo proponendo delle ipotesi non ancora verificate avremo cura di sottolinearlo, delineando se possibile ulteriore progetto di ricerca
- Utilizzare, sempre, dati realistici e prototipi funzionanti per avere la certezza di parlare di risultati concreti

4.2 RICERCHE PRECEDENTI

Come scelta di base, abbiamo ritenuto opportuno per quanto possibile citare gran parte della letteratura esistente direttamente nelle discussioni, senza concentrare le citazioni in un unico paragrafo, ritenendo che questo approccio meglio si adattasse a paragonare passo passo le nostre scelte ed ipotesi con quelle di altri ricercatori. Tuttavia riassumiamo qui alcune ricerche che ci hanno colpito ed influenzato, anche se affrontano il problema da approcci completamente differenti.

Esistono numerose ricerche nel campo dell'intrusion detection anomaly-based, ma molte di esse sono rivolte all'analisi di dati rilevati sull'host, per esempio il sistema IDES cerca di determinare comportamenti che deviano dalla norma esaminando parametri come l'utilizzo della CPU e la frequenza di vari comandi. È da notare che IDES utilizza questo approccio come "complemento" ad un più tradizionale sistema esperto.

Ricerche più complesse ed interessanti si basano invece sulla modellazione del comportamento dell'utente in shell di un sistema UNIX [47], basandosi su Instance Based Learning (IBL) ed altre tecniche. Questo ed altri lavori (ivi comprese alcune ricerche già svolte dall'autore e riportate in appendice al presente lavoro) si basano sostanzialmente sul concetto dell'identificazione di un modello di comportamento "normale" partendo dalla disponibilità di un insieme di dati di addestramento "normali".

Un altro approccio originale ed interessante inverte il discorso, studiando l'uso di comandi "rari", che sono zeri strutturali nelle matrici di incidenza comando/utente [48]. Sono stati proposti anche interessanti algoritmi a base neurale (NNID, Neural Network Intrusion Detection [49]), sempre concentrandosi sull'analisi delle sessioni di lavoro degli utenti e sul raffronto con un database storico. L'utilizzo di un sistema basato su algoritmi genetici è stato proposto [50], ma è difficile pensare a una implementazione pratica.

Altri autori [51] hanno già provato ad affrontare il problema in termini di data mining, cercando di spostarsi progressivamente da un approccio interattivo di "estrazione di regole" con l'intervento umano verso un approccio il più automatizzato possibile. Il loro approccio è efficace solo in termini molto locali e ridotti, e necessiterà di molti raffinamenti prima di poter essere proposto come prototipo. Tuttavia, ci ha rincuorato la scelta degli autori di dedicare attenzione anche all'uso dei dati provenienti dalla rete (mediante l'uso di tcpdump).

L'applicazione di algoritmi a base neurale al traffico di rete è stata proposta e studiata molte volte, ma raramente in maniera estensiva. Soltanto in una ricerca [52] si dimostrano risultati a nostro avviso interessanti e specifici, ed infatti alcune delle idee che abbiamo esplorato sono state ispirate da questo lavoro. Una interessantissima prospettiva basata sulle analogie tra sistemi immunitari biologici e difese di rete è stata proposta [53] da alcuni autori.

L'applicazione di algoritmi di anomaly detection di tipo statistico al traffico di rete, con scopi limitati quali ad esempio il rilevamento dei cosiddetti portscan, ovvero la scansione di tutte le porte TCP di una macchina alla ricerca di servizi in ascolto, è invece già una realtà anche in sensori di intrusion detection non accademici. Uno dei migliori esempi, disponibile con il codice sorgente in quanto si tratta di un prodotto open source, è SPADE (Statistical Packet Anomaly Detection Engine), un plug-in per Snort, studiato per rilevare pacchetti "anomali" e correlarli, nella speranza di individuare portscan effettuati con tecniche particolari di occultamento.

Anche uno degli autori del tool di monitoraggio di rete NTOP ha recentemente proposto [54] una architettura di anomaly detection di base di tipo statistico basato su tale strumento. Tuttavia, tale architettura è stata "scorporata" dal progetto principale per motivi di performance (le regole rendevano problematico l'utilizzo di NTOP in reti di velocità superiore ai 100Mb/s), lasciando però a disposizione tutti i contatori che possono essere utilizzati per tale scopo, e anzi studiando estensivamente quali valori possano fornire una indicazione empirica di "attività anomala", in modo da implementare tali controlli nelle future versioni di NTOP [55].

4.3 FORMULAZIONE DEL PROBLEMA

Vogliamo costruire un sistema di IDS network based, per reti basate su protocollo TCP/IP, basato su algoritmi non supervisionati di apprendimento. Il dominio che vogliamo analizzare è dunque il flusso dei pacchetti IP (e allo strato superiore ISO/OSI i protocolli TCP, UDP, ICMP principalmente) attraverso una rete nel tempo. Ora, si possono fare alcune osservazioni:

- Un pacchetto ha dimensioni che variano da circa 20 a 1500 byte (IP over Ethernet).
- I byte dell'header IP sono solitamente 20, e hanno una struttura ben nota a priori, da essi si può estrarre un gran numero di informazioni su cui abbiamo conoscenze complete di dominio; non abbiamo tuttavia (e non possiamo

nemmeno avere) conoscenze complete a priori sul "senso" che questi dati assumono all'interno del flusso.

- Il payload non ha, in generale, una struttura nota a priori, in quanto l'Internet Protocol può trasportare svariati protocolli di livello superiore, che a loro volta sono protocolli multifunzione. Protocollo per protocollo, e salendo ancora di livello, ovviamente la struttura si definisce, ma ogni filtro aggiunto ha un costo prestazionale non irrilevante. Inoltre, come abbiamo già indicato, diventa progressivamente più difficile per un singolo IDS stabilire gli effetti esatti di queste comunicazioni sui molteplici host di una rete mano a mano che si sale di livello. Dovremo dunque adottare un criterio di semplificazione, cercando di trattare i dati ad un livello simile a quello adottato da altri sistemi di IDS di rete esistenti.
- I pacchetti hanno raramente senso da soli, ma vanno correlati tra loro. In poche parole l'IDS deve essere in grado di conservare una memoria (simile all'analisi stateful), oppure di osservare una finestra temporale di dati, la più ampia possibile.

Purtroppo, semplici considerazioni di natura computazionale (contro cui ci siamo scontrati duramente anche durante l'implementazione proof-of-concept degli algoritmi di cui al capitolo 5) dimostrano che "poche migliaia" di byte sono il limite realistico di dimensione per un input su cui siano utilizzabili algoritmi di apprendimento non supervisionato. Pertanto, difficilmente potremmo mostrare a tale algoritmo più di un pacchetto alla volta, men che meno una "finestra" di pacchetti.

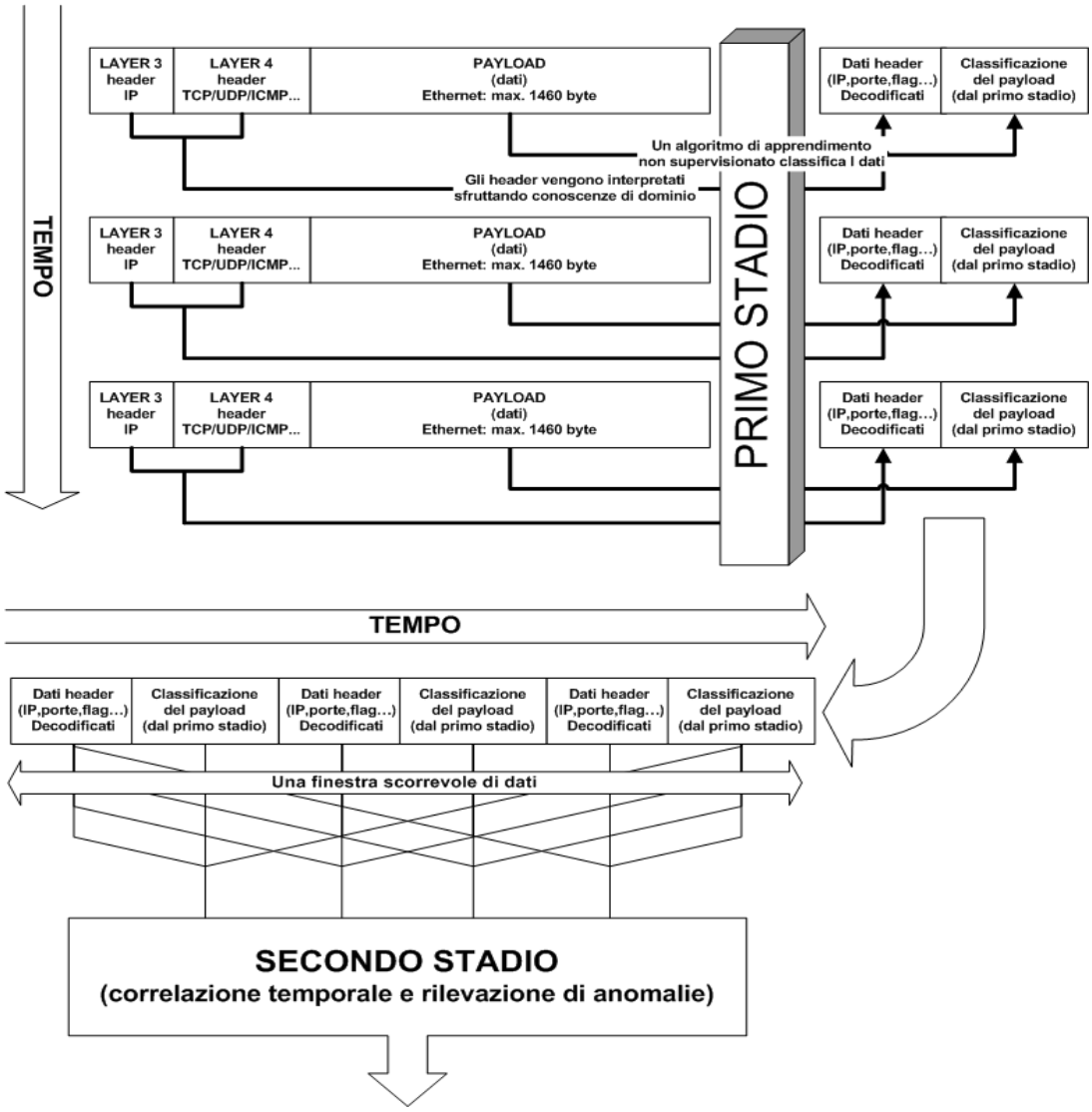
Questo deriva dalla struttura intrinseca degli algoritmi non supervisionati, che non possono affrontare informazioni di "qualsiasi" dimensione per una molteplicità di ragioni che chiariremo estensivamente nel capitolo 5. Nei rari casi in cui lo fanno, in generale si tratta di informazioni "sparse" (per esempio, matrici di incidenza parola-documento), per cui esistono degli appositi algoritmi di trattamento algebrico.

Molte ricerche esistenti (ne parleremo diffusamente nel capitolo 6) risolvono in maniera brutale il problema, "scartando" a priori il payload come intrattabile; nonostante questo trattamento gli algoritmi di apprendimento utilizzati esibiscono interessanti capacità di individuare traffico anomalo (seppure, chiaramente, in condizioni sperimentali molto controllate). Essendo convinti che questa strada conduca ad un inaccettabile perdita di informazioni fondamentali, ci siamo concentrati sul problema di come recuperare almeno in parte tali informazioni.

4.4 ARCHITETTURA PROPOSTA

4.4.1 ARCHITETTURA DI MASSIMA

Per risolvere il problema conservando il maggior numero possibile di informazioni, proponiamo di utilizzare una struttura costituita da due stadi connessi in cascata: un primo stadio che si occupi di "compattare" le informazioni contenute nel payload di ogni singolo pacchetto TCP o UDP, di modo da poter "mostrare" quanti più pacchetti possibile in contemporanea al secondo stadio, che si occuperà invece di mantenere una correlazione temporale e una memoria. Esemplifichiamo tale architettura in uno schema grafico.



Schema dell'architettura di intrusion detection da noi proposta

In sostanza, al primo stadio richiederemo di comprimere, o meglio di classificare, in un numero arbitrario di classi i payload associati al pacchetto, mentre il secondo stadio effettuerebbe l'anomaly detection vera e propria, osservando in parallelo una finestra di pacchetti sequenziali e cercando correlazioni e stranezze. Analizziamo con qualche dettaglio in più ciò che richiediamo ai due stadi dell'architettura.

4.4.2 IL PRIMO STADIO

Per il primo stadio il requisito è: trovare un algoritmo che riceva in ingresso i (massimo) 1460 byte del payload di TCP o UDP, e li classifichi in modo "sensato" dando origine a pochi numeri, in linea di principio uno solo. Si noti che per ICMP il problema non sussiste, visto che la decodifica del pacchetto dà di per sé origine a tutte le informazioni che ci servono su di esso (alcuni pacchetti ICMP hanno un payload opzionale, che in genere non è interessante; tuttavia, sono noti alcuni utilizzi malevoli di questo come covert channel per comunicazioni nascoste).

Per "classificazione sensata" intendiamo che essa debba, per quanto possibile, conservare per il secondo stadio le informazioni principali sulla "similarità" dei pacchetti. Intuitivamente, considerando che la nostra finalità è l'individuazione delle intrusioni, questa classificazione dovrebbe quanto meno esibire la proprietà di distinguere il più possibile pacchetti con payload "anomalo" o malformato da pacchetti più normali, oltre a suddividere tali payload per "ampie classi" corrispondenti grossolanamente ai tipi di traffico presente sulla rete.

Nel prossimo capitolo 5 illustreremo in dettaglio le tecniche con cui è possibile affrontare questo problema, esploreremo la letteratura in proposito e gli algoritmi tra cui scegliere, e proporremo delle valutazioni iniziali ed una implementazione proof-of-concept di tali algoritmi, con considerazioni di tipo qualitativo e computazionale.

Un altro compito (relativamente banale) del primo stadio sarebbe decodificare tutte le informazioni contenute negli header di layer 3 e 4. L'esempio che segue mostra (per un caso reale, decodificato da uno dei prototipi che descriveremo nel capitolo successivo) le informazioni che il primo stadio dovrebbe passare al secondo stadio, per un pacchetto TCP scelto casualmente.

```
ip_hdlen: 20
ip_tos: 16
total_len: 46
UID: 774
DF: 1
MF: 2
frag_offset: 0
TTL: 64
PROTO: 6
CHKS: 24174
SRC_ADD: 3.2553e+009
DST_ADD: 2.8868e+009
ip_options: 0
SRC_PRT: 1028
DST_PRT: 21
```

```
SEQn: 1.9607e+009
ACKn: 914812458
TCP_HLEN: 20
URG: 0
ACK: 1
PSH: 1
RST: 0
SYN: 0
FIN: 0
WND: 32120
checksum: 26509
urgent_ptr: 0
tcp_options: 0
payload_class: 80
(eventuali altri dati)
```

4.4.3 IL SECONDO STADIO

Collegato in cascata al primo stadio, dovremo disporre un secondo algoritmo non supervisionato, che dovrà prendere in considerazione i pacchetti ed identificare due tipi di correlazioni:

1. Correlazione sui *singoli pacchetti*, analizzando il contenuto, le flag, e i campi, che gli consenta di riconoscere situazioni tipo "questo pacchetto ha il contenuto sbagliato relativamente alla porta a cui è indirizzato".
2. Correlazione *temporale* e *statistica* che gli consenta di riconoscere eventi come "un aumento strano del flusso di richieste verso questa particolare porta", oppure "una distribuzione anomala del flusso di pacchetti rispetto al normale", o ancora "una serie di pacchetti destinati a porte diverse sullo stesso host, provenienti dalla stessa porta dello stesso host remoto".

A questo algoritmo vogliamo presentare i dati in arrivo uno alla volta (supponendo che esso sia in grado di preservare una memoria di ciò che vede), o più significativamente una finestra di dati.

Che tipo di informazione può realisticamente produrre uno stadio del genere ? Probabilmente un output "sfumato" di tipo logico, che oscilli tra una perfetta normalità e una completa discrepanza rispetto al consueto. Meglio ancora sarebbe se il secondo stadio andasse oltre, e riuscisse ad indicare all'operatore umano quali siano i valori che meno collimano, ovvero i pacchetti "strani" e i motivi, i dati che li

fanno ritenere strani. Un secondo stadio ancora più efficace sarebbe in grado di utilizzare gli input e la revisione dell'operatore per apprendere ed aumentare così la propria efficienza.

Nel capitolo 6 affronteremo in dettaglio i problemi posti dal secondo stadio, e cercheremo di inquadrare teoricamente i diversi tipi di approcci disponibili; citeremo esperimenti tratti dalla letteratura per mostrare come sia già stata sperimentata la realizzabilità teorica di alcuni sistemi di intrusion detection anomaly based sostanzialmente simili al nostro secondo stadio concettuale, e proporremo considerazioni ed esperimenti per indicare come il primo stadio da noi proposto possa utilmente integrare tali sistemi.

4.5 ADDESTRAMENTO E VALUTAZIONE

4.5.1 IL PROBLEMA DELLA VALUTAZIONE

Una classica definizione degli algoritmi in grado di apprendere mette in relazione la capacità di apprendimento al "miglioramento delle performance" con l'addestramento. Purtroppo, uno dei problemi più critici nel campo dell'intrusion detection è esattamente come valutare un "buon" sistema di IDS.

Se in campo pratico si possono dare svariati suggerimenti operativi su come installare e provare un IDS, è molto difficile stabilire una metrica di performance per un sistema di intrusion detection. Intuitivamente vogliamo misurare:

- quanto il sistema sia capace di segnalare dei problemi, e
- quanto il sistema sia prono a segnalare dei falsi problemi

Per dare una quantificazione di queste variabili, ricordiamo che i risultati di un IDS si definiscono:

- Veri Positivi, VP, quando viene generato un allarme per un problema realmente presente
- Veri Negativi, VN, quando l'IDS non segnala nessun allarme e realmente non vi sono problemi
- Falsi Negativi, FN, quando di fronte a un problema il sistema non segnala l'allarme
- Falsi Positivi, FP, quando viene segnalato un allarme a fronte di una situazione normale

Una metrica molto banale sarebbe quella di calcolare, ad esempio, gli errori rispetto alla totalità di eventi analizzati: $\frac{FP+FN}{VP+VN+FN+FP}$; tuttavia, questa metrica avrebbe il grosso svantaggio di dare ottime valutazioni anche ad un "finto IDS" che non segnali mai alcun problema (FP = 0), semplicemente perché gli eventi pericolosi (che per un sistema simile sarebbero tutti falsi negativi) sono, sperabilmente, molto pochi rispetto alla totalità. Definiamo dunque, in analogia a quanto fatto in altri campi, due quantità a nostro parere più significative:

La detection rate, $DR = \frac{VP}{VP+FN}$, che risponde al primo criterio intuitivo

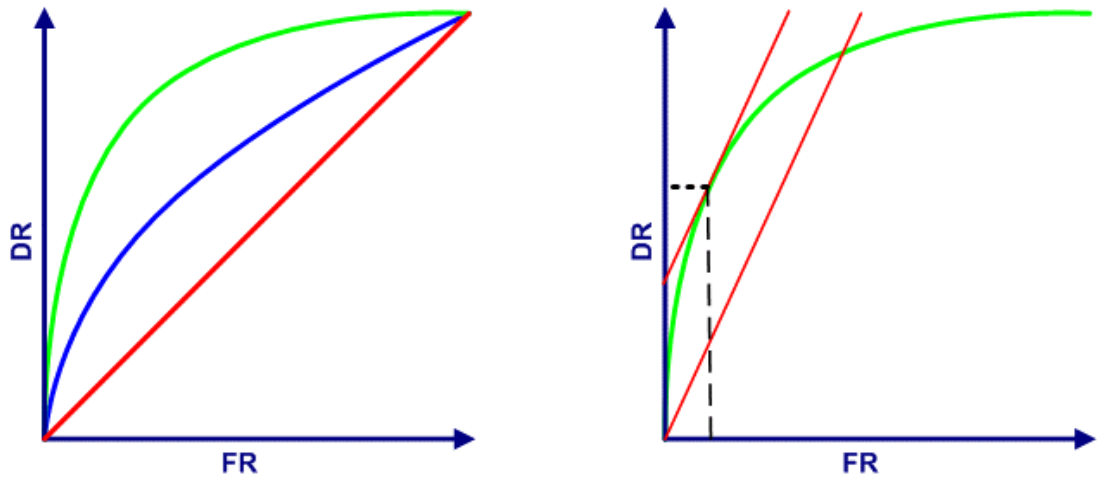
La false positive rate, $FR = \frac{FP}{VN+FP}$, che risponde al secondo criterio

È facile vedere la somiglianza di queste due metriche con le classiche misure di precision, recall e fallout tipicamente usate nella valutazione dei sistemi di information retrieval. In particolare, la DR è esattamente equivalente al recall, mentre non esiste un equivalente diretto della false positive rate che tuttavia cerca di catturare l'inverso del concetto di precision. Quindi, mentre precision e recall sono due metriche entrambe "positive", DR e FR sono l'una positiva e l'altra negativa.

Intuitivamente, queste due variabili (esattamente come precision e recall) sono collegate da un trade off, specialmente nel campo dei sistemi ad individuazione di anomalia. Per convincersene, basti pensare ad un sistema ipotetico che produce un numero (poniamo, compreso tra 0 e 1) che rappresenta un livello di "intrusione"; dovremo stabilire una soglia a cui far corrispondere un segnale di avviso: il trade off diviene chiaro, più alziamo la soglia, più riduciamo i falsi positivi, eliminando però anche una serie di indicazioni utili.

Non sappiamo con precisione quale sia la forma di questo trade-off, ma possiamo di certo dire che, posto che esista un parametro (o più parametri) di "soglia" o di "sensitività", denominato per ipotesi t, entrambi i valori sono funzione di t: $FR = f(t), DR = f(t)$

È pertanto indubbiamente possibile rappresentare queste due quantità su un grafico, e tracciando la curva parametrica rispetto al parametro t che le mette in relazione. Nella letteratura relativa all'analisi dei segnali e anche in quella relativa agli algoritmi di classificazione, questo tipo di grafico viene denominato ROC, Receiver Operating Characteristics.



Grafici R.O.C.: sulla sinistra il grafico comparato di tre algoritmi, sulla destra determinazione del punto di ottimo per un singolo algoritmo.

Come si può notare abbiamo messo in ascissa il false error rate, in ordinata il detection rate. Altrettanto ovviamente i due assi sono da considerarsi in scala compresa tra 0 e 1. Ovviamente, un IDS che non genera nessun errore avrà DR pari a 0 (in quanto VP=0), FR = 0 (in quanto FP = 0), mentre un IDS che genera errori su tutto avrà DR = 1 (in quanto FN = 0), FR = 1 (in quanto VN = 0). Tra questi estremi i comportamenti possono essere i più vari.

Ovviamente, la bisettrice disegnata nel grafico a sinistra rappresenta le performance minime plausibili, corrispondenti a quelle di un classificatore assolutamente casuale. Non è possibile fare peggio di così (se un grafico R.O.C. fosse tutto sotto la bisettrice, significherebbe che il sistema sta segnalando i dati in modo invertito). Inoltre, in generale ci aspettiamo che le curve dei R.O.C. siano monotone non decrescenti: accettando un aumento di FR, non ci aspettiamo in nessun modo una diminuzione di DR, anzi, ci aspettiamo che aumenti anche essa, fino a raggiungere 1.

Dal momento che a seconda del parametro di sensitività ci spostiamo lungo questa curva, dobbiamo determinare un criterio per valutare quale delle curve corrisponda al "miglior IDS". Intuitivamente comprendiamo che l'area sottesa dalla curva è legata alle performance del sistema: quanto maggiore è l'area sottesa, tanto migliori sono le performance globali del sistema. Questa definizione è evidentemente poco operativa, o comunque impraticabile per una valutazione automatizzata delle performance.

Inoltre, non è assolutamente detto che questo criterio di dominanza "globale" sia valido tout court. Infatti, in generale, i "costi" che associamo ad un falso positivo o

ad un falso negativo sono diversi e comunque soggettivi. Per esempio potremmo dare come specifica il fatto che per noi è molto più importante che non ci siano falsi allarmi, rispetto all'interesse che abbiamo per l'individuazione degli eventi di intrusione; o magari tutto l'opposto. Denominando α il costo di un falso positivo, e β il costo di un falso negativo, e chiamando p la proporzione di "eventi positivi" sul totale, il costo atteso per la classificazione è:
$C(FR, DR) = FR \cdot \alpha \cdot (1 - p) + (1 - DR) \cdot \beta \cdot p$

Possiamo tracciare sul piano R.O.C. il gradiente della funzione costo, che è una retta con coefficiente angolare $\frac{\alpha \cdot (1 - p)}{\beta \cdot p}$. Vediamo un esempio di questo nel grafico a destra. Sfruttando tale retta, possiamo determinare il punto di minimo per la funzione costo sulla curva R.O.C., e pertanto un valore di sensitività adeguato ai parametri da noi stabiliti.

Ciò che risulta evidente è che localmente ad una determinata scelta dei parametri, il minimo determinato per la funzione C può essere più basso per un algoritmo che, globalmente, ha delle performance peggiori.

4.5.2 SET DI DATI PER LA VALUTAZIONE E L'ADDESTRAMENTO

Un secondo problema, di ordine pragmatico, è la necessità di un set di dati ampio e completo da utilizzare in molteplici occasioni:

- Negli esperimenti e valutazioni progettuali, oggetto di questo lavoro di tesi
- Nella determinazione della correlazione tra i parametri dei vari algoritmi e la "sensitività" dell'algoritmo
- Nella taratura preliminare del sistema, una volta realizzato, e nelle valutazioni di prestazioni necessarie
- Infine, per l'addestramento e la taratura finale del sistema, al suo inserimento pratico nella specifica rete da monitorare

È evidente che la situazione ottimale sarebbe quella di disporre già in fase di progettazione concettuale di un set di dati ricavato dal vivo sulla rete a cui il sistema è destinato. Tuttavia, questa soluzione non è praticamente attuabile per evidenti ragioni. Pertanto, fermo restando che l'addestramento finale del sistema debba essere compiuto utilizzando dati ricavati sulla specifica rete in cui dovrà operare, per le prime tre fasi avremo bisogno di un insieme di dati di riferimento.

Questi dati devono avere le seguenti caratteristiche:

- *Realismo*: devono essere stati generati su una vera rete, dal vivo o in condizioni di funzionamento simulato realistiche.
- *Completezza:* devono contenere tutti i dati, sia la traccia delle connessioni che il contenuto dei pacchetti, possibilmente in un formato utilizzabile facilmente.
- *Pre-classificazione:* vogliamo dei dati che abbiano una legenda di riferimento, in cui eventuali situazioni anomale inserite o riscontrate abbiano una guida di riferimento che ci aiuti a individuare veri e falsi positivi.

Dopo accurate ricerche, abbiamo trovato che l'unico insieme di dati oggi disponibile in rete e di libero uso che presenti contemporaneamente tutte queste caratteristiche, ed in aggiunta sia stato specificamente creato allo scopo di effettuare benchmark e sperimentazioni su sistemi di intrusion detection, è il dataset del Lincoln Laboratory del M.I.T., altrimenti noto come "DARPA IDS Evaluation dataset" [56]. In particolare per i nostri esperimenti abbiamo usato porzioni del dataset del 1998, ampiamente descritti e commentati in un lavoro di tesi [57].

Laddove, nel seguito, ci riferiamo a campioni di dati TCP/IP, sottintenderemo sempre che essi provengano da tale dataset, a meno che vi siano espliciti riferimenti contrari.

4.5.3 ADDESTRAMENTO

Entrambi gli stadi dell'algoritmo hanno evidentemente necessità di essere addestrati. Ci occuperemo nel seguito, nei rispettivi capitoli, delle modalità di addestramento di ciascuno. Qui ci preme sottolineare alcuni concetti fondamentali legati alla struttura globale del sistema.

Innanzitutto, come abbiamo mostrato poco fa, manca un criterio operativo che ci consenta di tenere traccia del miglioramento complessivo di performance del sistema. È pertanto impossibile utilizzare un criterio globale come funzione di performance, e dobbiamo demandare allo studio dei singoli stadi le procedure di addestramento. Un interessante spunto di ricerca che rimane aperto sarebbe stabilire un metodo di validazione per la connessione in cascata dei due algoritmi, ovvero dimostrare la nostra supposizione, sperimentalmente confermata in più casi anche in letteratura, che al miglioramento delle performance dei vari componenti corrisponda un miglior comportamento globale dell'algoritmo.

Inoltre, dalla struttura del sistema risulta evidente che l'algoritmo di secondo stadio potrà essere addestrato soltanto dopo un addestramento iniziale dell'algoritmo

di primo stadio. Fortunatamente, la differente prospettiva dei due algoritmi rende possibile un addestramento disaccoppiato. Nel capitolo 5, pertanto, analizzeremo le performance degli algoritmi di classificazione utilizzati senza tenere in considerazione il collegamento al secondo stadio, ma solo i requisiti che ci siamo posti. Analizzeremo poi tutte le implicazioni di questa scelta nel capitolo 6.

La connessione in cascata di due algoritmi di apprendimento non supervisionato è una tecnica relativamente inusuale, le cui implicazioni teoriche non sono state approfondite completamente. L'architettura da noi proposta potrebbe beneficiare di ulteriori studi teorici su questo problema.

5

PROGETTO DEL PRIMO STADIO

5.1 DEFINIZIONE DEL SOTTOPROBLEMA DA ANALIZZARE

Nel primo stadio, dobbiamo gestire il seguente problema: abbiamo dei vettori di massimo 1460 elementi di dimensione byte (0-255), che devono essere categorizzati in modo da ridurli a 1 o 2 numeri significativi. I vettori rappresentano blocchi di dati che possono essere, in realtà, appartenenti ad un numero molto alto di tipi di protocolli e di tipi di dati.

Ci si potrebbe chiedere, a questo punto, perché non basarsi semplicemente sulle conoscenze di dominio (ovvero le well-known port, ad esempio), per pre-suddividere tali dati in differenti tipi di traffico (traffico http, traffico FTP, traffico telnet). La risposta è semplice e si articola in tre considerazioni.

Innanzitutto, se utilizzassimo le porte come indicatore di classificazione, daremmo per scontato che ogni singola porta stia, effettivamente, venendo usata per il tipo di protocollo che usualmente la sfrutta. Questo ci impedisce per esempio di osservare delle anomalie, come una risposta a un comando http che in realtà assume la forma di un output di comando di shell.

In secondo luogo, una ricerca precedente ha dimostrato la capacità degli algoritmi neurali di classificare e distinguere diversi tipi di protocollo [58]. È pertanto presumibile che un algoritmo di classificazione possa riconoscere e distinguere tipi diversi di traffico, anche se non con una precisione totale. Come vedremo, abbiamo verificato almeno in parte questa intuizione.

Infine, per come ci siamo posti il problema, dobbiamo cercare di semplificare il nostro approccio il più possibile, per non incorrere in tutte quelle tipologie di errori di interpretazione che minano la robustezza di molti IDS.

Si noti che nel seguito discuteremo la realizzazione di un primo stadio dedicato al payload dei pacchetti TCP. È nostra opinione che un identico primo stadio, separato, debba essere realizzato per UDP, mentre per ICMP possa essere sufficiente un passaggio di decodifica. Entrambi possono essere realizzati con delle semplici modifiche a quanto descriveremo per il TCP.

5.2 CLASSE DI ALGORITMI DA ANALIZZARE

5.2.1 ALGORITMI DI CLUSTERING NON SUPERVISIONATO

Nella premessa che intendiamo utilizzare un algoritmo *non supervisionato* di apprendimento, è evidente, dal problema che ci siamo posti, che non siamo di fronte alla necessità di fare knowledge-discovery (algoritmi di data-mining), bensì una riduzione dei dati a dimensione trattabile, ovverossia una classificazione non supervisionata o *clustering*.

Secondo una definizione classica, clustering è il "raggruppamento di oggetti simili all'interno di un determinato insieme" [59], ovvero "un algoritmo mediante il quale degli oggetti sono raggruppati in classi, in modo tale da minimizzare la distanza intra-classe e massimizzare la distanza inter-classe" [60].

La definizione, sebbene intuitiva e apparentemente semplice, dà origine ad un ampio ventaglio di problematiche sui metodi migliori per effettuare questa suddivisione, nonché sui criteri in base ai quali misurare la similarità o la dissimilarità tra elementi. Infatti, una soluzione completa ed ottima del problema sarebbe non polinomiale.

Gli algoritmi di cluster analysis sono stati sviluppati trasversalmente in numerose branchie della scienza, in particolare ricevendo un esplosivo interesse negli ultimi anni (grazie all'accresciuta quantità di dati disponibili e di potenza di calcolo, e al desiderio di processare questi dati per ottenere delle informazioni). Tipiche attività in cui si riscontrano problemi di clustering sono: segmentazione delle immagini, per esempio per una più accurata risoluzione delle query di similarità in un database di immagini; riconoscimento di pattern e riconoscimento ottico dei caratteri (OCR); analisi dei database di tipo GIS e delle immagini satellitari; studio della biologia molecolare per la determinazione delle aree di aggancio fra proteine, e per i progetti correlati al genoma umano; information retrieval, in particolare relativamente all'estrazione di informazioni dal web. Inoltre algoritmi di clustering vengono usati nel campo del data-mining come algoritmi di preprocessing, per semplificare i dati.

In un famoso articolo di alcuni anni fa [61], J. Frank, in uno studio sulle applicazioni futuribili dell'intelligenza artificiale alla robotica, indicò gli algoritmi di clustering come uno dei possibili "futuri" approcci all'intrusion detection mediante tecniche di apprendimento. Possiamo confermare che la sua intuizione, anche se con un certo ritardo, si è rivelata esatta.

5.2.2 ALCUNI REQUISITI PER UN ALGORITMO CANDIDATO

Innanzitutto, l'algoritmo che stiamo cercando deve consentirci di trattare con costi computazionali accettabili vettori di input da 1460 elementi. Non si deve credere che tale requisito sia, in assoluto, estremo: in molti dei domini applicativi indicati prima vi sono dati ad altissima dimensionalità. Certamente, molti degli algoritmi di clustering più semplici devono essere adattati o ristudiati per poter essere applicati a dati di alta dimensionalità senza problemi di efficienza. Ci interessa solo relativamente invece che l'algoritmo possa trattare grandi dataset: certamente, più "tempo di rete" l'algoritmo può osservare in addestramento, migliore sarà la qualità delle sue suddivisioni, ma questo è un problema relativamente secondario.

In secondo luogo, non abbiamo idea di "quante classi" possano servire per descrivere i dati. Per questo, dovremo guardare con occhio critico le "assunzioni" iniziali che facciamo nell'applicazione degli algoritmi, e premettiamo che tutti i nostri risultati andrebbero ulteriormente espansi per cercare di determinare criteri meno "euristici" di quelli da noi impiegati per inizializzare gli algoritmi. In particolare, vi sono tre criteri per stabilire se la classificazione raggiunta da un algoritmo non supervisionato è soddisfacente: inspection-based, expert-based, e task-based. Il primo criterio suggerisce l'esplorazione "manuale" delle suddivisioni create dall'algoritmo; il secondo un raffronto con una classificazione fatta da un esperto; il terzo, molto intelligentemente, suggerisce di misurare la performance dell'algoritmo una volta calato nel suo "task" specifico. Nell'impossibilità di disporre di una classificazione "da esperto" per uno solo dei due componenti dell'algoritmo, ovviamente il criterio migliore sarebbe quello "task-based". Tuttavia, le ovvie limitazioni imposteci dal lavoro di laurea ci richiederanno di affidarci, il più delle volte, ad una valutazione ispettiva delle classi create dagli algoritmi.

L'algoritmo deve altresì rappresentare in modo possibilmente intuitivo e semplice da utilizzare i criteri di suddivisione che ha utilizzato. Sarebbe comodo poter esplorare questi criteri per avere una idea "umanamente significativa" della suddivisione.

Sarebbe anche estremamente interessante che l'algoritmo prevedesse tra i suoi compiti l'individuazione degli outlier, ovvero dei dati che ricadono molto al di fuori

dai cluster individuati: "un outlier è una osservazione che devia talmente tanto rispetto alle altre, da farci sospettare che sia stata generata da un meccanismo completamente diverso" [62]. Potremmo per esempio definire un outlier come un dato che dista da un cluster più di quanto distino, mediamente, il 99% degli altri dati osservati. Questo però non rende giustizia della presenza di outlier "locali", che sono più distanti, nella loro regione di spazio, da un cluster, rispetto a tutti gli altri punti della stessa regione. Questo perché, specialmente in domini in cui le metriche non sono così chiaramente definite, ci potrebbero essere grosse variazioni nella distribuzione locale dei punti. Esistono metodologie più approfondite [63] che tengono invece conto della densità locale dei punti, e che potrebbero essere integrate in una versione definitiva del nostro sistema.

Inoltre, con l'aumentare della dimensionalità del problema (ed è certamente il nostro caso) alcune di queste metriche basate sulla distanza perdono il proprio valore a causa dell'estrema dispersione dei punti, ma sono stati proposti metodi specifici [64] per l'individuazione di outlier in queste situazioni che potrebbero, eventualmente, essere adottati.

È interessante notare che alcuni algoritmi già prevedono l'individuazione di outlier come una caratteristica binaria (un dato è un outlier oppure non lo è), mentre altri metodi possono essere adattati, per esempio dando una misura della distanza tra l'ingresso appena classificato e i suoi compagni di cluster. In generale si possono distinguere un insieme di test "statistici" basati sulle distribuzioni, e dei test basati sulla "distanza". Per ogni metodo sottolineeremo se una valutazione degli outlier è già prevista o può essere facilmente realizzata.

Sarebbe inoltre interessante scegliere un algoritmo che ci dia la possibilità di essere "riaddestrato" o "tarato" man mano che analizziamo dati. Il problema è che molti algoritmi che per natura sono incrementali sono anche in qualche modo "order-dependent", il che potrebbe essere un grosso svantaggio.

Analizziamo infine il tipo di dati in ingresso. Secondo letteratura, esistono tre macrocategorie di dati che possono essere dati in ingresso ad un algoritmo di clustering:

1. dati quantitativi, ossia semplici dati numerici; ogni vettore rappresenta quantitativamente una serie di caratteristiche;
2. dati categorici: dati sui quali non ha senso eseguire operazioni matematiche (ad esempio, il calcolo della distanza) o predicati se non l'uguaglianza; la misura di similarità in questi casi può essere definita in modo "creativo", ad esempio similarità(T1,T2) = $|T1 \cap T2| / |T1 \cup T2|$

3. dati metrici: dati per i quali è definita una funzione distanza (cioè una funzione che rispetta i quattro assiomi della metrica) che associa a due dati un numero reale rappresentate la distanza fra essi; ovviamente i dati quantitativi rientrano in questa categoria che però risulta estendibile: ad esempio il dominio delle parole può essere considerato metrico se si considera una funzione distanza definita come il numero di lettere da sostituire per trasformarle una nell'altra (distanza di edit).

Ovviamente i dati da noi analizzati non sono "quantitativi". Tuttavia è legittimo chiedersi se possano essere considerati dati "metrici", secondo una qualche particolare definizione di metrica. Presenteremo alcune prove sperimentali che ci incoraggiano a considerare i dati come "metrici". Sarebbe tuttavia un interessante spunto di ricerca ripetere le nostre analisi e le nostre prove con algoritmi di clustering dedicati a dati categorici. Inoltre, data la peculiare natura di questi dati, algoritmi con capacità di gestire regioni e cluster non sferici potrebbero avere un vantaggio.

5.3 ANALISI SINTETICA DEGLI ALGORITMI DI CLUSTERING

5.3.1 PROPRIETÀ DEGLI ALGORITMI DI CLUSTERING

Gli algoritmi di clustering possono esibire varie proprietà:

- Un algoritmo può essere **gerarchico** o **piatto** (in inglese si usa il termine "partitioning" per gli algoritmi piatti, ma lo troviamo fuorviante, in quanto troppo simile al concetto di "divisivo" collegato agli algoritmi gerarchici). Un algoritmo gerarchico suddivide, per raffinamenti successivi, i dati, in classi sempre meno generali. Un algoritmo piatto suddivide i dati in un insieme di classi tutte dello stesso "livello". Un algoritmo gerarchico, spesso, produce risultati più comprensibili dalle persone, ma è meno efficiente computazionalmente di un algoritmo piatto. Inoltre un algoritmo gerarchico difficilmente può "correggere" un eventuale errore commesso inizialmente.
- Gli algoritmi di clustering gerarchici possono essere **divisivi** oppure **agglomerativi**: possono cioè partire considerando tutti i dati come un unico grande insieme e ricorsivamente dividerlo, oppure possono considerare ogni dato come elemento di un "singoletto" e cercare di unirli ricorsivamente.
- Un algoritmo di tipi piatto può essere **iterativo** (cioè ripetere l'assegnazione dei dati di training nei vari cluster a ogni passo dell'addestramento) oppure no.
- Si parla anche di **hard clustering** e di **soft clustering**, ovvero di algoritmi che suddividono i dati in classi in modo netto (ogni dato finisce per

appartenere a una data classe), e algoritmi che invece attribuiscono a ogni dato una certa probabilità di appartenere a una certa classe. Alcuni autori preferiscono parlare di **fuzzy** clustering e quindi di "gradi di appartenenza" alle varie classi, ma visto che la base di questi algoritmi è tipicamente statistica, ci sembra un termine fuorviante.

- Gli algoritmi **disgiuntivi** prevedono che i cluster siano tutti separati, ma è possibile anche che l'algoritmo preveda la possibilità per un dato di appartenere a più cluster diversi.
- Alcuni algoritmi rappresentano i propri cluster mediante un **albero** di condizioni di divisione (in particolare gli algoritmi gerarchici e divisivi), mentre altri li rappresentano mediante una congiunzione di condizioni (algoritmi piatti di vario tipo); altri ancora li rappresentano mediante **centroidi**, ed esistono parecchie varianti; infine, alcuni algoritmi rappresentano i cluster usando più **rappresentanti** (ad esempio CURE). È ovvio che gli algoritmi ad albero possono rappresentare una varietà di regioni diverse (che le possano poi scoprire dipende dall'algoritmo, non dalla rappresentazione). Una rappresentazione basata su centroidi funziona bene per regioni compatte di forma sferica o comunque convessa. Rappresentazioni mediante più punti descrivono bene cluster di forma generica, anche se in generale meglio se convessa.

Dal momento che parleremo spesso di "distanze" è d'uopo definire i vari modi per misurare le distanze inter-cluster e intra-cluster:

- **Inter-Cluster**
  - Single linkage: la distanza minima tra i due cluster, ovvero la distanza dei due vettori più vicini appartenenti uno ad un cluster e l'altro al secondo
  - Complete linkage: la distanza massima tra due cluster
  - Average linkage: la media delle distanze tra ogni coppia di vettori appartenenti il primo ad un cluster e il secondo
  - Centroid linkage: la distanza dei centroidi dei due cluster
  - Centroid "manhattan" distance: la distanza secondo il metodo Manhattan, ovvero la somma semplice delle distanze lungo le varie dimensioni (invece della radice quadrata della somma quadratica)

- **Intra-Cluster**
  - Distanza media: la media delle distanze di ogni coppia di vettori (detto anche "diametro")
  - Varianza: la media degli scarti quadratici rispetto al centroide (detto anche "raggio")

- Distanza "nearest neighbor": la media delle minime distanze, ovvero per ogni vettore si prende la distanza dal vettore che gli sta più vicino
- Distanza dal centroide: la media della distanza dal centroide

5.3.2 TASSONOMIA DEGLI ALGORITMI DI CLUSTERING

Qui riportiamo solo una breve tassonomia degli algoritmi più noti, stendendo poi delle brevi note che ci serviranno per giustificare le nostre scelte sugli algoritmi da sperimentare. Per una tassonomia più sviluppata si rimanda alla "Review" recentemente pubblicata dall'ACM [65].

- **Algoritmi Gerarchici**
  - **Algoritmi divisivi**
    - Principal Direction Divisive Partitioning
  - **Algoritmi agglomerativi**
    - Hierarchical Agglomerative Clustering
- **Algoritmi Non Gerarchici**
  - Neural Gas (NG)
    - NG
    - Growing NG (GNG)
  - Self Organizing Map (SOM)
    - SOM
    - Recurrent SOM (RSOM)
    - Growing Hierarchical SOM (GHSOM)
  - K-Means clustering (KMC)
    - KMC
    - Growing KMC (GKMC)
    - GKMC with Maximum Size
  - Algoritmi basati su K-Medoids
    - PAM
    - CLARA
    - CLARANS
  - Sequential Leader Clustering (SLC)
  - Algoritmi basati sulla densità
    - DBSCAN
    - OPTICS
    - DENCLUE
  - CURE

- BIRCH
- Algoritmi basati sulla riduzione dimensionale
  - CLIQUE
  - PROCLUS
  - MAFIA
  - OptiGrid
  - O-Cluster
- Algoritmi basati su griglie
  - STING
  - WaveCluster
- Algoritmi basati su grafi
  - Association Rules Hypergraph Partitioning (ARHP)
  - Minimal Spanning Tree Clustering (MST)
- AutoClass (Bayesian Classification, EM Clustering)
- Algoritmi Genetici
- ART

5.3.3 ANALISI SINTETICA DEGLI ALGORITMI DI CLUSTERING

5.3.3.1 Hierarchical Agglomerative Clustering

Questo è un algoritmo di **hard clustering** di tipo **gerarchico** e **agglomerativo**, e si basa sul seguente principio: dapprima, ogni dato viene considerato l'unico elemento di una classe a se stante; questi elementi saranno le foglie dell'albero gerarchico di suddivisione. In seguito, cluster che risultano essere "vicini" vengono "fusi", unendoli in un nodo dell'albero gerarchico. La misura della "distanza" tra i cluster viene stabilita secondo una delle quattro definizioni di distanza inter-cluster illustrate in precedenza.

Questo metodo ha il vantaggio di essere semplice da realizzare e performante. Ha, d'altro canto, lo svantaggio di essere molto sensibile al "tipo di distanza" prescelto (ognuna delle 4 definizioni di distanza genererà un albero diverso), e in una situazione come la nostra, in cui il concetto di "distanza" è già messo in discussione alla radice, non è una notizia incoraggiante. Notoriamente, infatti, l'utilizzo della distanza "complete linkage" tende ad approssimare bene solo cluster solidamente compatti, mentre il "single linkage" presenta un problema di "chaining", ovvero una densa catena di punti tra due cluster può indurre in errore l'algoritmo e farli condensare in uno solo.

Inoltre, il metodo è sensibile alla presenza di outlier già nei dati di training. Problema ancora più importante, è difficile utilizzare il metodo per generare un "certo numero" di classi, perché va stabilito "dove" fermare l'albero.

L'algoritmo non prevede la possibilità di un addestramento on-line, né risulta semplice immaginare qualche metodo "euristico" per consentirla.

Inoltre, estrarre da questo tipo di algoritmo un criterio per la successiva suddivisione di dati non originariamente presenti nel set di addestramento è una operazione possibile ma che presenta le sue complessità.

5.3.3.2 Principal Direction Divisive Partitioning

Si tratta di un algoritmo di **hard clustering** ricorsivo di tipo **gerarchico** e **divisivo**. Dapprima viene considerato un cluster formato da tutti i punti. Di questo cluster viene calcolato il centroide, inoltre viene calcolata la decomposizione a valori singolari della matrice degli elementi del cluster (SVD), di cui si prende in considerazione il primo valore e il relativo vettore associato, che è appunto detto *"direzione principale"*. Il centroide viene proiettato sul vettore, e il cluster viene diviso in due parti, con un iperpiano perpendicolare al vettore e passante per la proiezione del centroide. Il cluster originario diventa un nodo padre, con le due metà come figli.

A questo punto l'algoritmo si ripete ricorsivamente, sul più disperso degli insiemi foglia, fino al raggiungimento di un obiettivo che può essere un numero massimo di classi, oppure una soglia massima di dispersione prefissata. La misura della dispersione può essere effettuata in molti modi differenti, ma approfondiremo questo problema nello studio appositamente dedicato a questo algoritmo. Inoltre, resta problematico stabilire per il nostro "tipo" di dati una soglia massima di dispersione (vedremo anche questo), quindi ci ritroviamo a dover prefissare un numero di classi.

L'algoritmo produce per ogni "nodo" un criterio di classificazione molto semplice (costituito da una disuguaglianza). L'interpretabilità umana del criterio, in spazi a grandi dimensioni, è tuttavia minima. L'algoritmo non si presta ad un addestramento o ad una calibrazione on line, anche se esistono varianti proposte a questo fine.

5.3.3.3 K-Means clustering (KMC)

K-means è un algoritmo di **hard clustering** di tipo **piatto** ed **iterativo**. L'algoritmo inizia selezionando a caso un certo numero K di centroidi. Per ogni iterazione vengono svolte le seguenti operazioni: ogni vettore viene assegnato al

centroide più vicino; per ogni cluster così formato viene calcolata la media dei punti che lo compongono come nuovo centroide; con questi K nuovi centroidi viene ripetuto il procedimento. Il procedimento si ripete fino a quando i centroidi non si spostano più.

Vi sono un certo numero di presupposti in questo scenario. Innanzitutto un presupposto è che i dati siano dotati di una distanza metrica (l'algoritmo non può evidentemente funzionare per dati categorici, anche se alcune estensioni [66] sono state proposte a tale scopo). In secondo luogo, il parametro K va imposto a priori, e determinare il numero di classi corrette per un algoritmo che opera su dati in cui nemmeno noi sappiamo che suddivisione aspettarci è sempre difficile. Rispetto ad altri algoritmi che operano con un numero predefinito di classi, K-means è più sensibile ad errori nella stima del parametro K.

Infine, uno dei grossi problemi dell'algoritmo è l'inizializzazione. Infatti il procedimento converge, rapidamente, ad un ottimo locale nella distribuzione dei centroidi, ma è ampiamente possibile che inizializzazioni diverse conducano a soluzioni finali molto diverse tra loro. Inoltre, K-means non sopporta molto bene gli outlier.

Pur non esistendo un metodo vero per l'addestramento on line dell'algoritmo, è possibile facilmente immaginare delle soluzioni euristicamente accettabili. Uno dei grandi vantaggi è la relativa leggerezza computazionale dell'algoritmo.

Esistono delle varianti da tenere in considerazione:

Growing KMC (GKMC): aggiunge all'algoritmo un elemento tratto dagli algoritmi di crescita. Se la distanza tra il punto in ingresso e il cluster più vicino è superiore a una determinata soglia viene creato un nuovo cluster in cui inserire il nuovo punto, alternativamente l'algoritmo procede in modo normale. Questo sistema può sia risolvere il problema della predeterminazione dei K, che in qualche modo eliminare o ridurre la variabilità dovuta all'inizializzazione casuale dei centroidi. È tuttavia necessario determinare le dimensioni di questa "soglia" in base al problema affrontato.

GKMC with Maximum Size: come il precedente, ma in ciascun cluster può essere inserito un numero massimo predeterminato di elementi. In questo modo invece di avere un singolo cluster molto ampio le zone con alta densità di punti saranno caratterizzate da molti cluster ravvicinati tra loro.

Global K-Means: questo algoritmo [67] è una versione di K-means sviluppata appositamente per contrastare il problema della località e della dipendenza forte

dall'inizializzazione. L'algoritmo risolve ricorsivamente il problema di determinare K cluster, partendo dalla soluzione del problema con K-1 cluster e utilizzando K-means come metodo di ricerca locale.

Scalable K-Means: questo algoritmo è una versione di K-means adattata ad ampi insiemi di dati. Viene stabilito un buffer di dati. Il database viene "pompato" nel buffer fino a riempirlo. Su questo sottoinsieme di dati viene eseguito l'algoritmo K-means normale. Quindi, l'insieme nel buffer viene "compresso", in due fasi. Vengono eliminati i punti "molto vicini" ai centroidi (è difficile immaginare che questi finiscano in un cluster diverso) e le informazioni al loro riguardo vengono "riassunte" associate al centroide.

Sui punti restanti viene eseguito un secondo K-means, con molti più centri rispetto al K-means principale, e su questi viene eseguito un algoritmo agglomerativo con una certa soglia di dispersione. In tale modo vengono individuate "nuvole dense" di punti, che si possono spostare da un cluster all'altro ma probabilmente si sposteranno tutti assieme. Anche di questi punti vengono riassunte le informazioni fondamentali. I restanti punti vengono mantenuti, e viene caricata una nuova infornata di dati dal database. Questo tipo di algoritmo potrebbe essere usato anche per un aggiornamento di tipo incrementale di una suddivisione basata su K-means.

5.3.3.4 Self Organizing Maps (Kohonen Maps)

Una S.O.M. [68] è una rete neurale utilizzata per un apprendimento non supervisionato. Si tratta di un algoritmo di **hard clustering** di tipo **piatto** ed **iterativo**. L'algoritmo costruisce un insieme di nodi di dimensione prefissata, collegati tra loro secondo uno schema deciso dall'implementatore (solitamente rettangolare o esagonale), che hanno una loro posizione in uno "spazio dei nodi" solitamente bidimensionale (per questo viene chiamata "mappa"). Ogni nodo rappresenta una classe, e corrisponde pertanto ad un punto nello spazio n-dimensionale dei vettori di ingresso.

L'addestramento avviene come nel caso dell' algoritmo K-means, mediante una ricorsiva approssimazione con i centroidi, e con la differenza che in questo caso vengono aggiornati non solo i centroidi immediatamente più vicini ai vari input, ma anche tutti i centroidi che sono "adiacenti" ad esso nella mappa, con un peso relativo alla distanza (nello spazio dei nodi) dal nodo più prossimo. Questo la rende lievemente più pesante computazionalmente, ma soprattutto fa sì che l'algoritmo non "converga" a stabilità in modo rapido ed indolore. Inoltre, una inizializzazione random può far sì che anche la S.O.M. converga a soluzioni non ottimali.

Come vedremo, questa mappatura da uno spazio n-dimensionale a uno spazio solitamente bidimensionale fa sì che una S.O.M. sia al confine tra un algoritmo di clustering e uno di riduzione dimensionale (vedi apposito paragrafo). Grazie ad algoritmi come LabelSOM [69] è anche possibile associare ai nodi della mappa dei campioni rappresentativi (etichette) che possono essere usati per esplorare più comodamente le suddivisioni.

Una S.O.M. esibisce un discreto campionario di problemi da risolvere. Innanzitutto anche per questo algoritmo va pre-determinato il numero di nodi (cluster) necessari, anche se rispetto a K-means è più tollerante a scelte "eccessive" (grazie ai vincoli nello spazio dei nodi, è possibile che alcuni di essi risultino "vuoti", senza nessun vettore di addestramento associato). Tuttavia, si paga per questa flessibilità con la necessità di pre-calcolare le epoche di addestramento, o di creare un qualche criterio di stop più o meno plausibile, in quanto la "convergenza" non è prestabilita come nel caso di K-means: in effetti si può dimostrare che non esiste una funzione obiettivo per la S.O.M., ovvero che le regole di addestramento utilizzate non sono il "gradiente" di nessuna funzione. Il criterio di convergenza è dunque soggettivo. Alcuni articoli documentano che per insiemi di dati con ampie dimensioni, anche tale convergenza soggettiva della rete potrebbe essere troppo lenta. Per essere più efficiente a una S.O.M. potrebbe essere premesso un algoritmo di preprocessing che elimini le dimensioni ridondanti dei dati e li normalizzi, tuttavia alcune obiezioni a questa strada sono presentate nel paragrafo sulla riduzione di dimensionalità più avanti.

Vi sono ricerche ed algoritmi che sembrano indicare la fattibilità di un addestramento "on line" delle S.O.M., tuttavia la pesantezza dell'algoritmo di addestramento unita alla sperimentalità delle ricerche non ci danno grandissime aspettative in proposito.

Trovando la distanza tra il vettore e il neurone più vicino, questo indica quanto "distante" sia il vettore dal neurone in cui la rete lo ha classificato. In questo modo sarebbe possibile individuare alcuni outlier grazie al fatto che il centroide della classe in cui la rete lo ha classificato (per mancanza di neuroni) dovrebbe, a logica, risultare "più distante" dall'outlier che da tutti gli altri vettori di input. Altri outlier possono essere individuati, per esempio, dal fatto che cadono in una classe fino a quel momento vuota.

Anche le S.O.M. hanno alcune varianti "evolute" che dobbiamo tenere in considerazione:

Recurrent SOM (RSOM): Mantiene una memoria dell'output precedente, consentendo di presentare a una SOM sequenze temporali. Per quanto l'algoritmo abbia un suo fascino, proviamo concettualmente ad analizzare ciò che gli verrebbe presentato. I payload presentati alla rete appartengono a diversi flussi di dati, incongruenti: tra di essi non c'è una correlazione temporale non-stocastica. Un algoritmo con funzioni di "memoria" andrà invece adottato per il secondo stadio del nostro IDS.

Growing Hierarchical SOM (GHSOM): questo tipo di S.O.M cerca di risolvere il problema del dimensionamento della mappa. L'idea di base è quella di consentire alla mappa di crescere in profondità e larghezza: la mappa cresce in larghezza aggiungendo nuove unità alla rete durante l'addestramento, nelle aree dove vengono a mancare dei nodi, con un metodo simile a tutti gli algoritmi in crescita. Cresce inoltre in profondità addestrando una nuova S.O.M. nelle unità che rappresentano grossi cluster (pertanto si distacca dal modello di base dell'algoritmo piatto, assumendo caratteristiche gerarchiche). L'idea di base è quella di consentire creando una struttura flessibile e gerarchica che si evolve basandosi sui dati. L'algoritmo è innovativo ed interessante, ma andrebbe estensivamente sperimentato prima di essere utilizzato (esiste ben poca letteratura sperimentale in proposito).

5.3.3.5 Sequential Leader Clustering (SLC)

L'algoritmo è estremamente semplice: il dato in ingresso viene analizzato, ne viene calcolata la distanza da ognuno dei cluster precedentemente trovati, e viene determinato il cluster più vicino: se la distanza tra il punto in ingresso e il cluster più vicino è inferiore ad una certa soglia il punto viene associato al cluster, altrimenti il punto diventa il primo componente di un nuovo cluster. La distanza tra punto e cluster può seguire varie metriche (nearest neighbor, farthest neighbor, centroid), ma l'algoritmo è sostanzialmente tutto qui, ed è anche abbastanza rozzo nei suoi risultati. Questo algoritmo si presta anchesì in modo molto intuitivo a un addestramento on line. Si tratta di un algoritmo di **hard clustering** di tipo **piatto** e **non iterativo**.

Bisogna definire la soglia di distanza con attenzione, e l'ordine con cui i dati vengono presentati può causare variazioni enormi nella suddivisione. Questa estrema variabilità è ancora più preoccupante in un problema in cui le conoscenze di dominio sono, volutamente, poche.

5.3.3.6 Neural Gas (NG)

Questo algoritmo prevede la presenza di un determinato numero di neuroni, che si comportano in modo molto simile a quelli di una S.O.M. o ai centroidi di K-means.

Durante l'addestramento viene calcolata la distanza di ogni input da ogni cluster, e l'adattamento di ogni neurone ad ogni input viene pesato in modo inversamente proporzionale a tale distanza. La distanza diventa sempre più importante man mano che l'addestramento prosegue. È dunque un algoritmo di **soft clustering** di tipo **piatto** ed **iterativo**

L'algoritmo condivide pregi e difetti con K-means e le S.O.M.: come in K-means, è necessario predeterminare il numero di neuroni; come nelle S.O.M. la convergenza è più lenta (specialmente ad altissime dimensioni), anzi, è ancora più problematica perché ogni input influenza potenzialmente tutti i neuroni: bisogna dunque fissare un numero di epoche di training o un parametro di stop. L'algoritmo non si presta in modo intuitivo ad un addestramento online.

Esiste, anche per Neural Gas, una variante evoluta nota come *Growing NG (GNG)*. L'algoritmo viene inizializzato con due cluster non connessi. Per ogni dato in ingresso vengono calcolati il cluster più vicino ed il secondo più vicino. La distanza quadratica del dato dal cluster più vicino viene sommata all'errore locale cumulato del cluster. Tra i due viene inserito un arco di età 0 (o viene azzerata l'età dell'arco esistente). Gli archi uscenti dal cluster vincente la cui età supera una certa soglia vengono rimossi. Se ciò produce dei cluster disconnessi da ogni altro cluster questi vengono eliminati. Ora vengono ricalcolati i pesi del cluster più vicino e di tutti quelli ad esso connessi. Inoltre ogni tot passi viene inserita una nuova unità: l'algoritmo cerca il cluster con il massimo errore accumulato, e il cluster ad esso connesso con i più alti errori accumulati; viene rimosso l'arco tra i due cluster, e viene inizializzato un nuovo cluster interpolato tra questi due, a cui vengono poi collegati con due archi i cluster originari. Gli errori cumulati dei due cluster originari vengono diminuiti di un coefficiente, e al nuovo cluster viene attribuito un errore accumulato basato su tali valori.

5.3.3.7 K-Medoids

K-Means è un algoritmo molto apprezzato per la sua intuitività e veloce convergenza, tuttavia non è apprezzabile la sua dipendenza dall'insieme di centri scelti casualmente all'inizio del processo.

Gli algoritmi di tipo K-Medoids sono **piatti** e **iterativi**, basati sull'idea fondamentale di K-means, ma con un approccio inverso. Scelti a caso i K punti rappresentativi, l'algoritmo K-medoids cerca di verificare, per ogni punto, se ne esista un altro all'interno del dataset con cui è vantaggioso sostituirlo.

Capostipite degli algoritmi K-Medoids è PAM, Partitioning Around Medoids, che applica proprio letteralmente il procedimento delineato sopra. È in grado di generare dei buoni cluster, ma ha il grosso difetto di essere pesantissimo computazionalmente: $O(k(n-k)^2)$, con n numero di oggetti e k numero di medoidi.

CLARA [70] (Clustering Large Applications) è un tentativo di migliorare PAM utilizzando sottoinsiemi dei dati di training. Il costo computazionale è $O(kS^2 + k(n-k))$, con S numerosità dei sottoinsiemi estratti. L'efficienza dipende però da tale numerosità, e ridurre il parametro S significa esporsi al rischio di produrre un clustering "falsato" se l'insieme di dati estratti non fosse un campione adeguatamente rappresentativo.

CLARANS [71] (Clustering Large Applications based upon RANdomized Search) è il capostipite degli algoritmi di clustering per database di tipo spaziale, ed è un miglioramento di CLARA. Applica una metodo di ricerca casuale tra i punti adiacenti ogni medoide per determinare un clustering ottimo localmente, poi passa a un set differente di medoidi per determinare un altro ottimo locale, eccetera. CLARANS ha un costo che è di gran lunga superiore a $O(n)$, e sebbene dipenda da variabili statistiche, si approssima a $O(n^2)$, richiedendo più di una scansione del dataset. Le sue performance sono state migliorate con l'applicazione di algoritmi R*.

5.3.3.8 Algoritmi basati sulla densità

DBSCAN (Density Based Spatial Clustering of Application with Noise) è stato il primo algoritmo ad utilizzare una nozione di densità. L'idea base di DBSCAN è che un cluster sia una regione di punti ad alta densità, circondata da una regione a bassa densità che la separa dagli altri cluster. In altre parole, ogni dato, per appartenere ad un cluster, deve avere entro un certo raggio almeno un certo numero di altri pattern: ossia, la densità dei dati nelle vicinanze del punto considerato deve superare una certa soglia. Tale raggio viene denominato Eps, mentre la soglia minima di punti MinPts. Il "raggio" può essere stabilito usando una qualsiasi funzione di distanza che rispetti i quattro assiomi della metrica.

DBSCAN inizia da un punto arbitrario. Calcola il cosiddetto Eps-neighborhood del punto (ovvero un insieme che contiene tutti i punti del database che stanno in una ipersferetta di raggio Eps centrata nel punto). Se tali punti sono più di MinPts questo punto inizia a formare un cluster. La procedura viene ricorsivamente ripetuta per tutti i punti della Eps-neighborhood, e così via. Ad un certo punto si determineranno dei dati che appartengono al cluster ma non hanno più di MinPts punti nel loro Eps-neighborhood: questi punti sono i limiti del cluster o "border points", mentre invece tutti gli altri sono detti "core point". La complessità di questo algoritmo è $O(n^2)$,

tuttavia l'utilizzo di una struttura di accesso ad albero paginato e bilanciato per dati spaziali, quindi con altezza logaritmica nel numero n di elementi del database, (R*-Tree, MVP-tree, M-tree), può portare la complessità ad $O(n \log(n))$. Dati sperimentali confermano la subquadraticità del costo, ma solo per dati di dimensionalità bassa: ad alta dimensionalità gli alberi perdono in efficienza e il costo torna circa $O(n^2)$. Altri autori suggeriscono ulteriori miglioramenti di efficienza basati sull'uso della disuguaglianza triangolare. Si tratta di un algoritmo di **hard clustering, piatto** e **non iterativo**.

Il vero problema di DBSCAN, tuttavia, come di tutti gli altri algoritmi di questa classe, è la sua sensibilità alla scelta dei parametri Eps e MinPts, per cui non esistono euristiche tranne quella suggerita dagli stessi autori. Inoltre, i parametri sono "globali", non adattandosi quindi alla presenza di cluster di varia forma e dispersione.

L'algoritmo non soffre invece della tipica vulnerabilità agli outlier che affligge tutti gli algoritmi che considerano tutti i punti del cluster, in quanto le considerazioni di densità attenuano questo problema, classificando come "noise" i punti tanto dispersi da non essere "density-connected" a nessun cluster. Esiste una versione incrementale dell'algoritmo.

OPTICS (Ordering Points To Identify Clustering Structures) nasce proprio con lo scopo di superare i problemi collegati alla scelta dei parametri. Gli autori osservano come, fissato un valore di Eps, diminuendo il valore di MinPts aumenta il numero di punti che vengono classificati come "core points". Viceversa, fissato MinPts, diminuire Eps può causare due effetti: aumentare il numero di punti che vengono classificati come "border points", e far sì che alcuni punti precedentemente appartenenti a un cluster divengano outliers.

Per ogni punto processato, OPTICS calcola una core-distance (ovvero il minimo valore di Eps tale per cui la neighborhood di quel punto contiene esattamente MinPts), e una "reachability distance" tra due dati, ossia il minimo valore di Eps tale che i due dati siano directly-reachable. L'algoritmo ha la stessa complessità di DBSCAN. Basandosi su questi dati è poi possibile estrarre i "risultati di DBSCAN" per vari valori di Eps con complessità lineare. Tuttavia mediante considerazioni di tipo diverso è possibile utilizzare OPTICS come algoritmo di clustering "a se stante", risolvendo anche parzialmente il problema della dipendenza forte dai valori di Eps e MinPts. Le considerazioni sulla complessità permangono simili a quelle fatte per DBSCAN.

DENCLUE (DENsity CLUstEring), invece, è un metodo di clustering basato sulla densità caratterizzato da una solida base matematica. Tuttavia, per non complicare eccessivamente l'esposizione, ci limitiamo a proporlo in termini intuitivi. Innanzitutto DENCLUE tratta esclusivamente vettori numerici nello spazio a D dimensioni, quindi possiamo significativamente parlare di "punti" al posto di "dati". L'idea di base di DENCLUE consiste nell'immaginare che i punti si "influenzino" a vicenda e, ovviamente, tale influenza sia tanto maggiore, quanto minore è la distanza che li separa; più specificatamente, si formula una "funzione di influenza" che indica con precisione l'influenza di un punto su di un altro.

In pratica, se la funzione che descrive l'influenza del punto y sul punto x è $f(x,y)$ (in generale funzione della distanza tra x e y), la "funzione influenza" del punto $\bar{y}$ è $f^{\bar{y}}(x) = f(x,\bar{y})$.

Intuitivamente, in un cluster denso di punti, la somma dell'influenza sarà maggiore che in altre regioni dello spazio. Definiamo per ogni punto la funzione densità, come somma delle influenze di tutti gli altri punti in quel punto: $f^D(x) = \sum_{i=1}^{N} f^{y_i}(x)$, con N numero dei punti.

È evidente la dipendenza dell'algoritmo dal tipo di funzione influenza. Gli autori propongono una funzione a gradino, $f(x,y)=1$ se $|x-y| \le \sigma$, $f(x,y)=0$ altrimenti; e una funzione gaussiana, $f(x,y)=e^{-\frac{|x-y|^2}{2\sigma^2}}$, ma altre funzioni possono essere proposte ed utilizzate.

Altrettanto evidente è che il parametro σ o "density factor" governa l'estensione dell'influenza. Per trovare i cluster, DENCLUE inserisce la nozione di "density attractor", ovverossia dei massimi locali della funzione densità, e di "density attracted". Un punto è "attratto" se esiste una catena di punti da esso a un massimo locale della funzione, tali che per ogni punto della catena il "gradiente" della funzione densità è rivolto verso il punto successivo. Il gradiente viene definito da DENCLUE come:

$\nabla f^D(x) = \sum_{i=1}^{N} (x_i - x) f^{x_i}(x)$.

Il cluster "centrato" nel density attractor $\bar{x}$ viene definito come l'insieme dei punti x "attratti" da $\bar{x}$ e tali che $f^{\bar{x}}(x) > \varepsilon$. Possono venire definiti degli "arbitrary shaped clusters" come l'unione dei cluster di più centri, posto che tra i centri esista un "percorso" di punti con densità superiore a ε.

Notiamo, per esempio, che se utilizziamo una funzione a gradino, DENCLUE diventa identico a DBSCAN, con $\varepsilon = MinPts, \sigma = Eps$. Anche in questo caso, i parametri vanno stimati ad occhio, esistendo delle euristiche.

L'algoritmo effettivamente usato utilizza una suddivisione dello spazio mutuata dagli algoritmi "grid-based", suddividendo tutto in ipercubetti di lato 2σ. La complessità totale risulta in tal modo ridotta a $O(n \log(n))$

In generale mancano in letteratura, per tutti questi algoritmi, prove applicative su problemi reali con alta dimensionalità dei dati.

5.3.3.9 BIRCH - Balanced Iterative Reducing and Clustering using Hierarchies

BIRCH [72] è un algoritmo di **hard clustering** di tipo **gerarchico** e **divisivo** per la suddivisione di grandi quantità di dati, sviluppato per ottimizzare l'utilizzo della memoria e l'ammontare delle operazioni di I/O. L'osservazione su cui si basa BIRCH è che lo spazio dei dati non è occupato in modo uniforme, e pertanto non tutti i dati sono ugualmente rilevanti. Birch gestisce solo dati metrici, e tratta cluster sferici.

Per ogni cluster, viene mantenuta in memoria una "clustering feature", che è una tripla (N,LS,SS) dove N è il numero di punti nel cluster, LS è la somma lineare di tutti i componenti del cluster, e SS la somma quadratica. Si può dimostrare che mantenere l'elenco completo di tali clustering features è più che sufficiente per calcolare tutte le distanze inter-cluster che abbiamo definito, oltre a raggio e diametro di ogni cluster. In aggiunta, se CF1 (N1,LS1,SS1) e CF2 (N2, LS2, SS2) sono due cluster disgiunti, la CF del cluster ottenuto dalla loro unione è ovviamente data da (N1+N2,LS1+LS2,SS1+SS2).

L'algoritmo costruisce un albero CF-tree, che è un albero bilanciato in altezza. Vi sono due parametri, il branching factor B e il threshold T. In pratica l'albero è costituito da nodi, ciascuno dei quali contiene al massimo B puntatori a dei figli. A ogni puntatore (cioè ad ogni figlio) viene associato il relativo CF. Alle foglie dell'albero troviamo una serie di CF di singoli subcluster. T è la massima soglia di raggio o diametro di un cluster ammesso. Al crescere di T l'algoritmo crea cluster sempre più ampi. L'impostazione rende evidente come BIRCH lavori al meglio su cluster di dimensioni e dispersione paragonabili.

La costruzione dell'albero avviene leggendo ad uno ad uno i record da suddividere, e piazzandoli automaticamente ciascuno in un subcluster appropriato.

BIRCH è un algoritmo locale, ogni decisione di clustering può essere presa basandosi sul singolo input in esame. In tal modo il costo di I/O cresce linearmente con la dimensione dell'insieme di dati e già una singola passata dell'insieme può dar luogo a un partizionamento corretto. Per ogni dato, partendo dalla radice, si usano i CF di ogni sottoramo dell'albero per calcolare la distanza tra la entry e tale sottocluster e discendere.

Raggiunta una foglia, scegliamo il subcluster più prossimo al dato. Se aggiungendo il dato non violiamo il vincolo di T, abbiamo finito, e aggiorniamo il CF. Altrimenti, aggiungiamo il cluster, dopodichè creiamo una nuova foglia, separando dagli altri il cluster "più lontano" e ponendolo in una foglia a parte. In tal caso dobbiamo ovviamente aggiornare il nodo padre, e così via ricorsivamente, ricordandosi che nei nodi c'è anche il limite imposto dal branching factor B.

Un possibile miglioramento dell'algoritmo, in caso di split, suggerisce che nel nodo dove lo splitting si ferma, ovvero il primo nodo che ha spazio per accogliere un nuovo ramo, si scelgano le due CF più vicine e si fondano i relativi rami, dopodichè li si ri-suddivida col criterio dei "più lontani" come prima se si supera il branching factor. In ogni caso i CF di tutti i nodi al di sopra della foglia vanno aggiornati: se non c'è uno split, semplicemente aggiungendo al CF padre i valori del nuovo input, altrimenti vanno effettuati alcuni calcoli intuitivi per gli spostamenti.

Può essere aumentato T per ridurre di dimensioni l'albero. Esiste un apposito algoritmo di re-scanning che a partire dal CF-tree con un dato T può costruire il CF-tree con un T più elevato. Un teorema mostra la dipendenza inversa tra T e le dimensioni dell'albero, inoltre, la trasformazione richiede uno spazio in memoria predeterminabile e dipendente dall'altezza dell'albero. Inoltre in fase di rebuild possono essere individuati e scartati dei gruppi di outlier, da reinserire a posteriori per minimizzare lo spazio occupato dall'albero.

Vi sono altri accorgimenti, relativi al riordinamento delle foglie e al trattamento corretto di più copie di dati uguali, che tralasciamo in questa descrizione sommaria dell'algoritmo.

Al termine si ottiene un "campionario" di cluster, che possono essere utilizzati per applicare algoritmi di clustering tradizionali, considerando ciascuno dei centroidi dei cluster, pesato col numero di punti presenti, come una "rappresentazione concentrata" dei dati.

I costi dell'algoritmo sono $O(dnBh)$ per la costruzione dell'albero, e $O(rdLBh)$ per il reinserimento di una "leaf entry", con d = dimensione dei dati, n =

numero di dati, B e L sono rispettivamente il numero di figli di un nodo e il numero di cluster nelle foglie, h = altezza dell'albero dal nodo a una foglia, r = numero degli spostamenti.

BIRCH è stato modificato per essere implementato in degli spazi metrici generici, non necessariamente vettoriali, dando vita agli algoritmi BUBBLE e BUBBLE-FM [73], che sono tra i pochi applicabili a tali spazi.

5.3.3.10 CURE – Clustering Using REpresentatives

CURE [74] è un algoritmo di **hard clustering** di tipo **gerarchico** ed **agglomerativo** che cerca di superare almeno due delle limitazioni tipiche degli algoritmi tradizionali, ovvero i problemi derivanti dalla presenza di outliers nei dati di addestramento, e la necessità di identificare cluster di forma non sferica senza utilizzare come descrizione tutti i punti del cluster (il che renderebbe l'algoritmo non scalabile).

CURE descrive ogni cluster con un insieme di n punti (con n parametro scelto) generati scegliendo a campione punti del cluster che siano "ben dispersi". A ogni passo dell'algoritmo questi vengono "spostati" verso il centroide del cluster di una certa frazione α della loro distanza da esso. In questo modo eventuali "outlier" presi erroneamente come punti descrittori vengono rapidamente riportati in linea. Il coefficiente α può variare da 0 a 1, con 0 l'algoritmo diventa sostanzialmente equivalente a un algoritmo che descriva il cluster con tutti i punti, con 1 diventa sostanzialmente un algoritmo basato su centroidi.

La complessità spaziale dell'algoritmo è $O(n)$ e quella temporale $O(n^2 \log(n))$, che può ridursi a $O(n^2)$ se i dati sono di bassa dimensionalità (non è sicuramente il nostro caso). Nonostante alcune affermazioni degli autori dell'algoritmo, il fatto che la curva di crescita del tempo di elaborazione con la dimensionalità del data-set sia stata omessa dall'articolo in cui viene presentato l'algoritmo, e che i ragionamenti parlino di "high-dimensional dataset" riferendosi a un set a 40 dimensioni, ci fanno presupporre che l'aumento di dimensionalità richieda un numero molto alto di punti "rappresentativi", complicando oltremodo i calcoli e/o creando problemi di memoria.

5.3.3.11 Algoritmi basati sulla riduzione dimensionale

CLIQUE [75] (Clustering In QUEst), appositamente studiato dai ricercatori di IBM presso il laboratorio di Almaden per lavorare in alta dimensionalità, si basa sulla ricerca di cluster in sottospazi dello spazio dei dati.

Più precisamente, l'ipotesi da cui parte CLIQUE è che in determinati sottospazi dell'insieme dei dati esistano cluster interessanti, più definiti che nello spazio completo. Al contrario dei metodi di riduzione dimensionale che tratteremo in seguito, CLIQUE non scarta a priori alcune dimensioni, ma cerca in vari sottospazi e proiezioni cluster interessanti. È da notare che CLIQUE considera solo sottospazi proiettati mediante la "soppressione" di alcune dimensioni, e non mediante la ricombinazione lineare di dimensioni, onde evitare di creare cluster lungo dimensioni di difficile interpretazione umana.

CLIQUE deriva le sue intuizioni sia dagli algoritmi basati sulla densità che da quelli basati sulla suddivisione dello spazio. I cluster sono, per CLIQUE, zone ad alta densità di punti, calcolate sfruttando la quantizzazione dello spazio dei dati in ipercubetti di lato ε, che è un parametro dell'algoritmo. La regione viene considerata densa se vi si trovano più di τ punti. Omettiamo i dettagli che sono analoghi a quelli di altri algoritmi basati sulla densità.

Il punto focale, ovviamente, sta nell'identificare quali sono i sottospazi che contengono cluster. Il metodo procede in modo bottom-up, sfruttando il fatto che un cluster rilevato in un sottospazio a dimensioni K sarà un cluster anche in qualsiasi sottospazio di questo sottospazio a dimensioni K-1.

L'algoritmo è composto di passi che prendono in ingresso tutte le regioni dense di dimensione K-1 (chiamiamolo insieme D_{k-1}), e restituisce un superset dell'insieme delle zone dense a dimensione K. Queste vengono analizzate per determinare le zone realmente interessanti, e una volta ottenuto l'insieme delle regioni dense K-dimensionali, queste vengono ricorsivamente usate come input per calcolare i candidati sottospazi K+1 dimensionali, eccetera. Il passo consiste nel fare una operazione di join tra l'insieme D_{k-1} e se stesso, ponendo come condizione che le prime K-2 dimensioni delle zone dense siano eguali. Le regioni individuate da queste intersezioni sono K-dimensionali e sono un superset di D_k. Sfruttando il teorema di monotonia, proiettiamo tali zone a K-1 dimensioni: se non sono presenti in D_{k-1} non sono zone dense in K dimensioni. Quindi otteniamo D_k.

L'algoritmo inizia dunque con una "passata" in cui analizza dimensione per dimensione trovando zone dense sulle singole dimensioni, e prosegue in modo ricorsivo. Sventuratamente, ciò significa che la complessità sia $O(c^k + mk)$, con k massima dimensionalità di un sottocluster denso, m il numero totale dei dati in ingresso, e c una costante. Il numero è evidentemente esplosivo, ma esistono ottime euristiche di pruning.

Gli output D_k rappresentano l'insieme delle regioni dense, che possono essere suddivise per ogni spazio K-dimensionale. A questo punto CLIQUE con un passo di costo $O(2KN)$, con N numero di regioni dense del sottospazio, le unisce in un insieme minimale di cluster connessi presenti. Tuttavia, descrivere questi cluster in modo ottimo in base alla griglia di suddivisione dello spazio (ovvero, col numero minimo di ipercubi) è un problema NP-completo [76]. Il metodo approssimato generale descrive i cluster con regioni sovradimensionate, con un bound di ln(K) volte il minimo [77]. CLIQUE usa un metodo in due passaggi (dapprima copertura greedy e in seguito rimozione) ciascuno dei quali opera con un bound che è $O(n^2)$.

Test approfonditi dimostrano performance reali che sono lineari rispetto alle dimensioni del database, e quadratiche rispetto al numero di dimensioni. Test di paragone con algoritmi di clustering standard mostrano che BIRCH per esempio "nota" cluster in sottospazi a 5 dimensioni se lo spazio originale ha meno di 10-15 dimensioni, dopodichè essi scompaiono in una nuvola di rumore. DBSCAN si comporta pure peggio. Meno chiara la comparazione con una riduzione dimensionale SVD.

PROCLUS [78] è un altro algoritmo con le stesse caratteristiche, con performance simili, che parte da un approccio diverso, sostanzialmente estendendo il meccanismo dei K-medoidi di CLARANS alla proiezione su subspazi. ORCLUS [79] applica lo stesso approccio alla determinazione di sottospazi arbitrari.

MAFIA [80] (Merging Adaptive Finite Intervals And is more than a clique) è una evoluzione di CLIQUE che implementa algoritmi adattativi per il dimensionamento non uniforme della griglia, risultando fino a 50 volte più performante. OptiGrid [81] cerca a sua volta di calcolare una griglia ottimizzata, con metodi diversi da quelli di MAFIA e cercando di usare iperpiani generici; purtroppo si dimostra sensibile ai parametri in ingresso all'algoritmo che ne controllano il comportamento. Una versione più evoluta di OptiGrid è O-Cluster. Quest'ultimo algoritmo ritorna al concetto di partizionamento in celle parallelo agli assi (in quanto è dimostrabile che i benefici di un partizionamento in celle non parallelo agli assi diminuiscono con l'aumentare delle dimensioni, mentre invece i problemi computazionali aumentano). La complessità di O-Cluster è $O(N \cdot d)$, con N numero di dati e d numero di dimensioni.

5.3.3.12 Algoritmi basati sulla suddivisione dello spazio (grid-based)

Questa categoria di algoritmi utilizza un approccio sostanzialmente distinto da tutti gli altri. Anziché ragionare sui dati, infatti, ragionano sullo *spazio*, che viene

quantizzato in un numero finito di celle (generalmente degli iperparallelepipedi) sulle quali vengono effettuate le operazioni di clustering.

Si può comprendere immediatamente che questa metodologia punti a rendere la velocità di computazione indipendente dal numero di pattern da classificare, ma dipendente dal numero di celle in cui lo spazio viene suddiviso. Inoltre, "riassumere" il contenuto delle celle in un singolo numero consente di mantenere anche grandi set di dati in memoria, velocizzando i calcoli.

Esistono molti algoritmi che adottano tecniche differenti in questo sottoinsieme, li esploriamo solo rapidamente:

- STING [82] (A Statistical Information Grid approach) adopera un approccio statistico, ed è stato studiato per problemi di data-mining di tipo spaziale. STING può essere facilmente parallelizzato, e utilizzato in modo incrementale. In sostanza crea una gerarchia di celle di "volume" sempre maggiore, e usa formule matematiche per calcolare i valori statistici delle celle superiori rispetto a quelle inferiori. Su questi valori si possono poi effettuare varie operazioni tra cui il clustering.
- WaveCluster [83] è una tecnica di clustering che si basa sull'applicazione di una trasformata wavelet ai dati in ingresso; tale approccio è impraticabile per dati ad alta dimensionalità
- Di CLIQUE, MAFIA, OptiGrid e O-Cluster abbiamo già parlato trattando degli algoritmi che riducono la dimensionalità. È interessante come questi due approcci si integrino così bene.

Tutti questi algoritmi presentano comunque una complessità lineare nel numero di dati. Vi è inoltre una buona capacità di riconoscimento di cluster di forma arbitraria e una modesta sensibilità del risultato rispetto ai parametri di ingresso. Il limite di questi metodi risiede nella qualità dei cluster forniti, che dipende in modo significativo (logicamente) dalla quantizzazione effettuata. Tuttavia, è da notare che con l'aumento delle dimensioni il numero di sottocelle esplode esponenzialmente, anche se sono stati sviluppati algoritmi per creare una suddivisione in celle ottimale (ad esempio nei citati OptiGrid, O-Cluster e MAFIA). Pertanto, gli algoritmi privi di tali accorgimenti potrebbero benissimo trovarsi in difficoltà con l'altissima dimensionalità del nostro caso.

5.3.3.13 Algoritmi basati su grafi

Uno degli algoritmi concettualmente più semplici da utilizzare per il clustering di un insieme di punti è la costruzione e il taglio del allbero minimo di copertura (Minimal Spanning Tree) del grafo che li rappresenta, una delle operazioni più studiate nell'algebra e nell'algoritmica di base.

Dato un grafo G formato da V nodi ed E archi dotati di pesi, l'MST è definito come il minimo sottografo connesso e aciclico di G che contiene tutti i nodi V. Esistono dei ben noti algoritmi che producono un albero minimo di copertura con un tempo estremamente efficiente, $O(E \log(V))$, e sfruttando spazio in ragione di $O(E+V)$. Dopodichè, una considerazione intuitiva ci dice che i punti appartenenti allo stesso cluster saranno uniti dagli archi di peso minimo all'interno dell'MST, mentre quelli appartenenti a cluster diversi saranno uniti da cluster di peso più alto. Pertanto ci limitiamo a disconnettere il grafo, rimuovendo progressivamente archi (in ordine di peso, dal più alto al più basso), fino a ottenere il numero di cluster desiderato.

Un algoritmo decisamente più efficace ed efficiente è Association Rules Hypergraph Partitioning (ARHP). Questo metodo [84] rappresenta i dati di grandi dimensioni in un modello a ipergrafo. In tale modello, ogni dato viene rappresentato come un vertice e le relazioni tra dati simili vengono rappresentate come archi. Il peso dell'arco riflette la forza della vicinanza tra i vertici. Un algoritmo di partizionamento viene applicato all'ipergrafo per trovare una suddivisione tale che i punti contenuti in ogni cluster siano fortemente correlati e gli archi tagliati dalla suddivisione siano di peso minimo. È un algoritmo di **hard clustering** di tipo **piatto** e **non iterativo.** In pratica si utilizza la stessa schematizzazione dell'algoritmo MST, ma considerando anche la densità degli archi che connettono vertici appartenenti allo stesso cluster, non soltanto la minimalità degli archi tagliati dalla suddivisione.

Il partizionamento degli ipergrafi è un problema studiato nel contesto dello sviluppo dei circuiti VLSI ed esistono algoritmi quali HMETIS che consentono nel trovare con ottima efficienza una buona soluzione al problema (ancorché non ottima). In particolare la complessità di questo algoritmo è $O((E+V)\log(K))$, dove V è il numero dei vertici, E il numero degli archi, e K il numero di cluster. L'algoritmo ovviamente non consente di fare addestramento online. Inoltre, il numero di classi deve essere necessariamente conosciuto a priori.

La correlazione tra i dati si può basare su regole di associazione (nei problemi di data-mining di tipo commerciale, ad esempio), ma anche su una metrica di distanza definita per ogni coppia di oggetti. Nel nostro problema sarebbe ovvio pensare all'utilizzo del secondo tipo di criterio, tuttavia la letteratura esistente descrive soprattutto problemi in cui l'algoritmo viene basato su un criterio del primo tipo: è tutta da dimostrare l'efficacia e l'efficienza di questo algoritmo nel contesto metrico.

L'efficacia viene messa in dubbio dalla mancanza di prove sperimentali che dimostrino che l'algoritmo produce una suddivisione di migliore qualità per questo

tipo di dati; ma anche le considerazioni di efficienza ci conducono ad essere dubbiosi. Infatti, costruire un grafo euclideo (ovvero, che contiene archi tra tutti i punti, ognuno dei quali dotato di un peso crescente in funzione della distanza) è relativamente semplice in due dimensioni: si tratta di costruire la triangolazione di Delaunay, per cui esistono algoritmi noti che hanno complessità temporale $O(V \log(V))$ e spaziale $O(V)$, ma che non scalano assolutamente a più di due dimensioni, in quanto la triangolazione diventa un problema esponenziale. Oltre le due dimensioni la complessità diventa quadratica, $O(V^2)$, inoltre l'algoritmo coinvolge il calcolo della distanza euclidea, che viene appesantito dalla crescita dimensionale.

Va ricordato, comunque, uno dei possibili punti di forza di questo tipo di algoritmo, che è la sua possibilità di descrivere regioni e cluster non sferici.

5.3.3.14 COBWEB

COBWEB è un sistema di clustering incrementale basato su alberi probabilistici. COBWEB è molto peculiare nel fatto che supporta, di base, solo attributi nominali, e non numerici.

Ogni foglia è costituita da una classe. Per ogni possibile valore di ogni feature, vengono definite le percentuali di probabilità all'interno della classe. Ogni nodo riporta i valori aggregati di tutti i suoi figli.

Per ogni nuovo dato COBWEB parte dalla radice dell'albero ed esegue ricorsivamente questo algoritmo:

COBWEB(dato,nodo)

1. Se nodo è una foglia, crea due figli, in uno inserisci la foglia, nell'altro il nuovo dato, e aggiorna
2. Aggiungi il nuovo dato al nodo e aggiornane le probabilità
3. Valuta il miglioramento nella Category Utility per ciascuna delle seguenti alternative
    a. creare una nuova classe per il dato
    b. inserire il dato nella migliore categoria figlia
    c. inserire il dato nella seconda migliore categoria figlia
    d. unire la prima e la seconda categoria in una categoria unica
    e. eliminare il nodo corrente e sostituirlo con le categorie figlie (split)
4. A seconda di quale sia il migliore metodo completa il passo
    a. Creare la nuova categoria e inserirci il dato
    b. Eseguire COBWEB(dato, c1)

c. Eseguire COBWEB(dato, c2)
d. Sostituire c1 e c2 con cm = c1+c2 ed eseguire COBWEB(dato, cm)
e. Eliminare il nodo corrente, sostituirlo con le categorie figlie e rieseguire COBWEB(dato, nodo_genitore)

La category utility è una funzione che cerca di massimizzare l'idea che abbiamo di cluster: istanze della stessa classe avranno frequentemente valori uguali, istanze di classi diverse avranno spesso valori diversi:

$$CU = \sum_C \sum_A \sum_v P(A = v \mid C) P(C \mid A = v) P(A = v)$$

Le tre parti della formula significano, rispettivamente, la probabilità che l'attributo A abbia valore v dato che il vettore appartiene a C, la probabilità che il vettore appartenga a C dato che A ha valore v, e la probabilità che A abbia, in generale, valore v (per "pesare" attributi comuni e non comuni). Per il teorema di Bayes:

$$CU = \sum_C \sum_A \sum_v P(A = v \mid C) \frac{P(C) P(A = v \mid C)}{P(A = v)} P(A = v) = \sum_C \sum_A \sum_v P(A = v \mid C)^2 P(C)$$

Estraendo un termine invariante, otteniamo: $CU = \sum_C P(C) \sum_A \sum_v P(A = v \mid C)^2$

Ma $\sum_A \sum_v P(A = v \mid C)^2$ non è altro che il valore atteso degli attributi che possono essere indovinati correttamente, sapendo che un certo vettore appartiene alla classe C (tirando a indovinare alla cieca la probabilità sarebbe $\sum_A \sum_v P(A = v)^2$).

In realtà la CU si definisce come "l'incremento nel numero atteso di valori che possono essere indovinati correttamente, dato un insieme di n categorie, rispetto al numero atteso di quelli che possono essere indovinati correttamente senza tale conoscenza", e vale pertanto:

$$CU = \frac{1}{N} \sum_C P(C) \sum_A \sum_v [P(A = v \mid C)^2 - P(A = v)^2]$$

dove il coefficiente 1/N viene inserito per consentire di confrontare cluster di diversa numerosità.

5.3.3.15 AutoClass (Bayesian Classification, EM clustering)

AutoClass [85] è un algoritmo **piatto** di **soft clustering** basato sul modello statistico detto "finite mixture model", dotato di un metodo Bayesiano per determinare il numero ottimo di classi. Questo algoritmo cerca di trovare il più probabili insieme di classi che descrivono l'insieme di dati di training, basandosi sulle aspettative a priori. La descrizione delle classi consiste in un tipo specifico di

funzione di densità di probabilità, e l'assegnamento di ogni dato a una classe viene fornito in termini di probabilità, piuttosto che in termini deterministici come normalmente avviene con gli altri algoritmi.

In pratica il modello descrive i dati usando due tipi di funzione di densità di probabilità: innanzitutto una bernoulliana che descrive la probabilità che un dato appartenga a una certa classe; un altra f.d.p. descrive invece, per ogni classe, la probabilità di osservare un determinato vettore di valori, posto che il vettore appartenga a quella classe. Tale f.d.p. è il prodotto delle funzioni di densità di probabilità dei singoli attributi (per quella classe) che possono essere indipendenti o covariate, Poissoniane, Bernoulliane, Gaussiane o altre. Come si può notare la costruzione dello schema di queste funzioni richiede l'inserimento di conoscenza e di modelli statistici che non abbiamo sui dati di rete che vogliamo classificare.

Per determinare i parametri, l'algoritmo effettua una stima mediante un processo di expectation-maximization (EM) che converge verso un massimo locale della funzione di verosimiglianza. Una delle approssimazioni che l'algoritmo impone è l'indipendenza tra le f.d.p. dei singoli attributi, ma ve ne sono altre, nascoste all'interno di passaggi di calcolo probabilistico troppo complessi per addentrarci in un esame approfondito.

L'algoritmo può determinare il numero più probabile di classi a se questo non è noto a priori, ma funziona notoriamente in modo più efficace se questo dato viene fornito.

Uno dei grossi limiti di AutoClass è che non funziona bene con dati di alta dimensionalità: la complessità computazionale è infatti $O(kd^2nI)$ con k numero di cluster, d numero di dimensioni, n numero di oggetti da clusterizzare, I numero medio di iterazioni dell'algoritmo.

Non ci risulta l'esistenza di metodi per il riaddestramento online, ed in ogni caso la complessità computazionale pare proibitiva in questo senso.

5.3.3.16 Algoritmi Genetici

Formulare un problema di clustering nei termini tipici di un algoritmo di tipo genetico (o ad altri algoritmi evolutivi) è a prima vista semplice. Come è noto, gli algoritmi genetici operano generando soluzioni candidate, valutandone l'adeguatezza (fitne ss), e facendo "riprodurre" le funzioni migliori, mediante speciali meccanismi (crossover e mutazione) che tentano di emulare il processo evolutivo delle specie.

Evidentemente, si può immaginare che lo spazio delle soluzioni sia ogni possibile suddivisione dell'insieme di training. Basta imporre un criterio di fitness (per esempio, la minimizzazione dello scarto quadratico nei singoli cluster) e siamo pronti ad applicare un algoritmo genetico al problema. Dobbiamo tuttavia codificare le singole soluzioni per poterle incrociare automaticamente. Sia T l'insieme dei dati di test, con t elementi. Per individuare K partizioni in T, il metodo intuitivo è quello di avere una stringa di t valori x1,x2…xt con 1<xi<K. In poche parole, a ogni elemento di T associamo il numero del cluster a cui appartiene in quella soluzione. Questo crea un problema di ridondanza: vi sono K! soluzioni perfettamente identiche (con numeri diversi). Inoltre, si mostra abbastanza semplicemente che l'operatore di crossover a punto singolo (si divide in due ciascuna stringa in un punto fissato ed il figlio prende parte dei dati del "padre" e parte di quelli della "madre") in questa rappresentazione può facilmente generare "figli" di qualità molto inferiore rispetto al padre.

Per questo sono stati proposti schemi diversi di codifica ed operatori di crossover migliorati; ad esempio una codifica in cui gli elementi di t vengono inseriti nella stringa di soluzione, separati da un * a marcare la divisione tra cluster, utilizzando l'operatore "permutazione" per il cross-over (con una soluzione simile a quella per il Traveling Salesman Problem [86]). Tuttavia questo non risolve il problema della ridondanza che è sempre fattoriale. Alcuni ricercatori [87] hanno anche proposto di usare un algoritmo genetico con un encoding dei dati su ipergrafo, simile a quello di ARHP, e usare un algoritmo di edge-based crossover [88], tuttavia l'ordine di complessità dell'operatore, che è $O(K^6 + N)$ con K numero dei cluster e N numero dei dati, lo rende estremamente sensibile alla crescita del numero di cluster, rendendolo improponibile per K > 10.

Altri problemi che rendono difficilmente praticabile, per ora, l'uso di GA in problemi pratici di classificazione non supervisionata è l'estrema sensibilità a macro-parametri quali la densità della popolazione di soluzioni, i parametri di crossover, eccetera. L'uso di GA in problemi di clustering viene reso efficace proprio da quelle conoscenze di dominio che in questo caso sappiamo di non avere. Inoltre, uno studio [89] dimostra come le performance dei GA rispetto ad altri metodi decrescano con il numero di dimensioni in esame.

5.3.3.17 ART

ART (Adaptive Resonance Theory) è un termine che indica una particolare struttura di algoritmo neurale basato sulla risonanza. La teoria che sta alla base della rete ART è che un input sufficientemente vicino a uno dei cluster riconosciuti dalla

rete la farà entrare in uno stato simile a quello del fenomeno fisico della risonanza, innescando un ciclo virtuoso di autoeccitazione.

Il modello originale (ART-1) è in grado solo di trattare dati booleani, mentre ART-2 [90] è in grado di gestire dati reali. ARTMAP è una implementazione supervisionata di ART, mentre Fuzzy ART estende il modello ART-1 per gestire dati fuzzy (e, incidentalmente, i dati reali, purchè normalizzabili).

Le reti ART sono capaci di effettuare learning non supervisionato e incrementale. Tuttavia, sono order-dependent, cosa che dal nostro punto di vista è negativa: esiste un algoritmo denominato IART-1 che diminuisce tale dipendenza ordinale, ma funziona solo per i valori booleani. Per migliorare ARTMAP (che è formato dalla connessione di due moduli ART-1) e diminuirne la dipendenza dall'ordine degli ingressi, è stato presentato Gaussian ARTMAP [91], basato su una evoluzione di ART-1 denominata Gaussian ART. Altre estensioni recentissime [92] al paradigma ART sembrano superare alcuni di questi problemi, ma è ricerca aperta.

Inoltre, un parametro globale detto "vigilance threshold" stabilisce quanto deve essere "nuovo" un input per stimolare la creazione di un nuovo cluster.

Non sono disponibili in letteratura valutazioni effettive sulla complessità computazionale di questo tipo di algoritmi (però ART-2 ha, riconosciutamente, problemi di complessità computazionale) né sulla loro adeguatezza a gestire dati di alta dimensionalità.

5.3.4 ALGORITMI DI RIDUZIONE DIMENSIONALE

Come abbiamo visto a più riprese, moltissimi algoritmi sono stati sviluppati per affrontare database di dimensioni continuamente crescenti (come quelli presenti nelle datawarehouse delle grandi imprese). Tuttavia, poco si è potuto fare per creare algoritmi che resistano alla cosiddetta "curse of dimensionality", ovvero la barriera ineliminabile contro cui si infrangono i tentativi di fare clustering su dati di alte dimensioni, come quelli che stiamo cercando di trattare noi.

Esiste una classe di algoritmi sviluppati appositamente per la riduzione dimensionale dei dati, ovverossia per cercare di proiettare uno spazio a N dimensioni in uno spazio a D dimensioni, con $D < N$, in modo tale che vengano preservate le caratteristiche di quei dati che ci interessano: in generale, parlando di dati su cui operiamo con algoritmi basati sulla distanza euclidea, vorremo che venga preservata la distanza relativa, ovvero che punti "lontani" nello spazio N dimensionale siano "lontani" anche dopo la proiezione.

Alcuni algoritmi principali sono:

- **Feature Selection:** esistono una varietà di algoritmi di "feature selection", che scartano parte delle dimensioni dei dati ritenendole poco significative. Un esempio banale potrebbero essere dei dati che nello spazio a 3 dimensioni sono tutti disposti sul piano (x,y): in tal caso evidentemente la coordinata z potrebbe essere soppressa conservando tutte le informazioni.
- **Latent Semantic Indexing (LSI):** con questo algoritmo le parole che compaiono frequentemente negli stessi documenti in cui ne compaiono delle altre vengono ritenute "simili" a queste. L'algoritmo crea una matrice di corrispondenza termine-documento (quindi uno spazio a moltissime dimensioni), preprocessandolo per eliminare desinenze, stop-word e verbi comuni. Pesa nella matrice i termini inversamente alla loro diffusione totale, e suddivide la matrice mediante una decomposizione per valori singolari, come avviene nella PCA (vedi sotto), ma senza passare per il calcolo della matrice di covarianza. L'algoritmo è utile nella compressione dei dati per analisi lessicali – ma non è assolutamente detto che sia utile per il nostro tipo di problema.
- **Principal Component Analysis (PCA):** tecnica molto nota, detta anche **trasformazione di Karhunen-Loève**. L'idea è quella di determinare la trasformazione da uno spazio originale m dimensionale a uno k-dimensionale con k<m che minimizza l'errore, ovvero la variazione nei rapporti di distanza tra punti. Si consideri un set di dati con n dati ed m variabili. L'algoritmo calcola la decomposizione a valori singolari della matrice di covarianza del set di dati M(m, n) e ne estrae i k autovettori principali. Questo autovettori sono le componenti principali dei dati, e sono delle combinazioni lineari delle componenti originarie. I dati vengono ora proiettati su queste direzioni e pertanto trasformati in uno spazio k-dimensionale. Esistono una serie di criteri per determinare il numero corretto di dimensioni basati sulle proporzioni della varianza o sulle radici caratteristiche della matrice di covarianza, tuttavia si tratta soltanto di euristiche. L'algoritmo ha una complessità computazionale superiore a $O(m^2)$ e pari a $O(m^2)$ in memoria.
- Anche le **SOM** a volte sono considerate preprocessing, infatti dopo che la rete è stata addestrata, ogni punto viene proiettato sullo spazio bidimensionale dei neuroni. Le S.O.M. però (al contrario di quanto accade per trasformazioni di tipo geometrico come le precedenti) non danno una misura di quanto "buona" sia la trasformazione.
- **Multi Dimensional Scaling (MDS):** questa tecnica considera le distanze tra coppie di punti e cerca di disporre i punti in uno spazio a dimensione inferiore, minimizzando un valore di stress. La funzione stress aumenta

quando punti lontani nello spazio a molte dimensioni si vengono a trovare vicini nella proiezione a dimensioni inferiori. Questo algoritmo è una versione euristica di PCA e LSI, e viene spesso usato per ottenere dei rapidi snapshot di dati multidimensionali su un piano o in uno spazio rappresentabili.

Dal momento che il nostro problema è effettivamente a grandissima dimensionalità, come mai ci soffermiamo relativamente poco sugli algoritmi di riduzione dimensionale? Perché siamo di fronte ad un problema molto difficile.

Questi algoritmi, difatti, vengono usati per "comprimere" le dimensioni sulla base di legami "quasi certi", che rendono alcune di queste dimensioni combinazioni "quasi lineari" di altre. Scartare le dimensioni che contengono pochi dati "non in linea" con queste relazioni predominanti è molto utile nel clustering puro, in cui ci interessa la maggioranza dei dati del cluster. Ma lo scopo di questo algoritmo di prima approssimazione, nel nostro caso, è duplice: comprimere ciò che si può comprimere, evidenziando però la presenza di outlier quanto più possibile.

I dati spurii vengono automaticamente "schiacciati" su quelli sani dalla trasformazione tra lo spazio a N dimensioni e quello ridotto a D dimensioni, perché le relazioni di distanza tra punti che vengono rispettate sono quelle della "maggioranza". Quindi abbiamo il sospetto che un approccio di questo genere elimini la possibilità di fare poi una outlier detection interessante. Vedremo come tra i vari test preliminari ve ne sia uno che ha confermato la nostra intuizione in proposito.

5.4 ESPERIMENTI DI IMPLEMENTAZIONE

5.4.1 METODI E OBIETTIVI

Ciò che ci proponiamo di fare ora è di analizzare se e come alcuni degli algoritmi di unsupervised clustering elencati possano essere utilizzati efficacemente per il primo stadio della nostra architettura.

Per fare ciò, intendiamo valutarne qualitativamente la capacità di estrarre classificazioni utili di dati del tipo da noi preso in considerazione, ed intendiamo altresì valutare le "prestazioni" di tali algoritmi. Intendiamo sottolineare che i nostri test di natura prestazionale non sono assolutamente da intendersi come benchmark assoluti (peraltro è noto [93] come i benchmark di qualunque natura siano spesso ingannevoli e non rendano ragione delle reali prestazioni dei sistemi in condizioni

effettive d'uso), ma come paragoni, relativi esclusivamente alla nostra implementazione, tra algoritmi diversi.

Per implementare gli algoritmi in questione abbiamo utilizzato Matlab. La scelta di una piattaforma per il calcolo, in alternativa all'implementazione diretta degli algoritmi, è stata fatta per due ordini di considerazioni. La prima è una considerazione di tipo puramente pratico: utilizzare algoritmi già testati e verificati ci ha consentito di essere molto più fiduciosi nella correttezza dei nostri risultati, e di passare più tempo sulla sperimentazione che sulla risoluzione di problemi di implementazione.

La seconda considerazione è relativa alla natura assolutamente sperimentale e dimostrativa del nostro lavoro. Abbiamo preferito una piattaforma meno performante, che ci desse però la possibilità di interagire passo-passo con i risultati ottenuti, grazie all'utilizzo della modalità interattiva tipica di Matlab.

Delineeremo descrivendo i vari esperimenti gli approcci seguiti e i dataset utilizzati, ma ci preme sottolineare ancora una volta che abbiamo utilizzato, sempre, insiemi di dati reali. Specificheremo di volta in volta i metodi impiegati per l'acquisizione e la generazione di tali dati.

5.4.2 SCELTE EFFETTUATE

Il numero ampio di algoritmi studiati per il clustering, di cui abbiamo esposto una sintetica rassegna, ci ha costretto ad operare una scelta preliminare, per individuare un gruppo di tre algoritmi "candidati" da implementare e sperimentare praticamente.

Abbiamo deciso di concentrarci su algoritmi che disponessero di implementazioni di esempio già realizzate, o che fossero semplici da realizzare, onde minimizzare il tempo dedicato al debugging rispetto a quello dedicato alla sperimentazione.

Con questa premessa, abbiamo scelto di sperimentare un algoritmo di tipo gerarchico, e almeno due algoritmi di tipo non gerarchico. Abbiamo scelto di utilizzare un algoritmo di tipo divisivo, in particolare il Principal Direction Divisive Partitioning, in quanto esso risponde in pieno al nostro requisito di generare una facile classificazione da usare poi su dati non utilizzati per l'addestramento.

Tra gli algoritmi di tipo non gerarchico, abbiamo deciso di esplorare le Self Organising Map di Kohonen, in quanto già utilizzate in letteratura per trattare dati di alta dimensionalità. Avremmo potuto, equivalentemente, utilizzare l'algoritmo

Neural Gas, tuttavia considerazioni di natura intuitiva ci fanno dubitare che esso sia adeguato a spazi con una dimensionalità tanto alta, in cui la distanza tra i centroidi, se calcolata nello spazio dei dati, potrebbe risultare scarsamente significativa.

Abbiamo poi implementato l'algoritmo K-means per una serie di motivi: il suo ampio successo, la leggerezza computazionale relativa, la semplicità. Come mostreremo, i risultati non sono stati in questo caso all'altezza delle aspettative.

Sarebbe interessante sperimentare anche una delle varianti di K-medoids, per verificare se il loro comportamento sia effettivamente migliore; tuttavia, la pesantezza computazionale ci è sembrata proibitiva. Un algoritmo basato sulla densità potrebbe, forse, ottenere risultati di buona qualità, ma il peso computazionale pare proibitivo. Idem dicasi per CURE e BIRCH.

Sarebbe da verificare sperimentalmente se i nostri dati sono nel campo di applicabilità di algoritmi di riduzione dimensionale quali CLIQUE o MAFIA. Tuttavia, una semplice osservazione sulla numerosità relativa dei pacchetti ci fa capire che nelle dimensioni più alte ci saranno relativamente pochi dati diversi da 0. Inoltre, sia questi algoritmi che quelli basati su griglie funzionano discretizzando dati continui: nel nostro caso abbiamo già dati a valori interi. Gli algoritmi basati su grafi sono di dubbia applicabilità al nostro problema, e di pesantezza computazionale rilevante. Considerazioni di pesantezza computazionale ci hanno convinti parimenti ad escludere AutoClass o simili algoritmi di divisione bayesiana.

Abbiamo preso in considerazione i modelli ART e li abbiamo esclusi, sia per la dipendenza dall'ordine di presentazione dei campioni, che per la scarsa disponibilità di valutazioni prestazionali su campi di alta dimensionalità. Infine, l'approccio genetico al clustering, profondamente interessante da un punto di vista teorico, ci pare praticamente troppo poco sperimentato e comunque costoso computazionalmente per essere applicabile al nostro problema.

5.4.3 PRIMO TEST: ANALISI DI UN LOG DI APACHE

5.4.3.1 Obiettivo del test

Dopo uno studio teorico preliminare, abbiamo cercato di capire immediatamente se il nostro approccio potesse o meno avere un senso. Per un test preliminare, abbiamo deciso di utilizzare l'algoritmo Self Organizing Map, e di provare a fargli suddividere dei dati.

Come primo insieme di dati, abbiamo pensato di utilizzare un log file di un server web Apache, estraendolo dall'insieme dei log di un sito web effettivamente presente su Internet. Abbiamo scelto di usare un log file per evitare, nel primo esperimento, tutta la complessità legata alla acquisizione e alla decodifica dei dati di rete; inoltre esistono categorie ben note di attacchi che possono venire portati ad un webserver esclusivamente utilizzando i comandi standard, che vengono quindi registrati all'interno del log file.

Questo ci ha consentito di creare in modo relativamente rapido un primo dataset di esempio, e qualche attacco di prova, per i primi esperimenti. Inoltre, i dati salvati all'interno di un log file costituiscono (limitatamente alle richieste HTTP) una buona approssimazione dei dati veri che l'algoritmo di primo stadio si troverà a dover classificare. Per simulare la situazione, abbiamo tradotto i vettori nello stesso formato (rappresentazione numerica) che i dati estratti dai dump di rete avranno.

L'esperimento ci ha inoltre aiutati ad esplorare le caratteristiche ed i limiti dell'implementazione delle SOM integrata in Matlab, e a prendere confidenza con i concetti della rappresentazione dei dati sotto forma di vettori. Le lezioni apprese in questo senso sono state preziose.

5.4.3.2 Implementazione

I dati di un logfile di Apache sono scritti in un formato standard (andiamo a capo per esigenze tipografiche, in realtà sono linee singole):

```
151.24.230.16 - - [07/Aug/2002:17:07:50 +0200] "GET
/link/nsbanner/400x40.gif HTTP/1.1" 200 13482
```

IP - - [data:ora +timezone] "comando URI protocollo/versione" risultato filesize

Di questa stringa abbiamo estratto esclusivamente l'URI, in quanto gli altri dati si riferiscono a informazioni non pertinenti al primo stadio (indirizzo IP, data, ora…) oppure sono invarianti o quasi (ci riferiamo alla struttura del comando "GET <URI> HTTP/1.1", costante nelle 999 righe utilizzate come campione).

Una volta addestrata la rete su queste 999 richieste "normali", le abbiamo sottoposto varie stringhe d'attacco, tra cui quella del virus NIMDA (anche qui l'interruzione di riga è per esigenze tipografiche):

```
/_vti_cnf/.%u002e/.%u002e/.%u002e/.%u002e/.%u002e/.%u002e/winnt/syst
em32/cmd.exe?/c+dir
```

Abbiamo implementato un semplice script che estraesse ed analizzasse i dati, addestrasse la rete e la utilizzasse per classificare, oltre agli stessi dati di test, anche le stringe "anomale". Ne presentiamo il codice sorgente commentato. I commenti sono preceduti da % nella tipica notazione di Matlab

```
% Ripuliamo la memoria
clear;

% Caricamento dei dati in un array dal file
[ip1 ip2 ip3 ip4 input_timedate input_command input_URI input_proto
input_result input_size] = textread('truedata.txt', '%3d.%3d.%3d.%3d
%*1c %*1c %28c %s %s %s %3d %s');

% Conversione dell'array degli URI da stringhe in un
% array di array di char
s = char(input_URI);

% Conversione da char a numeri:
t = double(s);

% Questa riga genera una matrice descrittiva degli input:
%  ogni input va da 32 a 127 (caratteri ASCII)
% la lunghezza e' la dimensione di s
PR = repmat([32 127],size(s,2),1);

% Creiamo una rete delle dimensioni volute
% per questo test 5x8 o 6x10 sperimentalmente funziona
 net = newsom(PR,[5 8]);

% Imponiamo il numero di epoche di addestramento
% Sperimentalmente 1000 epoche sono sufficienti
net.trainParam.epochs=1000;

% Addestramento della rete
net = train(net,t');

% Creiamo un vettore contenente un URI di attacco
% (in questo esempio, Nimda Worm)
strano =
'/_vti_cnf/.%u002e/.%u002e/.%u002e/.%u002e/.%u002e/.%u002e/winnt/sys
tem32/cmd.exe?/c+dir';
```

```
% Classifichiamo il vettore di attacco. La forma un po' convoluta è
% necessaria per portare il vettore a una lunghezza pari a quella
% degli altri URI
x = sim(net, (double([ strano repmat(' ', 1, size(s,2)-
length(strano)) ]))')

% Ora procediamo a classificare tutti i vettori di test
y = sim(net, t');

% Determiniamo a quali vettori corrisponda la classe in cui è finito
% l'attacco
y(find(x),:)

% Calcoliamo e disegnamo una tabellina delle classificazioni
% per vedere quali sono le classi più o meno numerose
[m n] = size(y);
[ (1:m)' full(sum(y,2)) ]
bar(full(sum(y,2)));
```

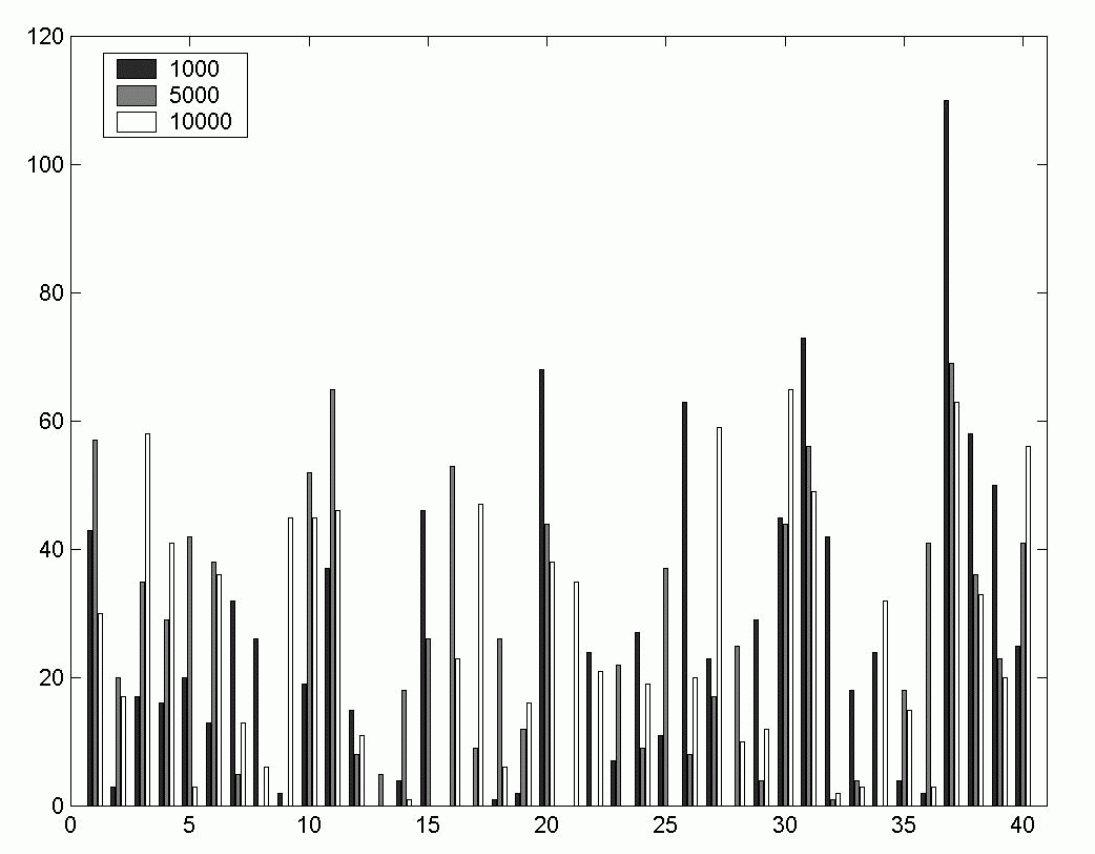
5.4.3.3 Applicazione ai dati

La rete è stata applicata a 999 righe di log file tratte da un sito web reale. Su di esse è stata addestrata per 1000 epoche, e in seguito le sono stati sottoposti per la classificazione vari input anomali del tipo di quello del virus NIMDA.

Il risultato è stato che con 1000 epoche di addestramento, il vettore d'attacco è stato classificato, da solo, nella categoria 21. Con 5000 epoche i risultati sono identici. Con 10000 epoche si verifica sperimentalmente (a volte succede nelle SOM) uno "scambio" di alcune categorie, e il vettore d'attacco finisce (sempre isolato) nella categoria 15. Riportiamo su un grafico (alla pagina successiva per esigenze tipografiche) la distribuzione dei cluster prodotti dall'algoritmo, per tre addestramenti di una rete S.O.M. di durata, rispettivamente, di 1000, 5000 e 1000 epoche.

Con questo esperimento preliminare riteniamo di aver mostrato che l'approccio di base su cui si fonda la nostra proposta ha senso, ed è possibile in linea di massima estrarre una suddivisione significativa dei dati di rete utilizzando tecniche di classificazione non supervisionata.

Riteniamo inoltre di aver mostrato in modo estremamente pragmatico che le differenze tra il traffico "normale" e i pattern di attacco possono essere riconosciute mediante un learning non supervisionato.



Suddivisione in classi operata da una rete S.O.M. di un log http. I tre toni indicano rispettivamente la suddivisione dopo 1000, 5000 e 10000 epoche.

Tuttavia il test nella sua preliminarietà lascia aperti molti dubbi. Innanzitutto, servono ulteriori verifiche prima di prendere seriamente in considerazione questo tipo di approccio. Questo nostro test è solo una dimostrazione che esiste una struttura di dati e che, forse, essa può essere utilizzata per riconoscere delle sequenze di attacco.

In secondo luogo il nostro test è estremamente limitato, sottoponendo a una SOM soltanto un particolare tipo di flusso di dati, le richieste HTTP. Ricordiamoci che nella struttura da noi proposta, l'algoritmo di secondo stadio dovrebbe trattare il payload dei protocolli di trasporto, senza scendere in una suddivisione dettagliata basata sul tipo di protocollo applicativo.

Rimane pertanto da dimostrare che il nostro approccio abbia un senso anche quando l'analisi viene estesa a dati misti derivanti direttamente dal dump dei pacchetti di rete. Questo sarà l'obiettivo dei paragrafi che seguono.

5.4.4 INTERLUDIO: DECODIFICA E ANALISI DI DATI TCP

5.4.4.1 Il problema

Giunto a questo punto necessitiamo di analizzare i dati di rete, che sono poi il nostro vero obiettivo. Il tipico strumento per ottenere una "registrazione" dei dati che passano su una rete è una utility UNIX denominata *tcpdump* [94]: ne esistono varie versioni, e una serie di utility minori che svolgono gli stessi compiti, ma in generale tutte utilizzano un formato di file compatibile, definito nelle librerie di cattura pacchetti *libpcap*.

In ambiente Windows esiste un port sia del software, denominato *Windump*, che delle librerie, *winpcap*, realizzato dal Politecnico di Torino [95]. Esiste inoltre un software integrato denominato *Ethereal* [96] che consente di analizzare in modo completo il traffico di rete, sia catturandolo che recuperandolo da un file salvato in formato libpcap. Ethereal funziona sia in ambiente Windows che UNIX, ed è dotato di utili funzioni di decodifica di protocollo a vari layer, oltre alla possibilità di filtrare pacchetti e ricostruire comunicazioni TCP.

Il dump di rete salvato nel formato libpcap, pur essendo la lingua franca di tutte queste applicazioni, non può venire importato in modo semplice in Matlab, e nonostante le nostre ricerche non siamo riusciti a reperire in rete librerie o script già realizzati a tale scopo, pertanto abbiamo dovuto sviluppare una soluzione. Utilizzando tcpdump, è possibile visualizzare (e salvare) i pacchetti decodificati nel seguente formato (anche qui, la prima linea va a capo per esigenze grafiche):

```
14:02:26.341844 197.218.177.69.1049 > 172.16.114.207.23: P
115:125(10) ack 641 win 32120 (DF) [tos 0x10]
                  4510 0032 02e6 4000 4006 a1d0 c5da b145
                  ac10 72cf 0419 0017 5732 db3a 2de4 dffd
                  5018 7d78 9e2b 0000 726f 6465 7269 6361
                  0d00
```

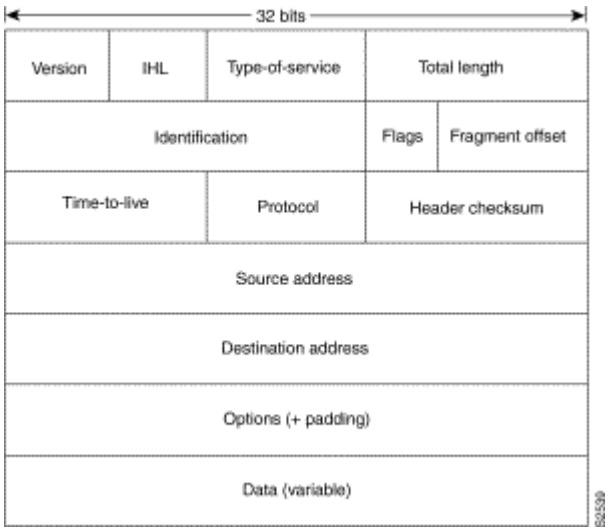
La prima riga è ovviamente un riassunto delle informazioni di header del pacchetto, decodificato da tcpdump. Segue il codice binario del pacchetto, scritto nel classico formato esadecimale (ogni carattere rappresenta perciò 4 bit, e ogni blocco di quattro caratteri 2 byte).

Per decodificare i dati, bisogna fare ricorso alle specifiche dei pacchetti dei protocolli della suite TCP/IP, che sono contenute negli appositi RFC:

- Protocollo IPv4: RFC 791 [97]
- Protocollo TCP: RFC 793 [98]
- Protocollo UDP: RFC 791 [99]
- Protocollo ICMP: RFC 792 [100]

Per comodità di riferimento riportiamo una descrizione ed uno schema degli header dei protocolli IP e TCP.

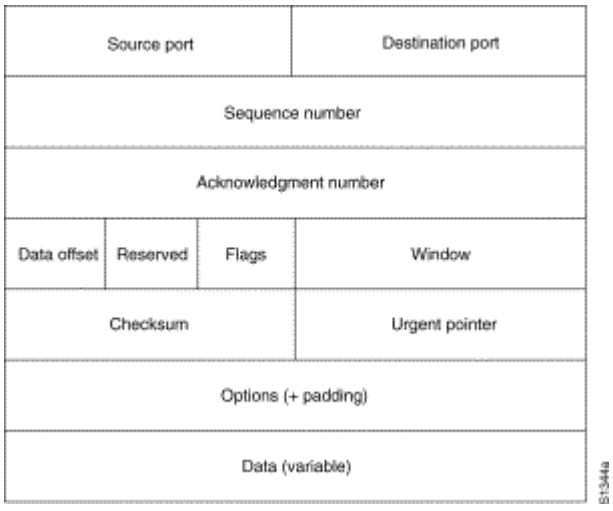
IP Protocol Header



- *Version*: la versione del protocollo in uso. Attualmente si usa IPv4, anche se già esiste la versione 6
- *IHL*: IP header length, la lunghezza dell'header IP, espressa in multipli di 32 bit;
- Type-of-service(TOS): ai bit di questo byte sono associati valori che consentono di gestire priorità differenziate o di richiedere determinate qualità di servizio per i pacchetti in transito. In generale queste richieste vengono onorate solo sulle reti interne, difficilmente sull'Internet globale.
- *Total length*: lunghezza totale del datagramma in ottetti (byte), incluso l'header;
- *Identification*: un valore univoco di 16 bit assegnato dal mittente;
- Campo *Flags*: comprendente i bit DF (Don't Fragment) e MF (More Fragments), che rispettivamente indicano che il pacchetto non può essere frammentato o che il pacchetto non è l'ultimo di una serie di frammenti
- *Fragment Offset*: spiazzamento (in multipli di 64 bit) rispetto al datagramma iniziale.

- *Time to Live*: indicazione del numero di hop (apparecchiature di rete attraversate) che sono concesse al pacchetto per raggiungere la destinazione. Quando questo valore arriva a zero il datagramma è scartato. Di solito è inizializzato a 32 o 64 (a valori minori nei vecchi sistemi), mentre il protocollo di controllo ICMP usa anche il valore massimo 255. Lo scopo è quello di eliminare eventuali pacchetti che stiano rimbalzando in un loop.
- *Protocol*: indica il protocollo di livello superiore che sta venendo trasportato da IP. Per ICMP questo valore è 1, per TCP è 6, per UDP 17.
- *Checksum*: una checksum di 16 bit del solo header. Dato che alcuni campi cambiano ad ogni hop (ad esempio TTL) questo campo è ricalcolato da ogni router.
- *Source Address*: è l'indirizzo IP del mittente del pacchetto
- *Destination Address*: è l'indirizzo IP del destinatario del pacchetto
- *Options*: il campo Opzioni ha lunghezza variabile (può addirittura mancare). Può contenere le seguenti opzioni:
    - *Sicurezza*: il datagramma può essere classificato secondo vari livelli conformemente agli standard militari;
    - *Source Routing*: il datagramma contiene un elenco di indirizzi IP che devono essere seguiti per arrivare a destinazione. In questa modalità il mittente impone l'instradamento;
    - *Record Route*: usata per registrare il percorso seguito dal datagramma;
    - *Timestamp*: usata dai vari router attraversati per registrare l'ora di attraversamento.

TCP Protocol Header



- *Source Port*: è un valore di 2 byte che identifica il processo mittente.
- *Destination Port*: un valore di 2 byte che identifica il processo destinatario.
- *Sequence Number*: è il numero sequenziale del primo byte di dati trasportato da questo segmento. Fa eccezione il caso del segmento di sincronizzazione (byte SYN posto a 1), in tal caso il numero è l'Initial Sequence Number (ISN) e il primo byte trasmesso ha numero ISN+1.

- *Acknowledgment Number*: numero del prossimo Sequence Number che il mittente del segmento si aspetta di ricevere.
- *Data Offset:* Campo di quattro bit che contiene il numero di parole da 32 bit che costituiscono l'header. Serve a individuare dove cominciano i dati.
- *Flags*: sono sei bit che rappresentano altrettante flag che consentono ai segmenti di trasportare oltre ai dati anche informazioni di controllo. Il loro significato è:
    1. URG: indica che il campo Urgent Pointer ha significato;
    2. ACK: indica che il campo Acknowledgment Number ha significato;
    3. PSH: Serve a realizzare la funzione di Push;
    4. RST: Reinizializza (Reset) la connessione;
    5. SYN: Serve a sincronizzare i numeri di sequenza;
    6. FIN: Il mittente non ha più dati da mandare.
- *Window*: è il numero di byte che si è pronti a ricevere.
- *Checksum*: la checksum è calcolata premettendo all'header TCP uno pseudo-header composto da 3 campi da 32 bit ricavati dall'header IP: 32 bit dell'indirizzo sorgente, 32 bit dell'indirizzo destinazione, 8 bit di zeri, 8 bit del campo protocollo, e 16 bit del campo TCP length, che è uguale alla TCP header length più la data length, espressa in ottetti. Si calcola quindi il checksum, come complemento a uno della somma dei complementi a uno di pseudo-header, header e dati, presi a blocchi di 16 bit, eventualmente aggiungendo zeri in fondo (padding). Per il calcolo incrementale dell'header si ricorre allle specifiche dell'RFC 1624 (che aggiorna l'RFC1141, che a sua volta aggiorna l'RFC 1071 che non è uno standard ma una implementazione suggerita).
- *Urgent Pointer*: sommato al Sequence Number punta al primo byte non urgente tra quelli del segmento dati.
- *Options*: l'unica opzione prevista dallo standard originale è un campo chiamato MSS (Maximum Segment Size) che indica la massima lunghezza del segmento che il destinatario accetta; questo campo può essere usato solo all'inizio della connessione (nei segmenti SYN). In seguito sono state definite altre opzioni.

5.4.4.2 Implementazione

Le funzioni built-in di Matlab ci consentono di scandire facilmente un testo alla ricerca di dati. Il problema nel caso specifico è che i pacchetti IP (come abbiamo sommariamente detto) hanno una lunghezza diversa a seconda del payload che portano, e ogni header può avere lunghezza variabile in base a svariate opzioni. Questa lunghezza può essere calcolata solo decodificando l'header del protocollo.

Di seguito riportiamo il codice che utilizziamo per la decodifica. È stato poi riutilizzato, con cambiamenti minori, in molti degli script descritti nei paragrafi successivi. Ricordiamo nuovamente che questo codice di esempio tratta i pacchetti TCP su protocollo IP, e che il protocollo UDP può essere implementato in maniera del tutto analoga. ICMP è addirittura più semplice, non includendo neppure un campo dati.

```
% elimino l'header scritto da TCP dump
% tanto interpreteremo noi il pacchetto.
header = fgetl(fid);

% leggiamo i primi 2 byte; dal momento che le cifre esadecimali sono
% a gruppi di 4 vengono visti come un numero solo
[a n] = fscanf(fid, '%x',1);

% controlliamo che sia ipv4 (primi quattro bit = 4)
if bitshift(bitand(a(1),61440),-12) == 4

% leggiamo i secondi 4 bit, per avere l'header length
ip_hdlen = bitshift(bitand(a,3840),-6);

% l'altro byte è il type-of-service
ip_tos = bitand(a,255);

% ora arriva la total length
total_len = fscanf(fid, '%x',1);

% con questo siamo in grado di leggere il resto del pacchetto:
rest_of_packet = fscanf (fid,'%x',ceil((total_len-4)/2));

% piccolo bugfix se il pacchetto avesse una lunghezza dispari
if mod((total_len-2),2) == 1
     rest_of_packet(length(rest_of_packet)) =
     bitshift(rest_of_packet(length(rest_of_packet)),8);
end

% riassembliamo il pacchetto in un'unica struttura per comodità
packet = [a; total_len; rest_of_packet];

% Ora calcoliamo tutti i campi dell'header
% unique ID del pacchetto
```

```
UID = packet(3);
% Don't Fragment Bit
DF = bitshift(bitand(packet(4),16384),-14);
% More Fragments Bit
MF = bitshift(bitand(packet(4),16384),-13);
% Fragment Offset
frag_offset = (bitand(packet(4),8191));
% Time to live
TTL = bitshift(packet(5), -8);
% Protocol field
PROTO = bitand(packet(5), 255);
% Packet Checksum
CHKS = packet(6);
% Indirizzo sorgente: quattro byte, ovvero due quartetti di cifre
SRC_ADD = bitshift(packet(7),16) + packet(8);
%Indirizo destinazione, idem
DST_ADD = bitshift(packet(9),16) + packet(10);

% Campo IP Options
ip_options = 0;
for i = 1:((ip_hdlen - 20)/2)
   ip_options = bitshift(ip_options,16) + packet(10+i);
end

% IF è un  pacchetto tcp
if PROTO == 6

% Calcoliamo l'offset a cui siamo arrivati
tcp_b = ip_hdlen / 2;

% Porta TCP sorgente
SRC_PRT = packet(tcp_b+1);

% Porta TCP destinazione
DST_PRT = packet(tcp_b+2);

% Sequence Number
SEQ = bitshift(packet(tcp_b+3),16) + packet(tcp_b+4);

% ACK Number
ACK = bitshift(packet(tcp_b+5),16) + packet(tcp_b+6);
```

```
% Lunghezza dell'header TCP
TCP_HLEN = bitshift(packet(tcp_b+7),-10);

% Urgent Bit
URG = bitshift(bitand(packet(tcp_b+7),32),-5);

% Acknowledge bit
ACK = bitshift(bitand(packet(tcp_b+7),16),-4);

%  Push Bit
PSH = bitshift(bitand(packet(tcp_b+7),8),-3);

%  Connection Reset bit
RST = bitshift(bitand(packet(tcp_b+7),4),-2);

%  Synchronize bit
SYN = bitshift(bitand(packet(tcp_b+7),2),-1);

%  Finalize bit
FIN = bitand(packet(tcp_b+7),1);

%  Window size
WND = packet(tcp_b+8);

% TCP header checksum
checksum = packet(tcp_b+9);

% Urgent Pointer
urgent_ptr = packet(tcp_b+10);

% TCP Options field
tcp_options = 0;
for i=1:ceil((TCP_HLEN - 20)/2);
   tcp_options = bitshift(tcp_options,16) + packet(tcp_b+10+i);
end

% Da qui iniziano i dati, quindi calcoliamo il punto di inizio:
data_start = (ip_hdlen + TCP_HLEN);

% Quest'ottimizzazione ci consente di preinizializzare l'array
```

```
% per ragioni di efficienza
% la massima dimensione è appunto 1460 byte (vedi RFC)
temp = zeros(1,1460);

% Leggiamo ciclicamente i dati del payload del pacchetto e li
% inseriamo in temp; la struttura strana con l'if serve a gestire
% pacchetti di lunghezza dispari
for data_p = 1:(total_len-data_start)
 if mod(data_start+data_p,2) == 1
   temp(1,data_p) = (bitshift(packet((data_start+data_p+1)/2), -8));
 else
   temp(1,data_p) = bitand(packet((data_start+data_p)/2), 255);
 end
end % END del FOR di decodifica pacchetto
end % END dell'if del pacchetto TCP
end % END dell'if del pacchetto Ipv4
```

5.4.5 ALGORITMO S.O.M.

5.4.5.1 L'algoritmo

Come è già stato spiegato, una S.O.M. è una rete neurale utilizzata per un apprendimento non supervisionato, che costruisce un insieme di nodi di dimensione prefissata, collegati tra loro secondo uno schema deciso dall'implementatore (solitamente rettangolare o esagonale), che hanno una loro posizione in uno "spazio dei nodi" solitamente bidimensionale (per questo viene chiamata "mappa"). Ogni nodo rappresenta una classe, e corrisponde pertanto ad un punto nello spazio n-dimensionale dei vettori di ingresso.

L'addestramento di una S.O.M. avviene iterativamente seguendo questa procedura:

1. Creiamo le strutture della mappa. Diciamo che ci siano k neuroni. Inizializziamo i "pesi" dei neuroni (che descrivono le loro caratteristiche nello spazio dei dati) con dei valori casuali, e chiamiamo questi vettori w_i, mentre chiameremo n_i le loro coordinate nello spazio dei nodi.
2. Per ciascuna epoca e, per ciascuno dei vettori di addestramento:
   a. Troviamo il nodo che meglio lo approssima (ad esempio, b). Se indichiamo con "dist" la funzione distanza utilizzata, ovviamente è il centroide tale che $\forall 0 < i < k, dist(w_i, x_j) > dist(w_b, x_j)$

b. Da questo nodo si determina un raggio di nodi attivati $N_b(e) = \{w_i \mid 0 < i < k, dist\_n(n_i, n_b) < R(e)\}$, dove il "raggio" è una funzione monotona decrescente dell'epoca di addestramento corrente e, mentre la distanza va intesa nello spazio dei neuroni e non in quello dei dati; solitamente la funzione dist_n è una distanza euclidea, ma potrebbe essere di tipo "manhattan distance", o altro.

c. Il centroide più vicino e i nodi attivati vengono aggiornati, gli altri non variano:

$$w_i(e) = \begin{cases} w_i(e-1) + \alpha(e)H(e)[x_j - w_i(e-1)], w_i \in N_b(e) \\ w_i(e-1), w_i \notin N_b(e) \end{cases}$$

La regola di addestramento è detta regola di Kohonen, e i termini che vi compaiono sono $\alpha(e)$, che è un coefficiente di addestramento monotonamente decrescente secondo l'epoca e compreso tra 0 e 1, e $H(e) = e^{-\frac{d^2}{4R(e)^2}}$, dove d rappresenta la distanza tra il neurone considerato e il neurone vincente, calcolata con le avvertenze descritte al punto b.

3. Il procedimento viene ripetuto per un numero selezionato di epoche

L'algoritmo necessita, come si è detto, di un numero di epoche predeterminato, in quanto la convergenza non è assicurata (come si è detto, è possibile dimostrare che non esiste una funzione obiettivo per la S.O.M., ovvero che le regole di addestramento utilizzate non sono il "gradiente" di nessuna funzione). Il criterio di convergenza è dunque soggettivo, o per meglio dire va determinato in base al problema e al buon funzionamento dell'algoritmo nel problema. Alcuni articoli presenti in rete documentano che, per insiemi di dati con ampie dimensioni, la convergenza della rete (intesa come numero di epoche di training) potrebbe essere molto lenta; tuttavia mostreremo nel seguito come la crescita del tempo richiesto per l'addestramento sia lineare rispetto al numero di epoche. Un altro problema da noi già sottolineato è che il numero di nodi va predeterminato (ma si veda sotto per una possibile soluzione di tale problema).

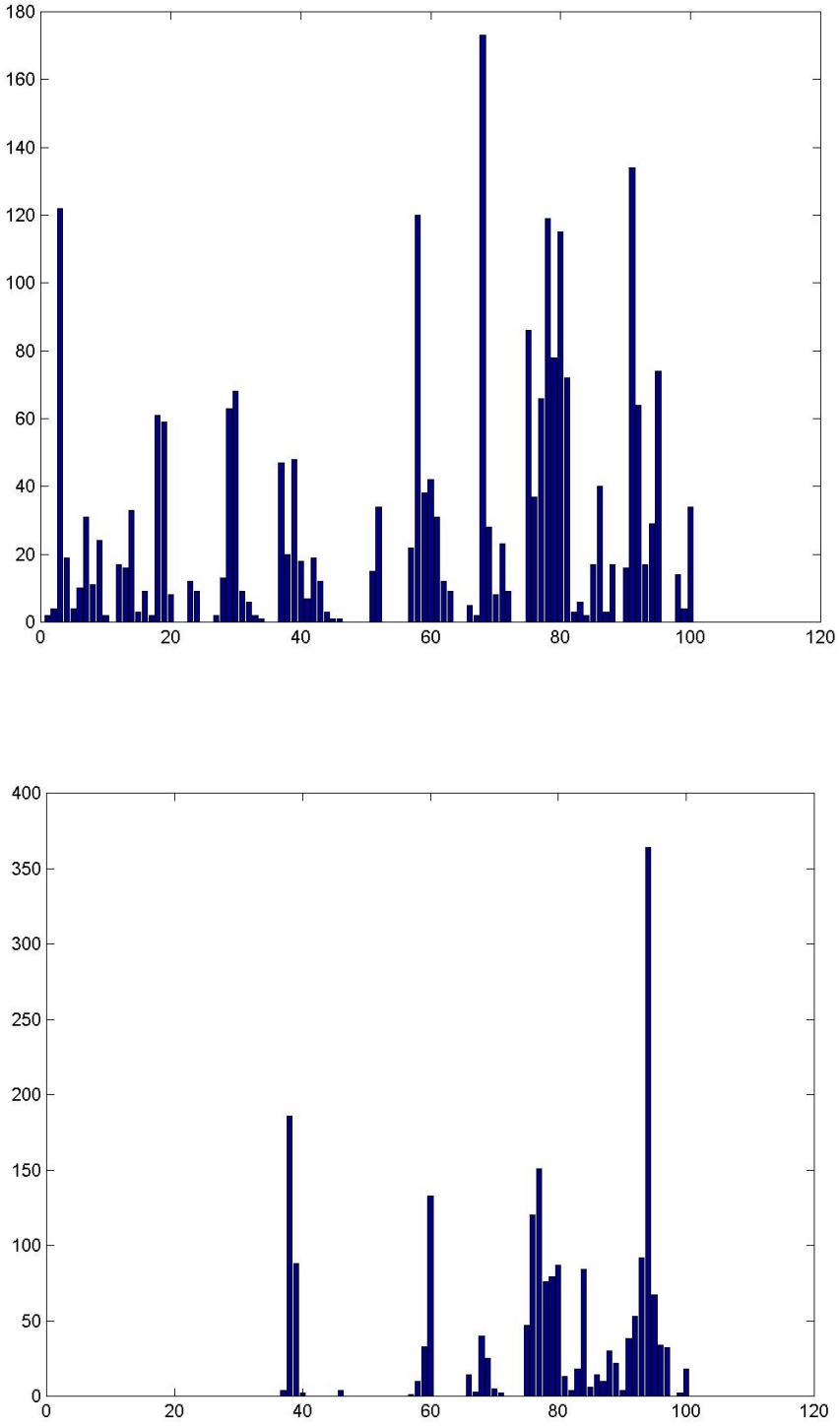
5.4.5.2 Implementazione e applicazione ai dati

Abbiamo utilizzato come implementazione di riferimento quella contenuta nel "Neural Network Toolbox" di Matlab, la stessa utilizzata per l'esperimento preliminare sui log di Apache. Abbiamo provato ad addestrare la rete con varie forme e numero di epoche, e abbiamo ottenuto risultati di qualità variabile. Ovviamente, riportiamo qui i risultati migliori, ed i parametri utilizzati per ottenerli. Dal momento

che, come si dirà, la numerosità del campione di addestramento è risultata influire esponenzialmente sul tempo impiegato, abbiamo anche fatto ricorso ad un piccolo "trucco". Dal momento che (sperimentalmente) molti payload risultano essere vuoti, abbiamo deciso di "pre-filtrarli", cioè di suddividerli manualmente in una categoria a parte. Se da un lato questo potrebbe teoricamente falsare in qualche modo l'approccio di apprendimento (in quanto, lo ricordiamo, i neuroni di una S.O.M. sono "legati" l'uno all'altro), dall'altro i nostri esperimenti hanno mostrato risultati coerenti in entrambi i casi, inducendoci a scegliere un approccio che ci consentisse di mostrare "più campioni" alla rete, a parità di tempo di addestramento.

Per effettuare un esperimento, abbiamo poi utilizzato un software di scansione di vulnerabilità, Nessus. Questo tipo di programma contiene un "campionario" di test per vulnerabilità note dei sistemi operativi e dei server di rete; un aggressore di media esperienza potrebbe utilizzare questo programma, o qualcosa di molto simile, per valutare i possibili punti di ingresso in un sistema. Abbiamo analizzato il traffico di Nessus e lo abbiamo sottoposto a una S.O.M. ad unità "rettangolari", 10 x 10, addestrata su un profilo di traffico "normale" di rete per 10000 epoche. Abbiamo quindi confrontato tale traffico categorizzato con un eguale volume di traffico proveniente da un utilizzo "normale" di una rete. Riportiamo alla pagina seguente i due profili della suddivisione del traffico. Spendiamo qualche parola per commentarli, visto che li utilizzeremo spesso nel seguito. In ascissa sono riportate le 100 classi in cui il traffico è stato suddiviso. In ordinata, il numero di campioni di traffico che ricadono in ciascuna classe. Da un rapido confronto si può facilmente notare il diverso "spettro", per così dire, della suddivisione. Evidentemente, il secondo stadio dell'algoritmo dovrà essere progettato in modo da rilevare ciò che "ad occhio" risulta evidente.

È anche necessario notare che una scansione mediante Nessus è sicuramente da considerarsi un attacco fragoroso, l'equivalente dell'approccio di un ladro che demolisca una porta con la dinamite. Gli aggressori che ci interessa veramente rilevare, quelli più pericolosi, non utilizzano sistemi così barbari, ma approcci molto più cauti. Purtroppo, per verificare l'applicabilità e l'efficacia del nostro approccio a tali situazioni, dovremmo disporre del sistema completo di entrambi gli stadi. Tuttavia, nel capitolo dedicato all'approccio concettuale al secondo stadio, spiegheremo quali sono le ragioni intuitive che ci inducono ad essere fiduciosi nell'efficacia dell'architettura da noi progettata.



Campione di traffico di una rete, in condizioni di utilizzo normali (sopra)
e durante una scansione effettuata con Nessus (sotto)

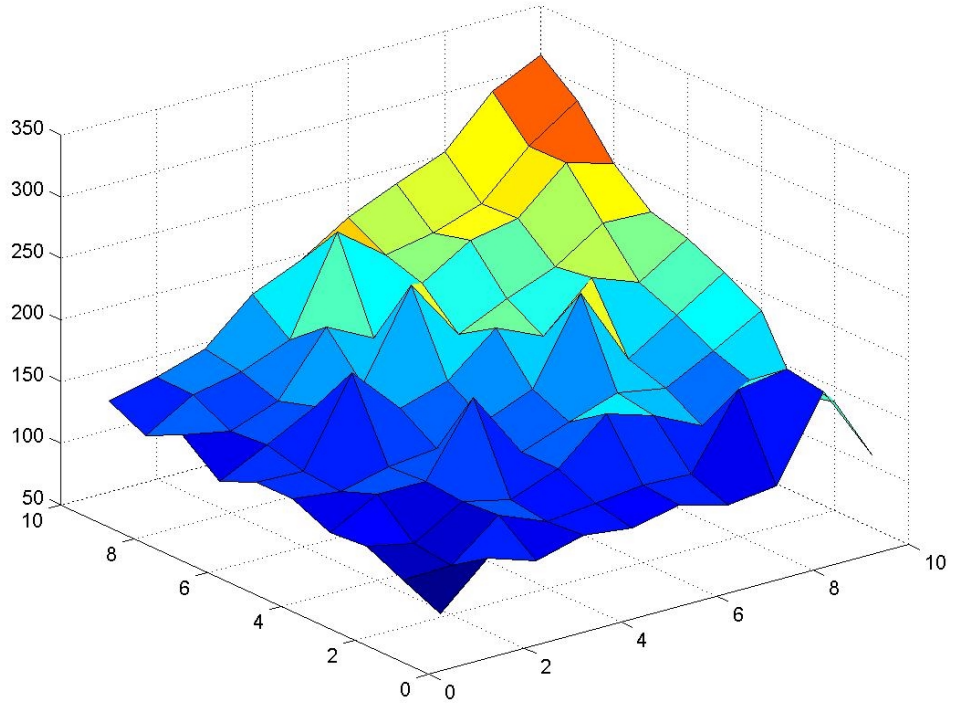
Come ulteriore verifica, abbiamo ispezionato a mano le classificazioni prodotte dalla S.O.M., rilevando alcune caratteristiche interessanti: per esempio, la divisione quasi perfetta del traffico telnet in ingresso da tutti gli altri tipi di traffico; la presenza di cluster quasi esclusivamente composti da messaggi di e-mail; la presenza di alcuni cluster contenenti esclusivamente comandi http ed FTP (la struttura di questi comandi è, effettivamente, molto simile).

Ovviamente, in ciascuno di questi casi si sono evidenziati degli "errori" di categorizzazione, che sono tuttavia del tutto attesi in un algoritmo di apprendimento non supervisionato !

5.4.5.3 Considerazioni sull'efficienza

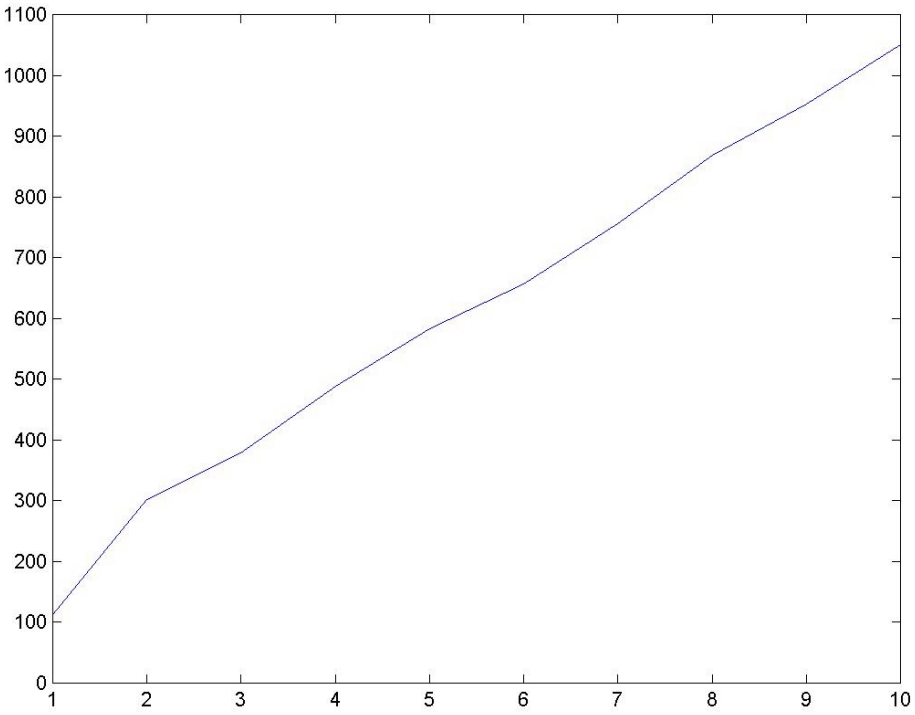
Per quanto riguarda le considerazioni di efficienza dell'algoritmo, premettiamo che molti autori suggeriscono l'utilizzo di un algoritmo di preprocessing che elimini le dimensioni ridondanti dei dati (per esempio mediante la Principal Component Analysis) e li normalizzi. Se il secondo problema è relativo (i nostri dati sono tutti su intervalli uniformi), per il primo problema abbiamo già espresso le motivazioni della nostra contrarietà all'interno del paragrafo dedicato agli algoritmi di feature selection e riduzione dimensionale.

Per verificare la crescita di complessità computazionale dell'algoritmo in dipendenza a vari parametri di dimensionamento, abbiamo ritenuto opportuno svolgere svariati test di cui riportiamo alcuni grafici significativi. Il primo riguarda la crescita del tempo di addestramento della rete, a parità di altri parametri, con le dimensioni. Ricordiamo che questi sono tutti valori sperimentali, pertanto piccole oscillazioni nell'andamento possono essere presenti. Si può notare come (logicamente) il tempo cresca dipendendo sostanzialmente dal prodotto delle due dimensioni, quindi dal numero totale dei nodi.



Crescita del tempo di addestramento in dipendenza dal numero di nodi. Sugli assi x e y sono riportati i nodi, da 1 a 10, per ciascuna dimensione. Sull'asse z il tempo totale di addestramento in secondi

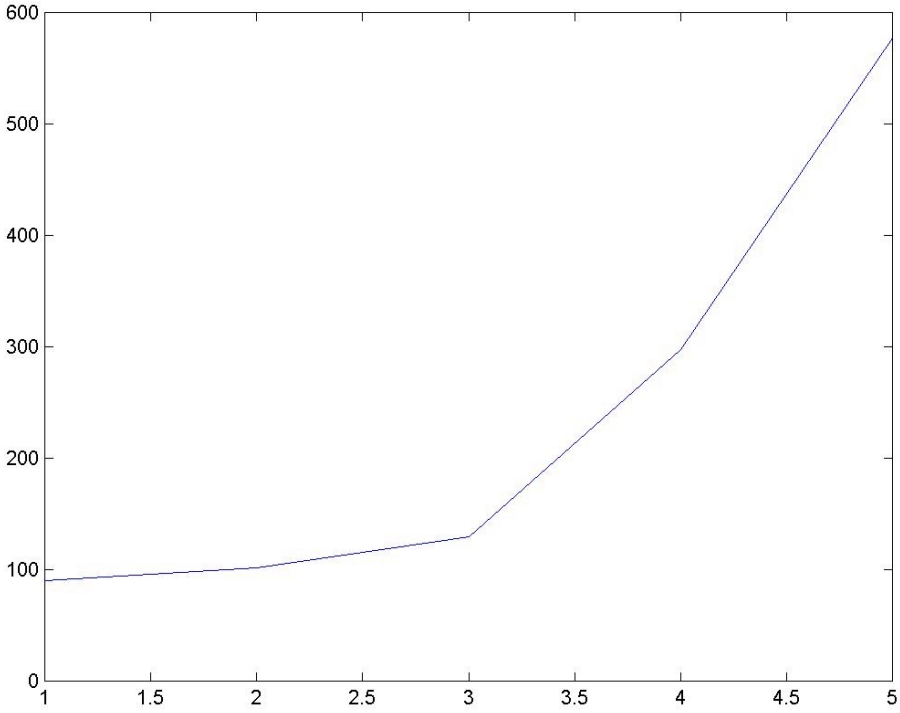
Un secondo test ha riguardato la dipendenza del tempo d'addestramento dal numero di epoche, nella speranza che ci fosse una progressiva velocizzazione dovuta alla fase di setup iniziale e ad eventuali ottimizzazioni. Così non è stato, e il tempo d'addestramento è risultato essere esattamente lineare nel numero di epoche. È molto probabile che questo risultato possa essere migliorato da una buona implementazione ottimizzata dell'algoritmo.



Salita del tempo di addestramento di una S.O.M. (in ordinata, in secondi) in dipendenza dal numero di epoche (in ascissa, in migliaia)

Il test che ci ha tristemente sorpreso è stato quello relativo alla dipendenza dalla dimensione del training set. Originariamente, la nostra intuizione (derivante dall'algoritmo di addestramento "banale" che abbiamo proposto) era che questa dipendenza dovesse essere lineare, in quanto in tale algoritmo il numero complessivo di operazioni dipende dal prodotto tra epoche e numerosità del training set.

Tuttavia, le nostre prove hanno indicato un comportamento ben diverso: gli algoritmi "batch" usati da Matlab per velocizzare l'addestramento – probabilmente a causa delle crescenti esigenze di memoria, con un incremento dell'I/O su disco per lo swap – fanno sì che essa diventi, rapidamente, esponenziale, creando di fatto una barriera computazionale per la numerosità del campione presentato alla rete.



Crescita anomala del tempo di addestramento di una S.O.M. (in ordinata, in secondi) in dipendenza dalla numerosità dell'insieme di addestramento (in ascissa, in migliaia)

Dal momento che questa crescita non ci sembrava assolutamente logica, abbiamo provato a ripetere lo stesso test con dati di dimensionalità inferiore, ottenendo come risultato una crescita perfettamente lineare. Ciò ci ha indotto a ipotizzare che il problema dipendesse dai requisiti di memoria dell'algoritmo di tipo batch, e dall'aumento delle richieste di swap necessarie per soddisfarle.

Abbiamo dunque studiato l'andamento delle prestazioni in dipendenza congiunta dalle dimensioni dei dati, e dalla numerosità del training set, ottenendo il risultato sperimentale illustrato dalla figura a pagina seguente.

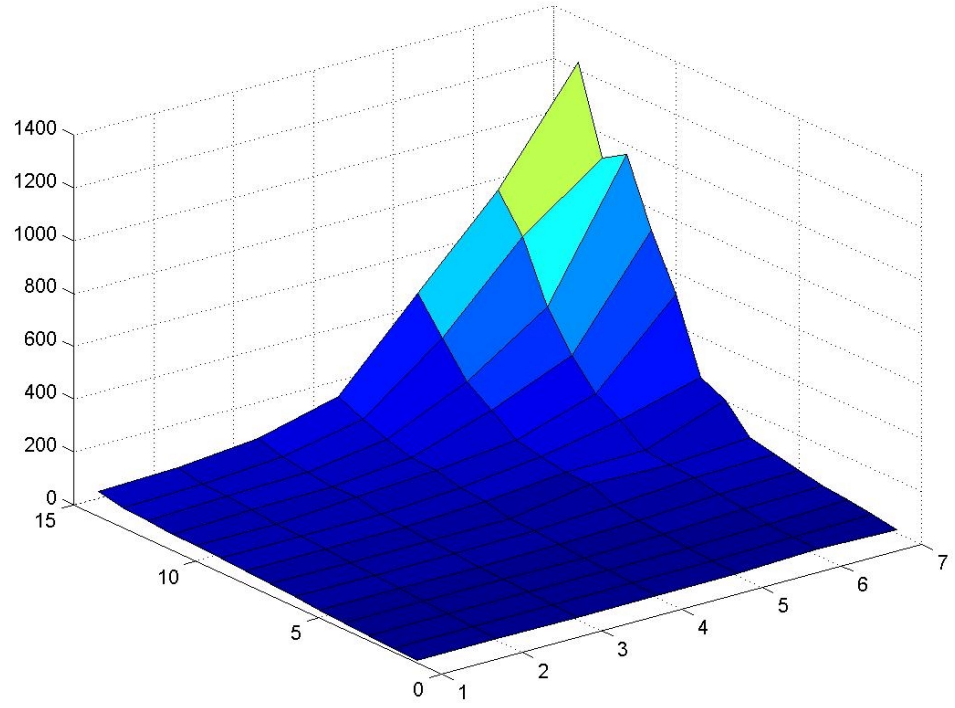


Grafico che presenta l'aumento del tempo di addestramento (asse z, in secondi) in dipendenza dal numero di dati (asse x, in migliaia) e dal numero di dimensioni (asse y, in centinaia)

La nostra conclusione è che l'algoritmo di addestramento "batch" delle S.O.M., così come implementato dal toolbox di Matlab, ottimizza sì il tempo di addestramento per dati di bassa dimensionalità (infatti, nel grafico, rimanerndo al di qua delle 1000 dimensioni la crescita in dipendenza del numero di elementi di addestramento è non solo lineare, ma anche limitatissima), mentre ad altissima dimensionalità come nel nostro caso incontra dei notevoli problemi (probabilmente dovuti all'esaurimento della RAM) e fa dipendere in maniera sostanzialmente esponenziale il tempo di addestramento dalla numerosità del training set. È nostra opinione che questo problema possa essere risolto mediante una attenta implementazione degli algoritmi, specificamente pensata per un problema ad alta dimensionalità.

Vi sono ricerche ed algoritmi che sembrano indicare la fattibilità di un addestramento "on line" delle S.O.M., tuttavia la pesantezza dell'algoritmo di addestramento unita alla sperimentalità delle ricerche non ci danno grandissime aspettative in proposito.

Intuitivamente (ma il concetto è già verificato anche in letteratura [101]) se si determina la distanza tra un vettore ed il neurone con cui la rete lo associa (BMU, Best Matching Unit), diventa possibile individuare alcuni outlier grazie al fatto che il centroide della classe in cui la rete lo ha classificato, a logica, risulta "più distante" dall'outlier che da tutti gli altri vettori di input. Altri outlier possono essere individuati, per esempio, dal fatto che cadono in una classe fino a quel momento vuota. Pertanto, all'algoritmo di secondo stadio potrebbe essere utile restituire due valori: una classificazione del payload, e una misura della distanza dal neurone di classificazione, di modo che l'algoritmo di secondo livello possa non solo trovare correlazioni di "stranezza" rispetto alla categoria, ma anche avere un input su quanto aderente alla categoria di classificazione sia il vettore. L'unico problema a questo proposito potrebbe essere quello di "scalare" questo valore di distanza.

Nell'esposizione teorica abbiamo anche citato l'algoritmo *Growing Hierarchical SOM (GHSOM)*, che cerca di risolvere il problema della struttura e delle dimensioni della mappa consentendole di crescere in larghezza (aggiungendo nuove unità alla rete durante l'addestramento, nelle aree dove vengono a mancare dei nodi) ed anche in profondità (addestrando una nuova mappa di livello gerarchico diverso nelle unità che rappresentano grossi cluster). La struttura flessibile e gerarchica dell'algoritmo ci ha affascinato, ed abbiamo deciso di effettuare qualche test preliminare anche su questo tipo di mappa. Per farlo abbiamo utilizzato un toolbox [102] disponibile liberamente su Internet.

Come immaginavamo, la flessibilità ha un prezzo elevato in termini di tempo d'addestramento, specialmente nelle chiamate ricorsive di crescita. Anche in questo caso il tempo cresce in modo inaccettabile con l'aumento del numero di vettori di esempio.

Per i motivi già esposti, abbiamo deciso di non sperimentare le *Recurrent SOM (RSOM)*, onde riservare le correlazioni temporali al punto dove esse sono veramente significative, ovvero al secondo stadio del nostro IDS.

Grazie ad algoritmi come LabelSOM [103] è anche possibile associare ai nodi della mappa dei campioni rappresentativi (etichette) che possono essere usati per esplorare più comodamente le suddivisioni. Questi algoritmi sono nati per risolvere il problema del labeling di larghe mappe con poca conoscenza di dominio. L'algoritmo determina quali "caratteristiche" di un vettore sono più "influenti" nel farlo ricadere in un certo sottoinsieme della mappa.

5.4.6 ALGORITMO PRINCIPAL DIRECTION DIVISIVE PARTITIONING

5.4.6.1 L'algoritmo

Abbiamo già descritto a grandi linee il funzionamento di questo algoritmo, che è gerarchico e divisivo. L'algoritmo comincia considerando un cluster formato da tutti i punti. Di questo cluster viene calcolato il centroide, ed inoltre viene calcolata la decomposizione a valori singolari della matrice degli elementi del cluster (SVD), di cui si prende in considerazione il primo valore e il relativo vettore associato, che è appunto detto *"direzione principale"*.

Il centroide viene proiettato sul vettore, e il cluster viene diviso in due parti, con un iperpiano perpendicolare al vettore e passante per la proiezione del centroide. Il cluster originario diventa un nodo padre, con le due metà come figli. Nel nodo viene salvato un criterio di classificazione molto semplice (costituito da una disuguaglianza). L'interpretabilità umana del criterio, in spazi a grandi dimensioni, è tuttavia minima.

In codice Matlab, questo è il passo ricorsivo dell'algoritmo:

```
function [rl,ll,scrl,scll, u, w] = PDP(input)

%Principal Direction Partitioning splitting
%Input:
%  input - Matrice in input
%
%Outputs
%  rl   - Partizione destra dei dati
%  ll   - Partizione sinistra
%  scrl - valore di dispersione destro
%  scll - valore di dispersione sinistro
%  u    - centroide del cluster
%  w    - la direzione principale utilizzata

% la notazione che usiamo è quella del paper
% di Boley - Borst - Gini [104]

% Calcoliamo le dimensioni della matrice
[n,k] = size(input);
```

```
% Calcoliamo il centroide
w = mean(input, 2);

% Calcoliamo la matrice A
A = input - w * ones(k,1)';

% Calcoliamo il valore singolare più grande
% e l'autovettore corrispondente
[u,l,v] = lansvd(A,1);

% Il discriminante sarebbe u'*A, ma calcoliamo (v * l)'
% per velocizzare
vc = (v * l)';

% Applichiamo il discriminante
ll = input(:, find(vc<=0));
rl = input(:, find(vc>0));

% Sui risultanti figli calcoliamo la dispersione
scll = max(std(ll,1,2))*size(ll,2);
scrl = max(std(rl,1,2))*size(rl,2);
```

A questo punto l'algoritmo si ripete ricorsivamente, sul più disperso degli insiemi foglia, fino al raggiungimento di un obiettivo che può essere un numero massimo di classi, oppure una soglia massima di dispersione prefissata.

5.4.6.2 Implementazione e applicazione ai dati

La struttura dati ad albero richiesta da un algoritmo di questo tipo non è semplicemente implementabile in MATLAB, a causa dell'assenza del concetto di "puntatore" nel linguaggio. Vista l'assenza di strutture adatte, abbiamo deciso di implementare l'albero utilizzando un cell array, che contiene vettori per i nodi e le matrici di dati come foglie. Ne diamo qui di seguito una definizione in stile BNF:

```
Tree = { <tree_element>* }
<tree_element> = <node> | <leaf>
<node> = { <left_tree_element_id> <left_scatter> <left_isleaf>
<right_tree_element_id> <right_scatter> <right_isleaf> [<centroid>]
[<direction>] }
```

```
<leaf>  = matrice 1460 x n con n vettori classificati
[<centroid>] = centroide della suddivisione
[<direction>] = direzione principale della suddivisione
<left_tree_element_id> = indice, all'interno dell'albero, del
<tree_element> contenente la foglia sinistra
<left_scatter> = dispersione del ramo a sinistra
<left_isleaf> = vale 1 se il ramo a sinistra è una foglia in realtà
<right_tree_element_id>, <right_scatter>, <right_isleaf> =
simmetricamente, come sopra
```

Non riportiamo il codice sorgente dell'algoritmo di addestramento, che in pratica esegue i passaggi già delineati: sceglie la foglia con il massimo di dispersione discendendo l'albero e sfruttando gli scatter precalcolati, esegue la suddivisione e crea un nuovo nodo con due foglie. Dopodichè calcola il massimo di scatter nel nuovo nodo (che è necessariamente minore del precedente) e aggiorna, risalendo, i valori di scatter nei rami superiori.

Perché abbiamo bisogno di questo valore, e come viene calcolato ? Come abbiamo ripetutamente accennato, la selezione del cluster che andrà ricorsivamente suddiviso è un sottoproblema non banale che l'algoritmo in sé lascia in sospeso. Per il criterio con cui selezionare il cluster da suddividere esistono vari approcci [105]. Un primo approccio possibile è quello di portare la suddivisione fino all'estremo, producendo un albero binario le cui foglie siano singoli vettori; un secondo approccio suggerisce di dividere i cluster col maggior numero di elementi; infine un terzo approccio cerca di dare una definizione di "dispersione" (che intuitivamente è per esempio la varianza rispetto al centroide). Scartando il primo approccio per motivi di tempo di elaborazione, e per la necessità di avere un numero definito di classi, oppure in alternativa un criterio che ci dica quante classi servono per descrivere i dati, ci restano due alternative.

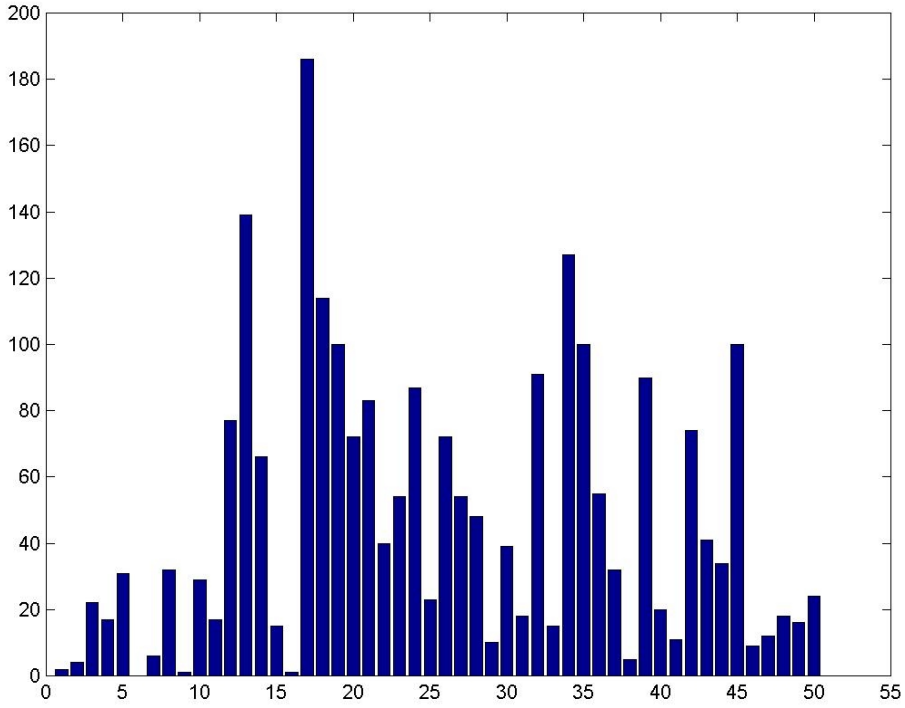
Nel tipo di dataset in questione abbiamo osservato che molto facilmente si creano grossi cluster di dati simili con piccole varianze, che verrebbero partizionati solo per la forza dei numeri. Abbiamo reputato pertanto più interessante cercare di determinare un criterio di dispersione simile al terzo, che sembra un po' "rozzo" ma nel complesso funzionale. Resta problematico però stabilire, per il nostro "tipo" di dati, una soglia massima di dispersione, quindi ci ritroviamo a dover prefissare un numero di classi come criterio di stop dell'algoritmo.

Un altro problema rilevato a questo punto è stato però che nel dataset di esempio (ma crediamo, a buona ragione, in qualsiasi dataset di questo tipo) esistono dei casi particolari in cui vettori molto diversi dagli altri ma anche diversi tra loro sbilanciano

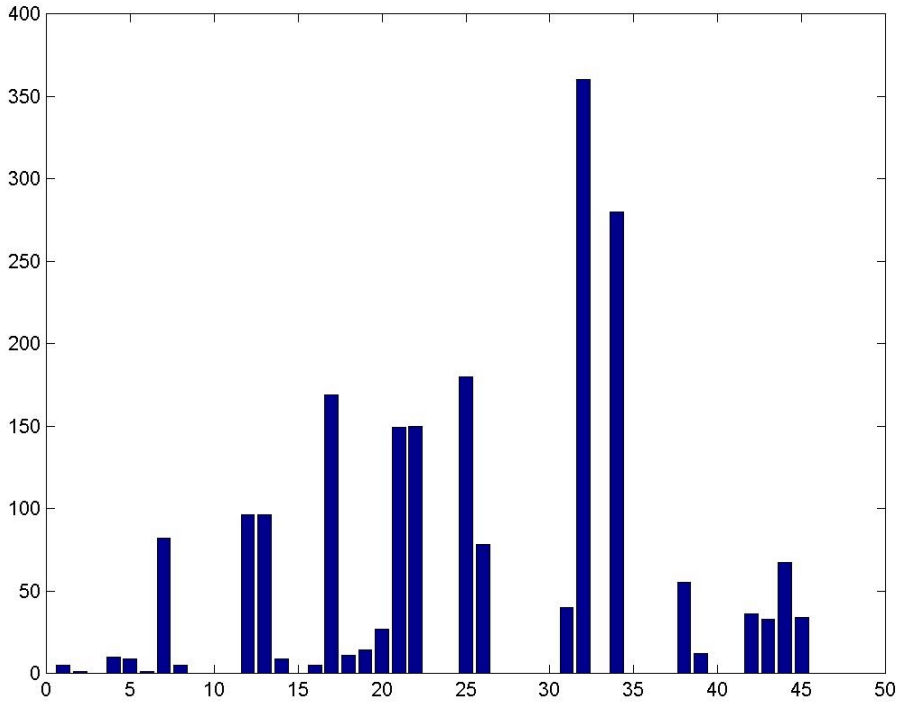
l'albero, in pratica risultando avere sempre una varianza molto elevata e impedendo una suddivisione corretta tra le classi (i.e. creando molte classi foglia con un singolo vettore contenuto).

Questo tipo di problema dipende evidentemente dalla misura "ingenua" assunta per lo scatter, ovvero la varianza, pertanto abbiamo deciso di complementarla con un coefficiente che rappresentasse la numerosità del campione.

Il sistema così "corretto" ha funzionato in modo sorprendentemente efficace. Successivi test hanno prodotto risultati corretti e coerenti, peraltro molto simili a quelli ottenuti con gli altri algoritmi. Ecco ripetuto, a titolo d'esempio, il raffronto tra traffico normale e traffico generato dallo scanner Nessus, suddivisi qui tra 50 classi (rispetto alle 100 utilizzate per la S.O.M.). Anche in questo caso abbiamo escluso a priori gli zeri.



Suddivisione di circa 2000 pacchetti di traffico normale, mediante un albero generato da PDDP



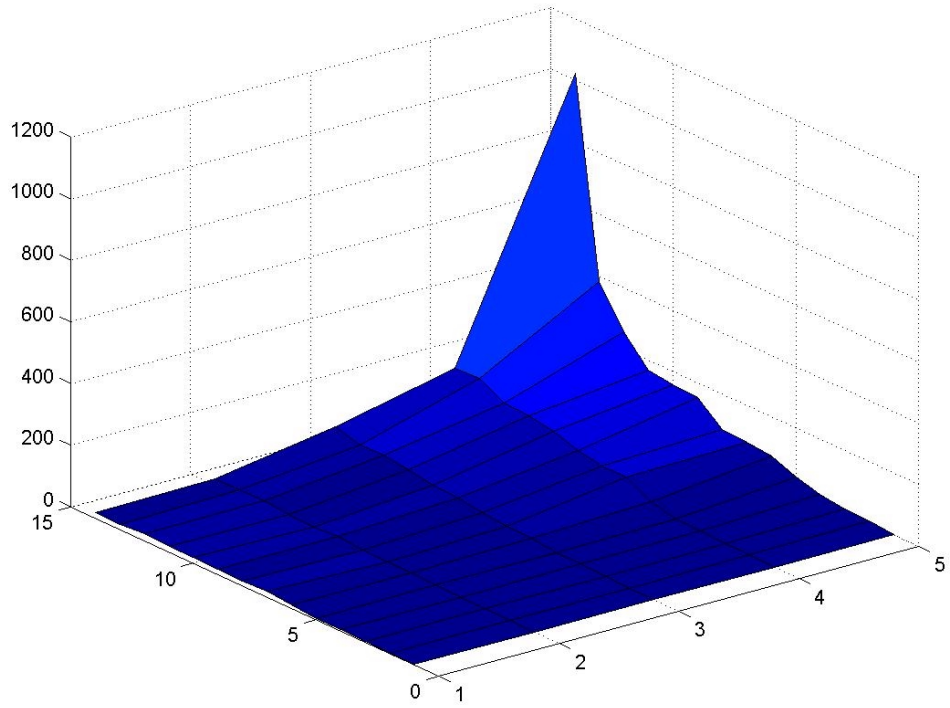
Suddivisione di circa 2000 pacchetti di traffico generato dallo scanner Nessus, mediante un albero generato da PDDP

L'ispezione manuale dei cluster generati conferma, in gran parte, gli stessi risultati ottenuti con la Self Organizing Map, anche se in questo caso la nostra percezione è che in alcuni casi la suddivisione sia qualitativamente inferiore. Però questo giudizio "soggettivo" andrebbe confrontato, a posteriori, con i risultati ottenuti ponendo in cascata i due stadi completi.

5.4.6.3 Considerazioni sull'efficienza

Parlando di efficienza il quadro è complicato. I nostri primi test sono stati fatti con l'implementazione della Singular Value Decomposition offerta da Matlab, e i risultati non ci hanno incoraggiati. L'algoritmo normalmente utilizzato per il calcolo della decomposizione a valore singolare infatti ha una complessità temporale pari a $O(pq^2 + p^2q + q^3)$, con p e q dimensioni della matrice [106]. Una implementazione [107] altamente ottimizzata dell'algoritmo di Lanczos (costruita mediante la bidiagonalizzazione con reortogonalizzazione parziale) ha consentito di superare molte di queste difficoltà, offrendo una complessità pari a $O(pqr^2)$, con r rango della matrice e quindi minore o uguale al minimo tra p e q.

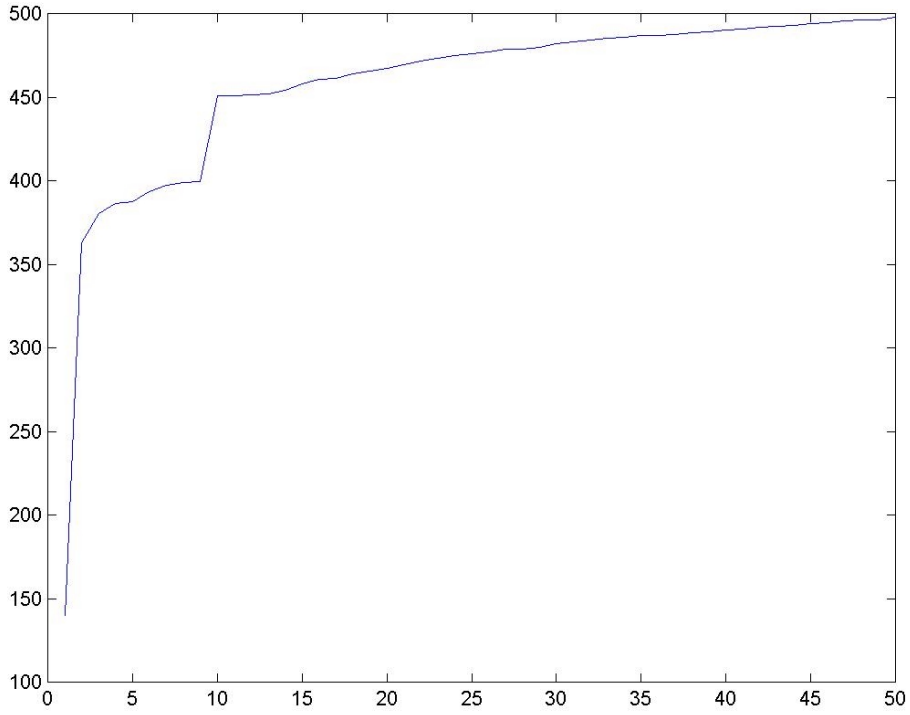
Anche con questo algoritmo ottimizzato, la decomposizione per valori singolari ha funzionato con matrici contenenti un massimo di 3-4000 vettori campione, dopodichè l'algoritmo viene messo in crisi. Memori dell'esperienza fatta con le S.O.M. abbiamo fatto degli esperimenti, variando il numero di campioni e la loro dimensionalità. Il risultato viene illustrato in figura.



Crescita del tempo di elaborazione per effettuare un passo divisivo con l'algoritmo PDDP, in base al numero dei dati (asse x, in migliaia) e alla dimensionalità (asse y, in centinaia)

Riteniamo che anche in questo caso ci sia lo stesso tipo di problema, con un picco in corrispondenza del limite di spazio in memoria. Questo rende l'algoritmo di addestramento sostanzialmente equivalente, per pesantezza, all'algoritmo S.O.M. nell'implementazione batch data da Matlab, ma probabilmente più pesante di una buona implementazione ottimizzata dello stesso. Si ricordi anche che il costo computazionale qui indicato è relativo ad un singolo passo della divisione, mentre nel grafico relativo alle S.O.M. veniva calcolato il costo dell'algoritmo di addestramento completo.

Abbiamo fatto un test anche per determinare il comportamento medio dell'algoritmo mano a mano che discende nell'albero di divisione. Anche se è ovvio che il caso pessimo vorrebbe tutte le divisioni ugualmente costose (nell'ipotesi che a ogni divisione da un blocco di n campioni si creino una foglia da n-1 campioni ed una a campione singolo), vorremmo se possibile avere una indicazione di comportamento effettivo dell'algoritmo, per capire di quanto influisca l'aumento del numero di classi in cui dividere i dati. Come si può capire anche intuitivamente, tutti gli algoritmi di tipo gerarchico e divisivo implicano una curiosa "inversione di costo", per cui la barriera computazionale è, in realtà, la suddivisione iniziale del dataset. Il grafico in questa pagina illustra un andamento possibile.



Costo computazionale dell'algoritmo PDDP: in ascissa il numero di passi, in ordinata il tempo

È abbastanza evidente l'andamento grossomodo logaritmico del tempo complessivo rispetto ai passi. Gradini come quello evidenziato in figura significano lo spostamento da cluster ad alta dispersione e numerosità relativamente bassa a un cluster più numeroso, precedentemente trascurato perché meno disperso.

È da notare che esiste da poco in letteratura [108] una implementazione di un diverso algoritmo, con ordine di complessità temporale pari a $O(pqr)$ e spaziale $O((p+q)r)$, con la possibilità di una applicazione incrementale dell'algoritmo al sopraggiungere di nuovi dati. Sarebbe probabilmente interessante provare ad applicare tale nuova formulazione al problema ed analizzare eventuali significative differenze.

5.4.7 ALGORITMO K-MEANS

5.4.7.1 L'algoritmo

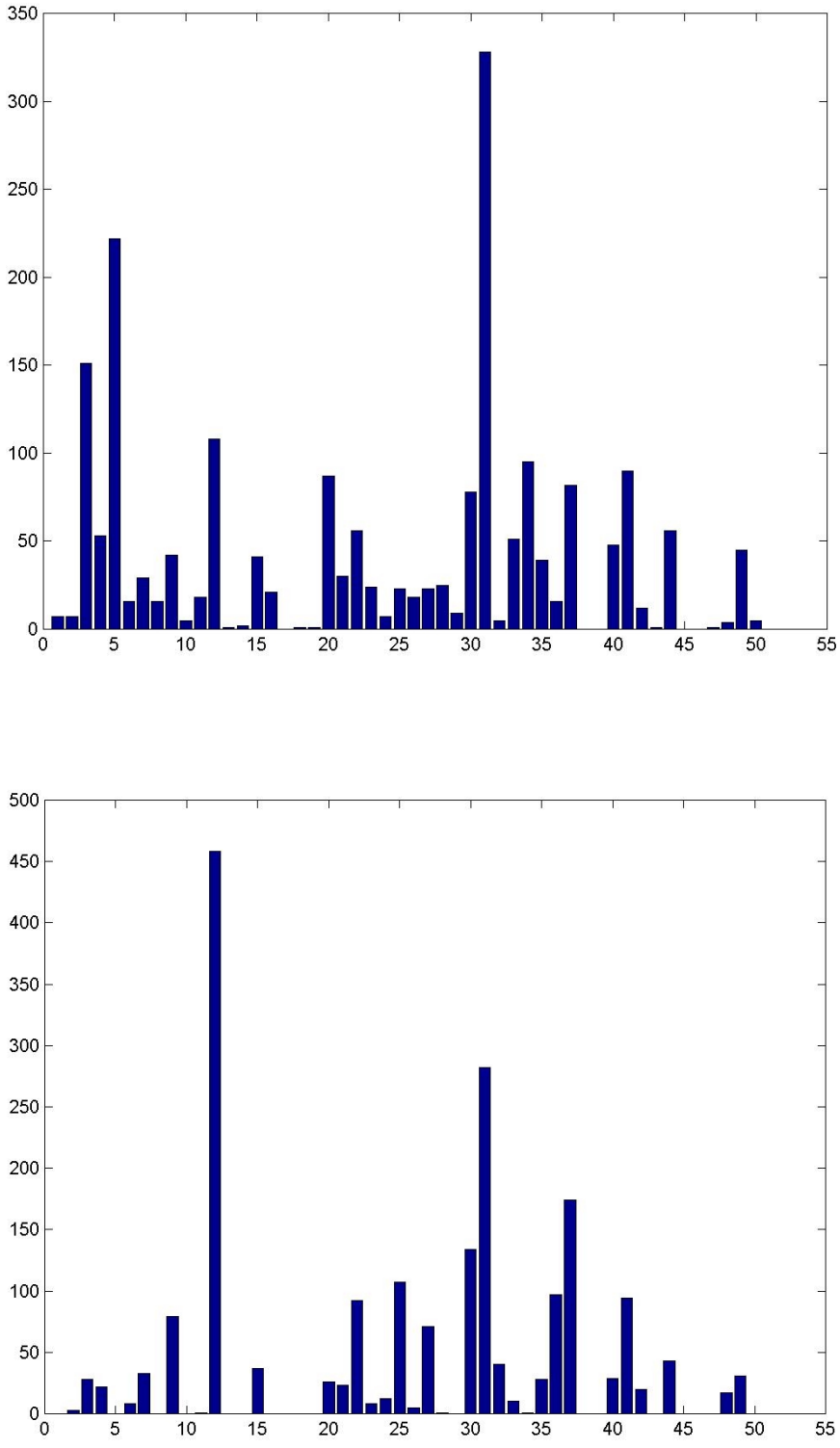
L'algoritmo K-means è probabilmente il più semplice tra tutti quelli presi in considerazione. L'algoritmo inizia selezionando a caso un certo numero K di centroidi. Per ogni iterazione vengono svolte le seguenti operazioni:

1. Ogni vettore viene assegnato al centroide più vicino, formando K cluster
2. Per ogni cluster così formato viene calcolata la media dei punti che lo compongono come nuovo centroide
3. Con questi K nuovi centroidi viene ripetuto il procedimento, fino a quando i centroidi non si spostano più.

Anche in questo caso si nota subito che il parametro K va fornito a priori, e con una certa attenzione, in quanto K-means è molto sensibile rispetto agli errori nella stima di questo parametro. Un secondo grosso problema è l'inizializzazione. L'algoritmo converge rapidamente ad un ottimo locale nella distribuzione dei centroidi, ma anche sperimentalmente esistono delle esecuzioni "mal inizializzate" che si appiattiscono su risultati non entusiasmanti. Inoltre, eseguendo più volte l'algoritmo sullo stesso insieme di dati, si verifica sperimentalmente il cambiamento di un discreto numero di cluster. Infine, per effetto della media, le dimensioni in cui ci sono meno dati (per esempio, la 1460ma dimensione che è "occupata" da molti meno dati che la prima), in presenza di un numero ristretto di cluster e/o di una inizializzazione casuale che "penalizzi" le dimensioni più elevate. Per questo secondo problema vedremo tra breve una soluzione proposta.

5.4.7.2 Implementazione e applicazione ai dati

Anche per K-means abbiamo ripetuto la stessa prova eseguita con PDDP e la S.O.M., analizzando del traffico "normale" e del traffico generato da Nessus. I risultati, come si può notare nelle figure a pagina seguente, sono meno indicativi.



Traffico normale classificato in 50 classi da K-Means (sopra) e traffico Nessus (sotto)

Anche una ispezione manuale dei cluster determinati da K-means ci ha lasciati un po' delusi, rispetto alle evidenti scoperte realizzate dai due algoritmi precedenti. Anche questo risultato dipende in gran parte dalla pseudocasualità dell'inizializzazione.

Per risolvere il problema dell'inizializzazione, come si è già accennato, è stato proposto l'algoritmo *Global K-Means*, composto di passi ricorsivi per cui data la "miglior soluzione" K-means del passo K-1, si trova la soluzione al passo K seguendo questo metodo:

1. Si scelgono K centroidi iniziali, prendendo i K-1 centroidi ottimali del passo precedente, con in aggiunta uno a caso dei punti rimanenti del dataset;
2. Si applica l'algoritmo K-means
3. Si ripete dal punto 1 fino ad esaurire tutti i punti del dataset
4. Si sceglie la miglior soluzione possibile, ovvero quella per cui la somma quadratica degli errori è minima

Dal momento che la soluzione ottimale per il passo 1 è semplicemente la media dei punti, potremmo pensare di aver trovato una soluzione eccellente, che sperimentalmente pare convergere all'ottimo globale (risulta difficile dimostrarlo, però). Come si dirà tra breve, purtroppo problemi computazionali rendono inapplicabile questo algoritmo al nostro caso.

Per ovviare invece al problema della predeterminazione del numero di classi, si potrebbe utilizzare l'algoritmo *Growing KMC (GKMC),* che per ogni punto in ingresso crea un nuovo cluster se la sua distanza dal cluster più vicino è superiore a una certa soglia. Questo sistema può anche attenuare la variabilità dovuta all'inizializzazione casuale dei centroidi. È tuttavia necessario determinare le dimensioni di questa "soglia", e non abbiamo molte conoscenze di dominio su cui basarci per farlo correttamente.

Per risolvere il problema dell'addestramento on-line si potrebbe pensare di ricorrere all'algoritmo *Scalable K-Means* già illustrato. Questa modifica non dovrebbe comportare significativi problemi.

5.4.7.3 Considerazioni sull'efficienza

Uno dei grandi vantaggi di K-means è la relativa leggerezza computazionale dell'algoritmo. In effetti, tra tutti e tre gli algoritmi provati, questo è l'unico a non

averci imposto vere limitazioni sulla dimensione dei dati o del set di addestramento. Tuttavia questo vantaggio si paga, salato, con la scarsa qualità del risultato determinato con una inizializzazione casuale.

È stato proposto un algoritmo di ottimizzazione globale per K-means, tuttavia, nella sua implementazione di base, Global K-Means è inapplicabilmente pesante, giacchè eseguiamo l'algoritmo K-means non più una volta, ma N(N-1)(…)(N-K) volte, quindi $O(N^K)$, con N numero dei dati e K numero delle classi. Non entusiasmante, ed infatti un test pratico ci ha dimostrato l'impraticabilità di questa strada.

Per ridurre la complessità esistono alcune euristiche, che allontanano lievemente dal risultato ottimale ma garantiscono la computabilità. Innanzitutto una prima ottimizzazione (fast global K-means) utilizza una tecnica di bound per determinare quale sia il punto più "promettente" su cui eseguire l'algoritmo. In pratica il ragionamento è questo: per ogni punto candidato ad entrare nei K centroidi, possiamo calcolare il peggior valore (upper bound) della somma degli errori quadratici, ovvero l'errore quadratico che risulterebbe inserendolo così com'e' nei K-means: infatti, da lì in avanti l'algoritmo a ogni passo ridurrebbe tale errore. Scegliere il minimo tra i bound massimi non è di per sé garanzia di ottenere il risultato migliore, ma sperimentalmente risulta distaccarsi poco. In questo modo eseguiamo K-means completamente solo una volta per ognuno dei K-1 centri che vengono progressivamente aggiunti. In aggiunta a ciò, ad ogni passaggio dobbiamo fare l'operazione di bound, che viene eseguita stimando quanti punti "cambieranno cluster" con l'inserimento del nuovo valore. Nel caso pessimo, quindi, N volte, per un totale di NK operazioni di distanza, più K-1 esecuzioni di K-means. Nonostante la significativa riduzione di complessità, anche questa strada si è dimostrata proibitiva all'atto pratico.

Un ulteriore meccanismo, basato sui k-d tree, consente di evitare di cercare tra tutti i punti del dataset, limitandosi solo a punti denominati bucket. Non è difficile rendersi conto che nonostante la differenza tra i nomi ciò che viene proposto è di utilizzare l'algoritmo PCA per determinare le direzioni principali dei dati, e su tale base generare dei buoni candidati come centroidi. Se questo, a livello teorico, è un buon metodo, a livello pratico sappiamo che con la grande dimensione della nostra matrice una suddivisione PCA è di per se un problema: a questo punto possiamo utilizzare piuttosto l'algoritmo PDDP, che ci offre risultati migliori.

5.5 CONCLUSIONI

In base alle prove sperimentali realizzate, possiamo affermare di avere mostrato come il problema della suddivisione dei payload del traffico TCP/IP sia affrontabile mediante un uso ragionato delle tecniche di clustering attualmente disponibili.

In base alle implementazioni e alle prove da noi realizzate, l'algoritmo Self Organizing Map, nella sua forma originale, si è dimostrato sufficientemente robusto e scalabile, oltre a fornire una suddivisione di buona qualità. L'algoritmo PDDP ha mostrato dei limiti di tipo computazionale, ma forse essi saranno risolti dallo sviluppo di nuovi metodi per il calcolo numerico. L'algoritmo K-means nella sua forma base si è dimostrato a nostro parere inadeguato, mentre le sue forme "globali" più evolute si sono dimostrate intrattabili computazionalmente.

Siamo convinti che risultati di performance ancora migliori dei nostri potrebbero essere ottenuti implementando gli algoritmi al di fuori della piattaforma Matlab, ed è possibile che alcuni degli altri algoritmi da noi non implementati si adattino bene al problema e forniscano risultati interessanti.

6

PROGETTO DEL SECONDO STADIO

6.1 DEFINIZIONE DEL SOTTOPROBLEMA DA ANALIZZARE

All'ingresso del secondo stadio abbiamo dei dati sui pacchetti IP in un formato più o meno standard, che comprende sia campi (decodificati) provenienti dagli header di livello 3 e 4 del pacchetto, sia la classificazione prodotta dal primo stadio, con l'eventuale aggiunta di un altro dato che indichi la "caratteristica di outlier" individuata dall'algoritmo di classificazione.

Questi dati debbono ora essere analizzati per rilevare anomalie, sia all'interno del singolo pacchetto, che soprattutto nel flusso di tali pacchetti nel tempo. L'algoritmo dovrà perciò osservare i dati, probabilmente come finestra scorrevole della massima dimensione che le esigenze computazionali ci consentono.

Un primo problema che ci possiamo porre è se sia più significativa nella nostra architettura la componente di individuazione delle anomalie su un singolo pacchetto o su una finestra di dati. Un'osservazione evidente è che molti attacchi pericolosi sono atomici, infatti molti sistemi di misuse detection funzionano adeguatamente vagliando solo il contenuto dei singoli pacchetti. D'altro canto, in un sistema per sua natura "statistico", le variazioni sui singoli pacchetti potrebbero venire facilmente disperse in una finestra lunga di dati. Siamo di fronte ad un trade-off tra l'accuratezza dell'analisi sul singolo pacchetto e la rilevanza delle correlazioni tra pacchetti diversi.

Per quanto riguarda l'output, in primissima approssimazione ci potrebbe bastare una uscita logica di tipo sfumato, che oscilli tra una perfetta normalità e una completa discrepanza rispetto al consueto. Meglio ancora sarebbe se il secondo stadio andasse oltre, e riuscisse ad indicare all'operatore umano quali siano i valori che meno collimano, ovvero i pacchetti "strani" e i motivi, i dati che li fanno ritenere strani.

L'ottimo sarebbe riuscire ad utilizzare gli input e la revisione dell'operatore per aumentare l'efficienza dell'algoritmo.

6.2 ANALISI DEI POSSIBILI ALGORITMI

6.2.1 TIPI DI ALGORITMI E LORO APPLICABILITÀ

Esistono moltissimi tipi di algoritmi utilizzabili per individuare anomalie in sequenze temporali, ma in generale la loro applicazione è limitata a serie temporali numeriche, in cui esiste un ordinamento temporale predefinito, e in cui è possibile definire un valore di distanza tra i campioni nei vari istanti di tempo. È proprio la mancanza di queste due ipotesi a proibirci l'uso di alcuni potenti strumenti matematici come l'analisi spettrale.

Escludendo a priori gli algoritmi di tipo supervisionato, gli algoritmi proposti ma non sperimentalmente utilizzati, e limitandoci alla letteratura esistente, vi sono solo pochi tipi di algoritmi che possono essere presi in considerazione.

Una possibile proposta sono gli algoritmi di Instance Based Learning (IBL), e un'altra le catene di Markov. Dell'applicazione di questi metodi all'anomaly detection abbiamo già parlato citando la letteratura esistente, tuttavia tipicamente sono stati applicati all'individuazione di anomalie nelle user session (comandi in shell), non sul traffico di rete; ma l'utilizzo del nostro primo stadio, che riduce tale traffico a pochi valori significativi, rende possibile immaginare l'estensione di tali modelli anche al traffico di rete. Esamineremo entrambi questi casi.

Gli algoritmi di clustering studiati per lo stadio precedente hanno (quasi) tutti la proprietà di essere time-independent (ne avevamo addirittura fatto un requisito primario). Algoritmi di clustering come ART sono time-dependent ma non sono fatti specificamente per studiare correlazioni temporali. Esiste tuttavia un metodo intuitivo per utilizzare un algoritmo di clustering anche per analizzare una sequenza temporale di pacchetti, e alcuni autori hanno proposto di utilizzare una S.O.M. anche per l'individuazione non supervisionata di anomalie nella sequenza temporale di traffico di rete. Proveremo a descrivere come il loro approccio possa essere ripensato come secondo stadio del nostro sistema.

Altri autori ancora hanno proposto un metodo statistico denominato P.H.A.D., Packet Header Anomaly Detection, applicabile all'insieme di dati estratti dagli header dei pacchetti. Altri metodi statistici per l'individuazione di anomalie su sequenze temporali sono basati sulla verosimiglianza. Analizzeremo anche un

metodo basato sulla teoria dell'informazione, denominato "Finestra di Parzen", che si può applicare generalmente a qualsiasi tipo di sequenza temporale. Per ognuno di questi algoritmi daremo una descrizione di base, e alcune osservazioni.

Altri autori hanno utilizzato un percettrone multistrato a cui presentare una "finestra" di eventi, ma in generale questa architettura viene utilizzata per l'apprendimento supervisionato. Tuttavia reputiamo importante esplorare anche questa possibilità.

Di seguito descriveremo brevemente gli algoritmi che abbiamo elencato, ed analizzeremo le loro applicazioni, presenti in letteratura, alle problematiche dell'intrusion detection o della rilevazione di anomalie in sequenze di dati di rete.

6.2.2 INSTANCE-BASED LEARNING

Instance Based Learning (IBL) è una classe di algoritmi di apprendimento molto generica, che si basa sulla rappresentazione di un concetto mediante una serie di istanze, denominate *dizionario*. Ogni volta che si incontra un valore nuovo, esso viene categorizzato in base alle istanze già presenti nel dizionario.

Per esempio uno schema tipico è quello del K-nearest-neighbor, in cui la nuova istanza viene categorizzata come la maggioranza delle K istanze più vicine ad essa secondo qualche metrica.

Questo algoritmo di classificazione sembrerebbe a prima vista supervisionato, ma alcuni autori ne hanno proposta una variante non supervisionata [109] da utilizzare proprio nel dominio dell'intrusion detection. In particolare viene discussa una regola di classificazione alternativa a K-nearest neighbor (in quanto quest'ultimo, in presenza di soli esempi di comportamento "normale", categorizza qualsiasi cosa come "normale"), e ulteriori adattamenti per ridurre la dimensione complessiva del dizionario.

Questo algoritmo presenta risultati molto interessanti, ma pare inevitabilmente legato alla anomaly detection host-based, in cui vi sia un numero potenzialmente limitato di codici. Non è evidente se questo paradigma si possa utilizzare per studiare le anomalie del traffico di rete.

6.2.3 S.O.M.

Dell'algoritmo S.O.M. abbiamo già parlato diffusamente analizzandone l'utilità come meccanismo di unsupervised clustering per il primo stadio del nostro sistema.

L'utilizzabilità di un algoritmo basato su S.O.M. per l'individuazione di attacchi all'interno del dataset DARPA è stata già studiata [110]. Nell'architettura proposta dagli autori, la S.O.M. viene addestrata su 6 caratteristiche delle connessioni TCP, quindi eliminando il concetto di pacchetti mediante una forma di preprocessing. È tuttavia interessante che l'architettura S.O.M. si dimostri in grado di operare anche in questo campo.

Altri autori hanno proposto NSOM [111], un prototipo che nella sua forma originaria è in grado di individuare solamente gli attacchi di tipo denial-of-service. Ciò che è interessante di NSOM non è tanto la sua semplice forma (per ogni pacchetto vengono presi come input l'indirizzo sorgente, l'indirizzo destinazione, e il protocollo), ma il concetto che utilizza per rappresentare alla rete il tempo, ovvero disporre una sequenza di N (in questo caso, 10) pacchetti come input per la S.O.M.; in seguito NSOM addestra la rete su una sequenza che descrive del traffico "normale", ed utilizza le caratteristiche di outlier detection della S.O.M. per individuare delle sequenze di traffico anomalo. Sorprendentemente, questa struttura così semplice riesce comunque ad individuare un certo numero di attacchi di denial-of-service.

Altri autori hanno proposto dei test su un dataset proposto dalla MITRE corporation [112]. Questo dataset è stato scartato da noi a causa della mancanza del payload dei pacchetti TCP, e di varie manipolazioni effettuate per "anonimizzare" gli indirizzi IP sorgente e di destinazione dei pacchetti. In questo caso sono state mostrate alla rete le seguenti feature:

- Timestamp del pacchetto
- Indirizzi e porte sorgente e destinazione
- Flag del pacchetto
- Numero di sequenza dei dati trasmessi
- Numero di acknowledge
- Receive Window
- Lunghezza del pacchetto

Nonostante le menomazioni, e nonostante il fatto che all'algoritmo siano stati mostrati i pacchetti "uno alla volta", utilizzando il timestamp come metodo per "insegnare" alla rete le correlazioni temporali, la S.O.M. ha dimostrato delle positive capacità di individuare i pacchetti anomali. A nostro avviso l'utilizzo del tempo come feature nei vettori di addestramento è da ritenersi alquanto inefficace; bastano semplici considerazioni sulla metrica per capire che questo può funzionare quando la rete è utilizzata in modo "batch", ovvero per suddividere un singolo campione di

dati. Quando essa venisse applicata on-line, la dipendenza dei cluster dal tempo dei campioni di addestramento sarebbe, nel migliore dei casi, irrilevante.

6.2.4 P.H.A.D.

P.H.A.D. (Packet Header Anomaly Detector) [113] è un metodo molto semplice di modellazione delle "sequenze normali" di header. In sostanza, il metodo prende in considerazione un certo numero di campi. Per ciascuno di questi campi vengono registrati (durante l'addestramento) i valori osservati. Ogni volta che viene osservato un valore mai visto in precedenza questo viene calcolato come "una anomalia". Per ogni campo, si registra il numero totale di anomalie che avvengono durante l'addestramento, e si calcola la probabilità di una anomalia, ovvero, chiamando n il numero di pacchetti usati nell'addestramento e r il numero di anomalie, $p = r/n$. Inoltre un campo misura il tempo trascorso dall'ultima rilevazione di una anomalia sul campo. Per i campi di lunghezza superiore al byte viene utilizzata una funzione di hash.

Durante l'esecuzione dell'algoritmo, ogni campo del pacchetto in esame viene confrontato con la lista di "valori osservati" per quel campo, e nel caso che non sia incluso in tali valori viene contrassegnato come una anomalia di peso t/p, con t tempo trascorso dall'ultima anomalia su quel campo, e p probabilità di anomalia su quel campo. Le componenti anomale vengono sommate per dare una misura della anomalia del pacchetto.

Con questo metodo, apparentemente semplice, viene rilevato circa il 50% degli attacchi contenuti nel dataset DARPA del 1999, secondo quanto riportato dagli autori. Il tempo di esecuzione dell'algoritmo è risibile, consentendo delle ottime performance.

6.2.5 METODI STATISTICI

Nel mondo reale è quantomeno problematico stabilire a priori se un set di dati sia costituito da sole azioni lecite o anche illecite, senza avere a priori un campione di cosa sia "lecito". Tuttavia un approccio di tipo statistico [114] indica come sia possibile, almeno in linea teorica, individuare le anomalie anche in assenza di un set di dati normali, sfruttando il semplice fatto che le anomalie, su grandi insiemi di dati, siano statisticamente molto più rare della normalità.

La finestra di Parzen è un metodo di stima delle densità di probabilità non parametrico, che può essere utilizzato in un test d'ipotesi di novità per individuare gli outlier in una sequenza temporale. Tra l'altro, la formulazione come test d'ipotesi consente di utilizzare direttamente un parametro di false detection rate come sensitività, caratteristica che potrebbe essere molto interessante per il discorso che abbiamo fatto sulle metriche delle performance. Alcuni autori hanno mostrato [115] come la finestra di Parzen possa essere applicata ad una sequenza di connessioni TCP (non pacchetti) per individuare degli outlier, utilizzando il dataset della Third International Knowledge Discovery and Data Mining Tools Competition [116]. Noi non abbiamo utilizzato tali dati, in quanto costituito dalla traccia di "connessioni" appunto, e non di pacchetti. Il set di dati è stato proposto per una competizione di learning supervisionato, ma ovviamente è perfettamente utilizzabile per sperimentare algoritmi non supervisionati.

L'unico vero problema del metodo della finestra di Parzen è il suo running time molto alto, che non viene in alcun modo compensato, a nostro parere, dal fatto di non richiedere sostanzialmente addestramento.

Smart Sifter [117], infine, è un algoritmo a base statistica estremamente avanzato, che cerca di individuare gli outlier in una sequenza temporale utilizzando un modello di apprendimento non supervisionato basato su un discounting learning algorithm. L'algoritmo è intrinsecamente in grado di adattarsi alle variazioni che, nel tempo, possono modificare il modello del traffico; è computazionalmente poco costoso (oltre al calcolo teorico, gli autori propongono alcuni risultati sperimentali molto incoraggianti); può gestire sia dati di tipo categorico che dati di tipo numerico. Il meccanismo si basa sulla creazione e il progressivo aggiornamento di un modello statistico della "sorgente" dei dati. A ogni ingresso il modello viene aggiornato: ingressi che tendono ad aggiornare "molto" il modello vengono considerati outlier. Anche questo modello è stato sperimentato sul dataset precedentemente citato, con risultati molto positivi.

6.2.6 PERCETTRONE MULTISTRATO

La capacita delle reti neurali a percettrone multistrato (M.L.P.) di individuare e segnalare comportamenti intrusivi è stata ampiamente dimostrata in letteratura. Le reti M.L.P. non vengono da noi prese approfonditamente in considerazione, in quanto il loro addestramento è di tipo *supervisionato* e la nostra ricerca si è focalizzata su modelli non supervisionati.

Tuttavia, riteniamo opportuno ricordare che anche tutti gli approcci illustrati in letteratura e basati su M.L.P. viene esaminato un sottoinsieme di caratteristiche dei

pacchetti di rete che esclude quasi sempre a priori il contenuto del payload, per i soliti motivi di ordine computazionale.

6.3 OSSERVAZIONI, PROPOSTE ED ESPERIMENTI

6.3.1 OSSERVAZIONI SUGLI ALGORITMI ESAMINATI

Questo breve esame degli algoritmi di anomaly detection non supervisionati fin qui proposti ed utilizzati sul traffico di rete ci porta ad alcune osservazioni. Innanzitutto, *nessuno* degli algoritmi considerati (neppure includendo la parentesi dedicata alla rete neurale supervisionata) ha utilizzato in alcun modo i dati contenuti nel payload del pacchetto. Eppure, per una notevole quantità di attacchi (basti pensare a tutti gli exploit diretti contro i webserver, che sfruttano vari problemi nella decodifica delle richieste URL) è proprio all'interno del payload dei pacchetti che si trovano le informazioni necessarie a distinguere un attacco da una richiesta perfettamente legittima. Nelle analisi sugli algoritmi proposti per il primo stadio abbiamo mostrato come essi siano in grado di categorizzare questi attacchi in maniera differente rispetto al traffico normale.

In secondo luogo, praticamente tutti gli algoritmi considerati sono stati applicati ad un sottoinsieme più o meno arbitrario delle feature di un pacchetto TCP/IP; tuttavia possono tranquillamente essere estesi a considerare tutte le feature del pacchetto.

Infine, è significativo notare come in generale l'aumento dei campi presi in considerazione dai vari algoritmi corrisponda ad una crescente capacità di individuare attacchi sempre più sofisticati.

È necessario notare che alcuni algoritmi, per esempio P.H.A.D., non considerano adeguatamente le correlazioni tra campi dello stesso pacchetto. Per fare un esempio banale, è molto curioso che un pacchetto abbia IP sorgente ed IP destinazione uguali come nel caso dell'attacco *land*, che infatti P.H.A.D. non è in grado di identificare.

6.3.2 LA NOSTRA PROPOSTA

Sulla base delle osservazioni precedenti, riteniamo di poter affermare che aggiungere in ingresso ad uno di questi algoritmi i dati sulla classificazione non supervisionata del payload dei pacchetti possa, in linea teorica, incrementare anche significativamente il numero di attacchi che un intrusion detection system anomaly-based è in grado di rilevare.

Ad esempio, come è già stato fatto notare, sfruttando in maniera adeguata le informazioni relative alla classificazione del payload, diventano accessibili, per il nostro IDS, tutti gli attacchi che sono contraddistinti da un particolare contenuto del pacchetto di rete.

Inoltre, tutta una serie di correlazioni interessanti può essere stabilita. Ad esempio, nei nostri esperimenti è stato notato come gran parte dei pacchetti di comunicazione telnet avessero un formato ben specifico, che li faceva ricadere all'interno di specifici cluster, in cui ricadevano ben pochi altri pacchetti. Pertanto, qualsiasi modello di anomaly detection potrebbe probabilmente associare con una certa confidenza tale tipo di attività con la porta numero 23. Nel momento in cui un eventuale intruso dovesse installare su una macchina della rete una backdoor, aprendo il telnet su una porta diversa, questo potrebbe legittimamente generare un alert.

Un analogo ragionamento può essere effettuato per molte altre situazioni. Abbiamo notato come, per esempio, nei trasferimenti di dati ed immagini via http ed FTP vi sia una tendenza, per gran parte dei pacchetti della comunicazione, a ricadere nello stesso cluster. Questa potrebbe essere un'altra regola d'associazione che un algoritmo non supervisionato potrebbe riuscire a scoprire.

Questo discorso, ovviamente, implica una analisi approfondita delle relazioni sul tempo non solo dei singoli campi del pacchetto, ma considerando le inter-relazioni dei campi tra loro. Questo squalifica automaticamente P.H.A.D. e affini.

Abbiamo inoltre già notato mentre discutevamo dei risultati ottenuti con gli algoritmi di clustering quanto sia diversa la distribuzione, nel tempo, tra i vari cluster, tra una scansione di vulnerabilità effettuata con Nessus e un campione di traffico normale.

Ovviamente il nostro è un ragionamento di tipo teorico e qualitativo, che indaga la possibilità teorica per l'algoritmo di scoprire una determinata correlazione, ma non può essere ritenuta una prova certa della qualità complessiva del risultato, per tutte le ragioni che abbiamo discusso parlando di valutazione dei sistemi di intrusion detection.

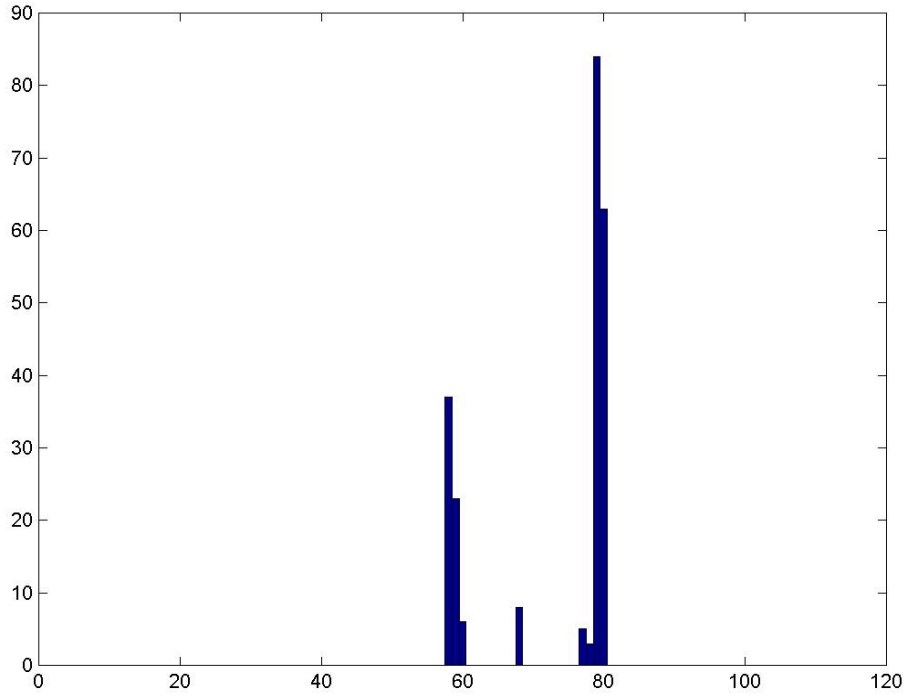
Va anche notato che questi algoritmi analizzano sostanzialmente la corrispondenza tra i valori dei campi e un qualche modello da loro costruito del traffico (più o meno esplicitamente). Ciò ha senso per tutti i dati ricavati decodificando gli header di pacchetto e per la classificazione del tipo di traffico, ma non crediamo sia particolarmente sensata per il coefficiente che esprime la possibilità che il pacchetto sia un outlier. Il fatto che questo valore sia significativo o meno, e

che debba essere considerato come uno dei valori da sottoporre all'algoritmo di secondo stadio oppure essere trattato a parte con qualche euristica, rimane dubbio; ulteriori esperimenti e valutazioni sono necessarie per decidere quanto esso sia significativo.

6.3.3 ESPERIMENTI PROOF-OF-CONCEPT

Pur non approfondendo tutte le scelte implementative per realizzare un particolare algoritmo di secondo stadio, abbiamo effettuato un discreto numero di prove per verificare la nostra proposta. Ci siamo limitati a considerare la relazione "classificazione del payload – porta di destinazione", che dovrebbe essere sufficientemente "semplice" da analizzare per uno qualsiasi degli algoritmi precedentemente esaminati.

Abbiamo utilizzato come algoritmo di primo stadio una S.O.M. 10x10, addestrata per 10000 epoche, e abbiamo suddiviso con essa circa 5000 pacchetti di traffico. Abbiamo preso in considerazione il traffico in ingresso sulla porta 21/TCP, ovvero le richieste FTP. Queste hanno una struttura molto chiara, come risulta evidente dal diagramma che riportiamo.



Classificazione effettuata da una S.O.M. del traffico TCP in ingresso sulla porta 21

Abbiamo quindi estratto, da un database di ricerca disponibile su Internet, ArachNIDS [118], le firme di alcuni attacchi diretti contro i server FTP, o meglio, i dump TCP dei pacchetti in questione, e li abbiamo analizzati con la stessa rete. ArachNIDS cataloga gli exploit usando anche i relativi codici CVE (Common Vulnerability Exposure) e Bugtraq, che riportiamo per riferimento.

Il primo attacco analizzato (codice ArachNIDS IDS453), è l'uso di un exploit contro la vulnerabilità "format string" del server FTP wu-ftpd (CVE CAN-2000-0573, Bugtraq 1387). Il pacchetto contiene un payload così strutturato (lo rappresentiamo con il classico output di tcpdump, a sinistra il contenuto esadecimale, a destra la rappresentazione ASCII):

```
53 49 54 45 20 45 58 45 43 20 25 30 32 30 64 7C  SITE EXEC %020d|
25 2E 66 25 2E 66 7C 0A                          %.f%.f|.
```

Il payload viene classificato nella classe 69, in cui (come forse si può notare anche dal grafico) non ci sono stringhe FTP. Questa anomalia dovrebbe essere semplice da notare per un sistema di anomaly detection.

Abbiamo ottenuto un risultato interessante anche analizzando un attacco di denial-of-service mediante l'uso del globbing (codice ArachNIDS IDS487). L'aggressore qui cerca di far crollare il server FTP facendogli espandere molte volte il carattere wildcard (*). Di questo attacco esistono almeno 3 varianti note, ma se ne potrebbero scrivere una infinità:

```
Variante 1:
4C 49 53 54 20 2A 2F 2E 2E 2F 2A 2F 2E 2E 2F 2A  LIST */../*/../*
2F 2E 2E 2F 2A 2F 2E 2E 2F 2A 2F 2E 2E 2F 2A 2F  /../*/../*/../*/
2E 2E 2F 2A 2F 2E 2E 2F 2A 2F 2E 2E 2F 2A 2F 2E  ../*/../*/../*/.
2E 2F 2A 2F 2E 2E 2F 2A 2F 2E 2E 2F 2A 2F 2E 2E  ./*/../*/../*/..
2F 2A 0D 0A                                      /*..

Variante 2:
4C 49 53 54 20 2A 2F 2E 2A 2F 2A 2F 2E 2A 2F 2A  LIST */.*/*/.*/*
2F 2E 2A 2F 2A 2F 2E 2A 2F 2A 2F 2E 2A 2F 2A 2F  /.*/*/.*/*/.*/*/
2E 2A 2F 2A 2F 2E 2A 2F 2A 2F 2E 2A 2F 2A 2F 2E  .*/*/.*/*/.*/*/.
2A 2F 2A 2F 2E 2A 2F 2A 2F 2E 2A 2F 2A 2F 2E 2A  */*/.*/*/.*/*/.*
2F 0D 0A                                         /..

Variante 3:
```

```
4C 49 53 54 20 2E 2A 2E 2F 2A 3F 2F 2E 2A 2E 2F  LIST .*./*?/.*./
2A 3F 2F 2E 2A 2E 2F 2A 3F 2F 2E 2A 2E 2F 2A 3F  *?/.*./*?/.*./*?
2F 2E 2A 2E 2F 2A 3F 2F 2E 2A 2E 2F 2A 3F 2F 2E  /.*./*?/.*./*?/.
2A 2E 2F 2A 3F 2F 2E 2A 2E 2F 2A 3F 2F 2E 2A 2E  *./*?/.*./*?/.*.
2F 2A 3F 2F 0D 0A                                /*?/..
```

Ciò che è interessante è che la rete S.O.M. non soltanto classifica tutte e tre le varianti in un nodo che normalmente non ospita traffico destinato alla porta 21 (per la precisione il nodo 97), ma le classifica tutte e tre assieme. Le firme per un IDS basato su signature, nell'intento di generalizzare, fanno un match solo su '/*'. Questa è per esempio la firma di Snort:

```
alert TCP $EXTERNAL any -> $INTERNAL 21 (msg: "IDS487/ftp_dos-ftpd-
globbing"; flags: A+; content: "|2f2a|"; classtype: denialofservice;
reference: arachnids,487;)
```

Tuttavia, l'uso di tale firma è sconsigliato, perché molto prona a individuare dei falsi positivi (e in effetti si possono pensare comandi perfettamente leciti che genererebbero un alert di tale firma). In questo caso, la potenzialità di generalizzazione del nostro sistema sarebbe superiore a quella di una architettura misuse-based.

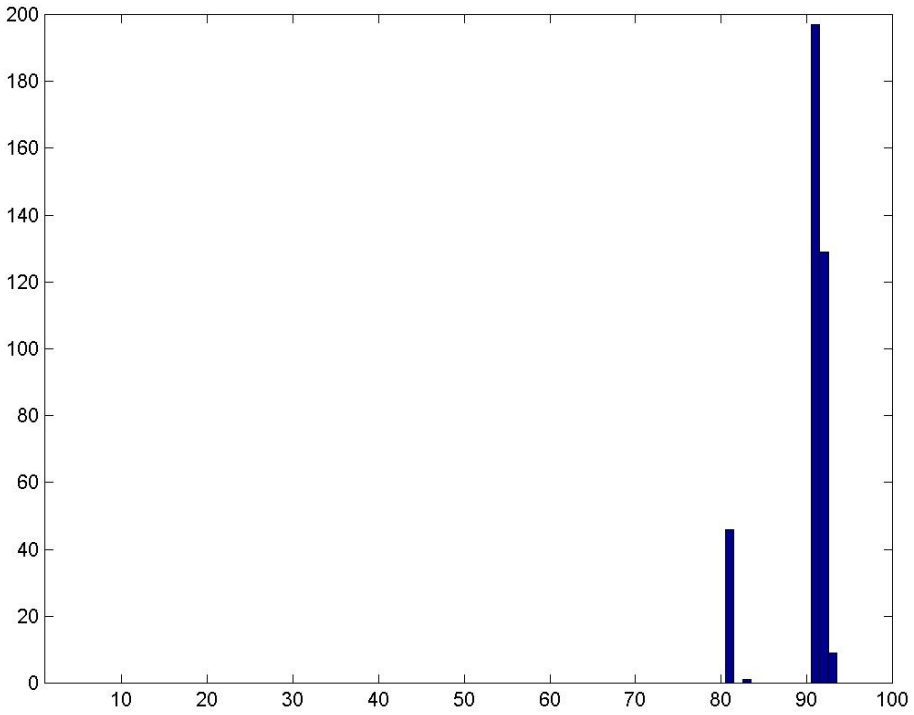
Abbiamo poi sperimentato un attacco basato su un overflow della validazione della password (ArachNIDS IDS 287):

```
50 41 53 53 20 90 90 90 90 90 90 90 90 90 90 90  PASS ..........
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
[ la stessa riga si ripete altre 13 volte, poi: ]
90 90 90 90 90 90 90 90 90 90 31 C0 31 DB 31     ..........1.1.1
C9 B0 46 CD 80 31 C0 31 DB 43 89 D9 41 B0 3F CD  ..F..1.1.C..A.?.
80 EB 6B 5E 31 C0 31 C9 8D 5E 01 88 46 04 66 B9  ..k^1.1..^..F.f.
FF FF 01 B0 27 CD 80 31 C0 8D 5E 01 B0 3D CD 80  ....'..1..^..=..
31 C0 31 DB 8D 5E 08 89 43 02 31 C9 FE C9 31 C0  1.1..^..C.1...1.
8D 5E 08 B0 0C CD 80 FE C9 75 F3 31 C0 88 46 09  .^.......u.1..F.
8D 5E 08 B0 3D CD 80 FE 0E B0 30 FE C8 88 46 04  .^..=.....0...F.
31 C0 88 46 07 89 76 08 89 46 0C 89 F3 8D 4E 08  1..F..v..F....N.
8D 56 0C B0 0B CD 80 31 C0 31 DB B0 01 CD 80 E8  .V.....1.1......
90 FF FF FF FF FF FF 30 62 69 6E 30 73 68 31 2E  .......0bin0sh1.
2E 31 31 0D 0A                                   .11..
```

L'exploit viene categorizzato nella classe 81, in cui non sono inclusi altri campioni di traffico destinati alla porta FTP. È anche da notare che di questo exploit esistono alcune varianti non funzionanti che vengono individuate allo stesso modo.

Il database di firme fa notare che questo attacco viene riconosciuto anche per la lunga sequenza di NOP (valore esadecimale '90'). Per sperimentare ulteriormente, abbiamo sostituito le coppie '90 90' con l'alternativa 'eb 00'. La S.O.M. classifica anche questa variante nella classe 81.

Siamo poi passati a considerare le richieste sulla porta 80/TCP, ovvero le richieste web. Il diagramma che segue, ottenuto dalla stessa rete e con lo stesso insieme di pacchetti utilizzati per gli esempi di cui sopra, mostra come anche in questo caso vi sia una distribuzione notevolmente "compatta" tra le varie classi.



Classificazione delle richieste TCP sulla porta 80 (web), effettuata dalla stessa S.O.M.

Abbiamo sottoposto alla rete un exploit di I.I.S. (Internet Information Services, il server web di Microsoft) basato sull'uso di un buffer overflow in una estensione ISAPI (CVE CAN-2001-0241). L'exploit è catalogato come IDS535 da ArachNIDS (per la stessa vulnerabilità ne sono riportati altri, ma questo è completo); non

riportiamo la traccia dei pacchetti in quanto estremamente lunga. Si tratta infatti di 3 pacchetti successivi inviati alla porta 80. Tuttavia, i tre pacchetti potrebbero anche essere assemblati in uno solo (in questo caso sono stati divisi, probabilmente a causa di una maximum segment size fissata a 576 byte).

Abbiamo sottoposto alla rete sia i tre pacchetti separati (classificati come appartenenti alla classe 71, che non contiene pacchetti destinati alla porta 80), sia congiuntamente (classificati appartenenti alla classe 22, che non solo non contiene nessun pacchetto destinato alla porta 80, ma contiene solo pochissimi pacchetti "outlier" derivanti da altre classificazioni).

Abbiamo sperimentato anche un exploit contro un bug del demone telnet (CVE CVE-2000-0733, Bugtraq 1572, codice dell'exploit IDS304), sottoponendolo alla rete. Il pacchetto ha questo aspetto:

```
FF FA 24 00 01 5F 52 4C 44 00 20 20 20 20 20 7F  ..$.._RLD.     .
C4 98 1C 20 20 20 20 7F C4 98 1E 20 20 20 04 10  ...    ....   ..
FF FF FF FF 24 02 02 03 F3 23 FF FF 02 02 14 23  ....$....#.....#
E4 FE 08 23 E5 FE 10 AF E4 FE 10 AF E0 FE 14 A3  ...#............
E0 FE 0F 03 FF FF FF FF CC 2F 62 69 6E 2F 73 68  ........./bin/sh
25 33 32 36 31 34 63 25 31 31 24 68 6E 25 38 36  %32614c%11$hn%86
30 30 30 63 25 31 32 24 68 6E FF F0              000c%12$hn..
```

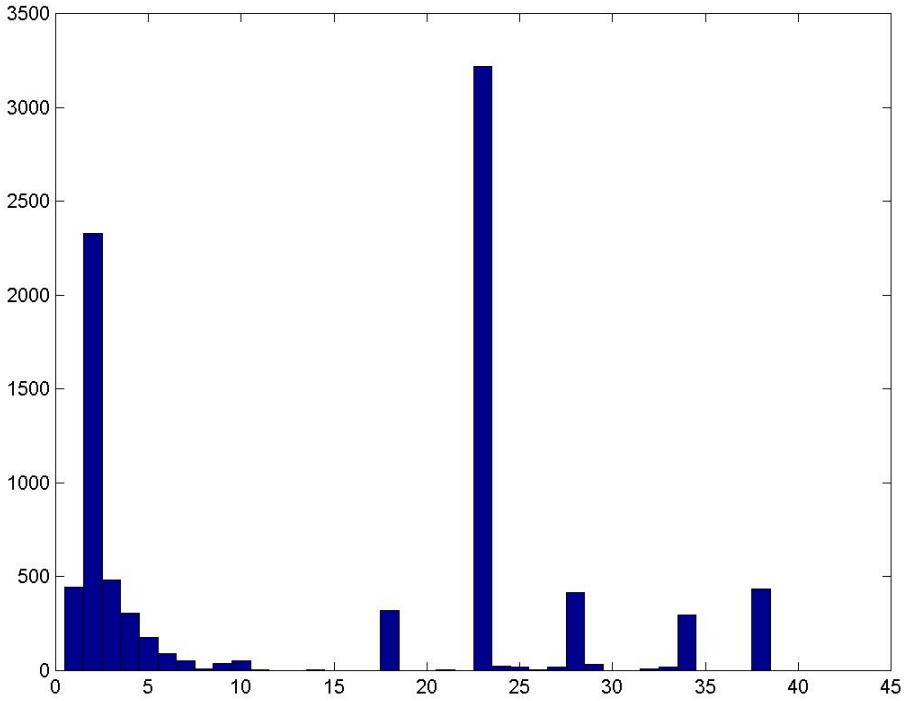
La S.O.M. lo classifica nella categoria 93, in cui non sono inclusi altri pacchetti destinati alla porta telnet.

Riteniamo con questi esempi di avere mostrato come un algoritmo di correlazione di secondo stadio possa, apprendendo le correlazioni tra "porta" e "classi di traffico dirette ad essa normalmente", individuare facilmente una serie di attacchi. Tra l'altro, realizzare un secondo stadio del genere è concettualmente banale.

6.3.4 IMPLEMENTAZIONE PROOF-OF-CONCEPT

Pur non approfondendo le nostre valutazioni, abbiamo voluto effettuare qualche esperimento realizzando un secondo stadio "di prova" basato su uno degli algoritmi descritti ad inizio capitolo. Per semplicità abbiamo scelto di utilizzare un algoritmo S.O.M. a cui mostrare una finestra di pacchetti. Abbiamo pertanto creato una struttura FIFO da 290 elementi, in cui fare "scorrere" una finestra di 10 pacchetti, rappresentati da tutte e 29 le caratteristiche calcolate dal nostro primo stadio. Le caratteristiche sono state tutte normalizzate sull'intervallo tra 0 e 1 dividendo ognuna per il "massimo" registrato nei pacchetti d'addestramento.

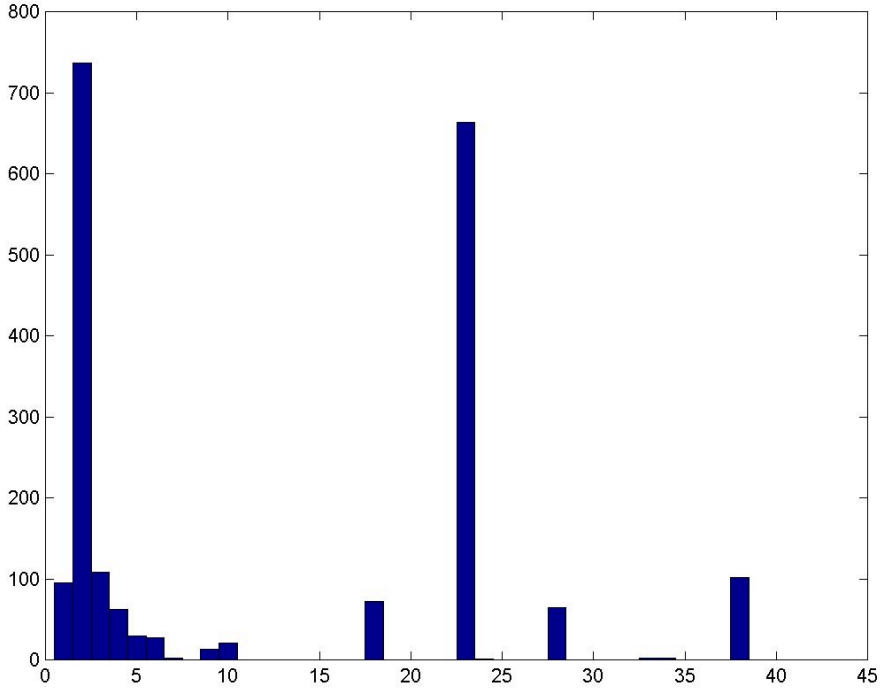
Abbiamo fatto scorrere in questa struttura alcune migliaia di pacchetti di traffico "normale", addestrando con questi una rete S.O.M. "di secondo stadio" di dimensioni 5x8, per 10.000 epoche. Successivamente, abbiamo mostrato alla rete il flusso di 8.000 pacchetti, ottenendo per ciascuno di essi una classificazione, relativa alla finestra formata da esso e dai 10 pacchetti precedenti. Riportiamo in figura tale classificazione.



Classificazione di un flusso di 8.000 pacchetti effettuata da una S.O.M utilizzata come secondo stadio; ogni classificazione si riferisce ad una finestra da 10 pacchetti

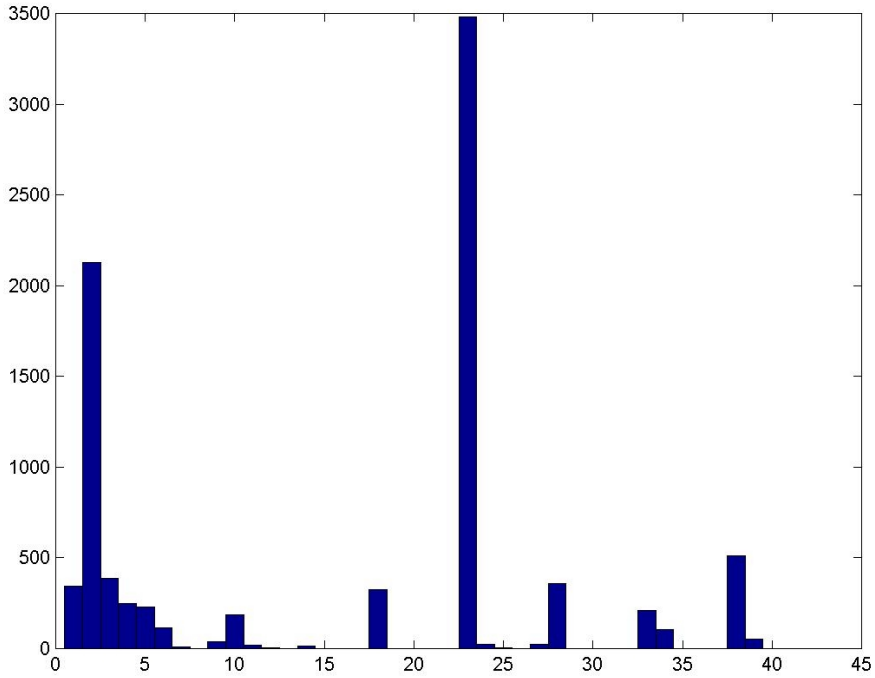
Come si può facilmente notare, la grandissima maggioranza dei pacchetti converge in pochissimi dei 40 cluster della rete. Ci preme sottolineare che questa seconda classificazione viene effettuata in modo non supervisionato, osservando ogni pacchetto in "coda" a una finestra formata dagli ultimi 10 pacchetti. È quindi, concettualmente, una versione molto "rozza" del secondo stadio che abbiamo prospettato fin qui. Ulteriori esperimenti ci hanno consentito di notare che la dispersione in classi non cambia di molto se invece di usare una finestra di 10 pacchetti ne utilizziamo una con 20.

Inoltre, estraendo una finestra assolutamente casuale di 2000 sugli 8000 pacchetti, abbiamo ottenuto il seguente grafico, che dimostra una perfetta invarianza della distribuzione delle classificazioni effettuate dalla S.O.M. di secondo stadio:



Classificazione di 2000 pacchetti a caso scelti tra i precedenti 8000

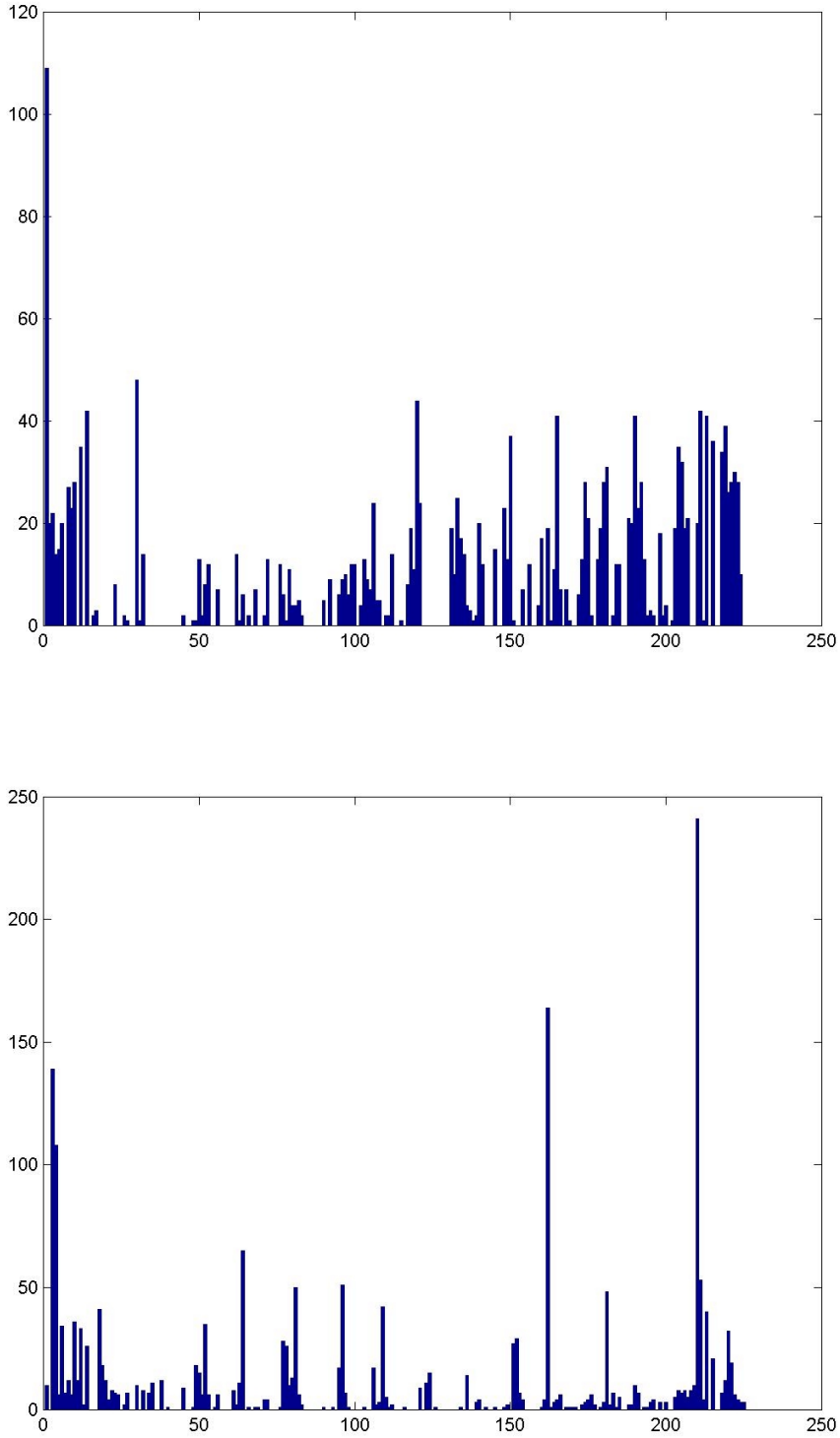
Abbiamo quindi provato a classificare 2000 pacchetti tratti dalla traccia del solito scanner “Nessus”. Il risultato (che riportiamo in un grafico nella pagina successiva) ci ha lasciati lievemente delusi: infatti, la traccia è sostanzialmente identica alla precedente, con sole minime variazioni: ciò non corrisponde minimamente alle nostre aspettative, per cui abbiamo provato a dedurre quale potesse essere il problema.



2000 pacchetti dello scanner Nessus classificati dalla stessa S.O.M. di secondo stadio, con tutte le 29 variabili

Abbiamo provato ad eliminare alcune delle 29 variabili, lasciando solo un sottoinsieme di dati che avevamo visto essere particolarmente significativi: porta sorgente e destinazione, classificazione del payload e le flag di frammentazione del TCP. Abbiamo inoltre ritenuto opportuno aumentare il numero di nodi della S.O.M., considerando che uno degli elementi da classificare erano i 100 tipi differenti di payload.

Abbiamo dunque addestrato una S.O.M. di secondo stadio da 15x15 nodi, sempre per 10.000 epoche, e sempre con una finestra di 10 pacchetti (anche se le dimensioni ridotte ci avrebbero consentito finestre più lunghe), ed abbiamo nuovamente sottoposto alla rete del traffico normale e il traffico generato da Nessus, ottenendo risultati ben più lusinghieri. In totale, ben 290 "finestre" su 2000 vengono classificate, dalla rete di secondo stadio, in cluster che il traffico normale lascia completamente vuoti. Ci teniamo a sottolineare che questo risultato viene verificato, in media, qualunque sia l'intervallo da 2000 campioni di traffico normale prescelto. Sicuramente, per una implementazione totalmente sperimentale e relativamente poco studiata di un algoritmo di secondo stadio, questo è un risultato molto incoraggiante.



Risultati della rete 15x15 su un insieme di caratteristiche ridotto. Sopra, traffico normale, sotto, traffico Nessus

6.3.5 ALCUNE OSSERVAZIONI SULL'ADDESTRAMENTO

Come abbiamo già rilevato in precedenza, i due stadi possono essere addestrati anche separatamente. In particolare, il primo stadio *deve* essere già addestrato per iniziare l'addestramento del secondo.

Inoltre, come è stato fatto notare, per gli algoritmi del primo stadio esiste la possibilità teorica di un addestramento continuo on-line, ma la praticabilità di tale soluzione è tutta da dimostrare. Viceversa, gli algoritmi del secondo stadio sono quasi tutti sviluppati per addestrarsi on-line, analizzando il flusso di informazioni mano a mano che gli viene sottoposto (di nuovo, P.H.A.D. fa eccezione).

Il secondo stadio inoltre affronta in maniera più significativa il problema del "concept drift". Laddove, sperimentalmente, le suddivisioni effettuate dall'algoritmo di primo stadio su un campione anche ristretto di dati paiono estendersi bene anche ad altre situazioni, è altamente probabile che il flusso nel tempo dei pacchetti di rete cambi, seguendo l'evoluzione della rete e delle tecnologie utilizzate. Alcuni esempi banali possono essere l'aggiunta e la rimozione di macchine, o il cambiamento degli IP di rete interna.

Questa variazione del contesto ambientale deve essere riassunta nel modello di normalità che il secondo stadio di IDS mantiene; questa problematica tipica di tutti i sistemi di apprendimento on-line viene denominata "concept drift", e nell'ambito particolare dell'intrusion detection ha un risvolto non trascurabile: il sistema deve essere sufficientemente elastico da adattarsi ai lenti cambiamenti lungo il tempo, e sufficientemente rigido da rilevare comportamenti devianti composti anche di pochi pacchetti.

Ciò ci conduce a considerare una strategia diversa per l'addestramento del primo e del secondo stadio: l'uno verrà addestrato in modo batch all'inizio delle operazioni dell'IDS; il secondo invece verrà continuamente aggiornato on-line.

Questo tipo di configurazione non è totalmente sperimentale: esistono infatti in letteratura sistemi analoghi di cui è stata verificata e studiata la stabilità e la convergenza [119].

Non dobbiamo dimenticare che in altri casi la nascita di nuovi strumenti e tecnologie può, ovviamente, rendere "superata" o da aggiornare anche la suddivisione effettuata dall'algoritmo di primo stadio: negli anni del boom degli

strumenti peer-to-peer per lo scambio di file musicali, ad esempio, il flusso di pacchetti scambiato da questi software è sicuramente cresciuto fino a diventare una porzione significante del traffico di Internet, e in scala, anche delle reti di cui il nostro IDS avrebbe potuto occuparsi.

Si può perciò prevedere che l'algoritmo di primo stadio debba essere strutturato in modo da poter essere aggiornato in particolari situazioni.

6.4 CONCLUSIONI

Abbiamo mostrato come esistano, e siano già parzialmente utilizzabili, alcuni algoritmi di apprendimento non supervisionato che corrispondono al secondo stadio del sistema da noi proposto. Abbiamo analizzato la letteratura disponibile su di essi, evidenziando pregi e difetti di ognuno.

In particolare abbiamo rilevato il difetto comune di scartare a priori come intrattabile il contenuto di ogni pacchetto, e abbiamo proposto delle ragioni logiche per mostrare come i dati ottenuti dal nostro primo stadio possano ampliare il numero di tipi di intrusione rilevabili da un sistema del genere. Abbiamo proposto una serie di osservazioni preliminari che giustificano i nostri ragionamenti.

Abbiamo altresì proposto il risultato, da ritenersi assolutamente preliminare ma molto incoraggiante, di una implementazione "proof-of-concept" di un possibile secondo stadio, e della sua applicazione a dati reali.

7

CONCLUSIONI

In questo lavoro abbiamo illustrato le problematiche fondamentali connesse alla sicurezza dell'informazione, e come queste conducano naturalmente alla formulazione del problema della intrusion detection. Abbiamo illustrato gli approcci all'intrusion detection che si riscontrano nel panorama scientifico e commerciale, ed abbiamo discusso i tipici problemi che i sistemi IDS devono affrontare nel "mondo reale".

Abbiamo proposto un modello innovativo di Intrusion Detection System di tipo "anomaly detection" e "network based", basato su tecniche di apprendimento non supervisionato. Abbiamo illustrato le caratteristiche generali dei due stadi separati di cui tale sistema si compone, ed abbiamo proposto una serie di valutazioni implementative per il sistema.

In seguito abbiamo analizzato approfonditamente il primo stadio del sistema, esplorando le tecniche a disposizione per realizzarlo. Abbiamo creato delle implementazioni sperimentali di tre tecniche per il clustering non supervisionato, e le abbiamo applicate a dati di test reali, producendo una serie di prove comparative che hanno dimostrato a nostro avviso la validità dell'approccio seguito. Abbiamo corredato tali prove di valutazioni prestazionali qualitative, sia teoriche che pratiche.

Infine, abbiamo analizzato le problematiche connesse alla realizzazione del secondo stadio. Abbiamo constatato che il problema è già stato affrontato, sia pure parzialmente, in letteratura, ed abbiamo analizzato gli approcci seguiti, evidenziandone i limiti e le qualità. Abbiamo poi mostrato, mediante prove di concetto di tipo qualitativo, come queste tecniche possano essere utilizzate come secondo stadio del nostro sistema, e come l'utilizzo del nostro primo stadio basato su clustering possa ampliare le capacità di individuazione di attacchi di questo tipo di strumenti.

Per ora i nostri risultati sono da considerarsi di natura preliminare. Una valutazione complessiva ed approfondita comporterebbe la realizzazione, per il

secondo stadio, di una serie di prove ed esperimenti analoghi a quelli da noi compiuti per il primo stadio del sistema, ed in seguito di prove sperimentali sul sistema interconnesso dei due stadi, in quanto, come abbiamo fatto notare, non ci sono "prove" teoriche della convergenza di funzionamento di due algoritmi di apprendimento non supervisionato connessi in cascata, per quanto esistano numerosi esempi, in svariati campi, che mostrano "praticamente" che una architettura del genere funziona. Ci permettiamo di osservare che ulteriori approfondimenti di tipo teorico su questa specifica problematica potrebbero essere di interesse in una vasta serie di applicazioni.

Inoltre, potrebbe essere interessante esplorare varie "strade" alternative, algoritmi diversi o modificati, e vari approfondimenti complementari che ci si sono presentati di fronte durante il lavoro, e che abbiamo tralasciato per ovvie esigenze di semplicità. Un'altra tematica da noi lasciata sullo sfondo, ma di sicuro interesse nel campo pratico, è l'aiuto all'interpretazione "umana" dei risultati ricavati dal sistema, e l'eventuale possibilità per l'operatore di intervenire "manualmente" per raffinare le scoperte fatte dal sistema, con una sorta di apprendimento semisupervisionato.

A

APPENDICE - UN IDS COMPORTAMENTALE

INTRODUZIONE

Queste note sono il sunto dello studio, presentato come progetto d'esame per il corso di Ingegneria della Conoscenza e dei Sistemi Esperti, sotto la guida del professor Marco Colombetti. Si potranno riconoscere gli abbozzi di alcune delle idee che si trovano, più compiutamente formate, all'interno del lavoro di tesi.

Nell'elaborato d'esame abbiamo proposto un punto di vista originale per l'approccio agli Intrusion Detection System. In particolare ci siamo focalizzati sull'esigenza di classificare il *comportamento dell'utente*. Un IDS cerca di distinguere tra un utilizzo normale e consentito del sistema informatico, e un abuso. Pertanto abbiamo suggerito di analizzare tale problema con un approccio derivato dall'ingegneria del comportamento o *behavior engineering*.

Il problema in questione è noto in molti campi (per citarne alcuni, la psicologia, l'etologia, e la sociologia) come problema della *classificazione dei comportamenti*. La nostra ricerca ha esplorato gli approcci di queste scienze al problema e ha cercato di determinare approcci che potessero essere tradotti in un sistema informatico.

SPUNTI DALL'ETOLOGIA E DALLA PSICOLOGIA

TERMINOLOGIA COMPORTAMENTI, PATTERN, DEVIAZIONI

Abbiamo innanzitutto osservato che esiste già una sovrapposizione tra le terminologie utilizzate dalle diverse discipline. Il termine *"comportamento"* per esempio viene descritto dall'etologia come "the *stable, coordinated* and *observable* set of reactions it [an animal] presents to some kinds of stimulations, either *inner stimulations* (or *motivations*) or *outer stimulations* (or *stimuli*)".

È abbastanza evidente che la definzione non può essere adattata in modo semplice al nostro problema. Per esempio, riferendoci alla cornice dell'intrusion detection, potremmo definire il comportamento dell'utente come "l'insieme osservabile e coordinato di azioni (comandi e input) che un utente esegue su un sistema informatico per ottenere uno scopo

Vorremmo essere chiari sull'approccio. Non ci stiamo concentrando sul comportamento del sistema informatico (che è quasi completamente predicibile e computabile, eccettuate le ovvie restrizioni imposte dal teorema di Rice), ma sul comportamento dell'utente le cui azioni e reazioni "osservabili e rilevanti" sono i comandi e i dati che egli inserisce nel sistema.

Parleremeo di comportamento *tipico* intendendo non normale, ma *più probabile* Il complementare sarà un comportamento *atipico* o *anomalo*. Sottolineiamo questo concetto che è sfuggito nelle prime implementazioni di sistemi IDS basati sull'anomalia: non sempre l'atipico è indice di un fine non in linea con le politiche di sicurezza prestabilite.

Altro concetto che utilizzeremo è quello di *pattern*, che indica un insieme di elementi condivisi tra comportamenti *lievemente* differenti, utilizzati per classificarli.

LE MOTIVAZIONI DELL'AZIONE

Teitelbaum propose, nel 1967, tre livelli crescenti di complessità nell'analisi dei comportamenti. Il primo livello riguardava i comportamenti di riflesso, lo studio dei riflessi generati dai ricettori sensoriali e trasportati dai nervi. Il secondo livello di complessità riguarda lo studio della struttura degli istinti di una specie, aggiungendo quindi agli stimoli e ai riflessi componenti come "piacere" e "rabbia". Al di sopra di questi due livelli di azioni spontanee e innate c'è un terzo livello di azioni "intenzionali", ovvero quelle che un animale inizia spontaneamente per raggiungere i propri fini.

Il concetto di "motivazione" è dunque cruciale in etologia così come in psicologia, ed è stato tema di approfondimenti filosofici. Ricordiamo le posizioni edonistiche di Hobbes; la teoria evolutiva di Darwin, dove le motivazioni sono parte fondamentale del concetto di adattamento; anche i filosofi funzionalisti hanno dato grande rilevanza alle motivazioni, fondendole con il concetto di istinto, e individuandole come la ragione fondamentale di qualsiasi azione. Anche la psicoanalisi dedica parte della propria teoria alle motivazioni, chiamandole "pulsioni".

Noi definiamo:

- *le motivazioni* come i fattori dinamici del comportamento, che fanno in modo che un organismo agisca e lo dirigono verso un obiettivo;
- *gli istinti* come le risposte innate e organizzate, tipiche di una specie che si è adattata filogeneticamente ad un ambiente;
- *i reflessi* come risposte innate ed involontarie agli stimoli che agiscono su un organismo;
- *le pulsioni* come componenti psicologiche in grado di produrre uno stato di eccitazione in un organismo e di dirigterlo ad una azione; le pulsioni sono una miscela di componenti innati ed appresi.

Ovviamente, ciò che ci interessa realmente nel campo della intrusion detection sono le motivazioni. In sostanza, ciò che cercheremo di fare sarà di stabilire quale *finalità* un utente sta perseguendo, e se questa sua finalità è accettabile o meno. Un comportamento anomalo di per sé non è un problema se non ha una motivazione deviante. D'altro canto, traffico perfettamente anonimo e normale potrebbe essere dettato da una finalità deviante.

LA SCELTA DELL'AZIONE

Per sopravvivere, un animale deve impegnarsi in un numero di attività differenti, molto spesso mutuamente esclusive tra loro. Il problema di capire come e perché un animale scelga di fare una attività e non un'altra è noto come problema di scelta dell'azione, o *action selection* [120]. Incidentalmente lo stesso problema si pone nella progettazione di agenti intelligenti nel campo dell'Intelligenza [121]. Sono stati sviluppati svariati modelli computazionali per la scelta dell'azione, spesso sulla base di osservazioni etologiche, per esempio nello studio dei cosiddetti *animats*.

Normalmente, la ricerca punta a comprendere (o nel particolare caso della I.A., a riprodurre) il meccanismo di scelta che fa sì che un animale proceda a compiere un'azione invece di un'altra. Dobbiamo comunque essere realisti ed ammettere che non c'è alcuna prova reale che un tale meccanismo esista. In effetti, alcune ricerche mostrano come il comportamento non possa essere semplicemente ritenuto un prodotto dello "stato interno" dell'agente ma piuttosto un risultato congiunto dell'agente, dell'ambiente in cui è immerso, e anche dell'osservatore che a tale comportamento sta attribuendo un significato. Altri approcci [122] eliminano completamente i concetti di "scelta dell'azione", e un comportamento credibile e consistente viene ottenuto come sovrapposizione di svariate "funzioni d'attivazione", nessuna delle quali rappresenta un comportamento di per sé. Un algoritmo genetico viene usato per calibrare tale sovrapposizione e ottenere un comportamento complessivo specificato.

È dunque possibile che sia solo la nostra interpretazione del comportamento degli animali a darci l'impressione – errata – di una "scelta" tra vari corsi d'azione. Il meccanismo reale su cui si basa il loro comportamento è insomma tuttora ignoto, e forse lo sarà ancora per lungo tempo.

I nostri modelli per l'individuazione delle ragioni di un particolare comportamento sono evidentemente basate sulla supposizione (indimostrata) che esista in effetti una motivazione alla base della sequenza d'azioni osservata. Questo ci pare un presupposto ragionevole per l'intrusion detection (le azioni degli utenti hanno *quasi* sempre una spiegazione razionale), ma dobbiamo tenere in conto che per l'etologia tale supposizione non è da ritenersi valida a priori.

FIXED ACTION PATTERN

Gli etologisti definiscono comunemente "fixed action patterns" le unità atomiche di comportamento istintivo negli animali. Si tratta dunque di comportamenti situati al secondo livello dello schema di Teitelbaum, e che perciò hanno le seguenti caratteristiche: sono meccanici, stereotipati (cioè sempre autosomiglianti) sia a livello dello stesso individuo che trasversalmente attraverso tutta la specie, e in genere ottengono qualche obiettivo particolare.

Inoltre, i comportamenti istintivi in generale, ed i FAP in particolare, sono di tipo "all-or-nothing": una volta iniziati, vengono portati a termine, e se qualcosa interrompe l'animale mentre compie un FAP, egli non lo porterà più a termin; alternativamente, lascerà perdere, o ricomincerà la sequenza da principio.

Per descrivere una sequenza di azioni come un FAP gli etologi richiedono che essa sia estensivamente presente, ben documentata, stereotipata, ed indipendente da qualsiasi altra situazione o comportamento, tranne al più uno chiamato *releasor*, il quale attiva il FAP attraverso un meccanismo tipo "filter-trigger", chiamato *Innate Release Mechanism* o IRM. È importante notare che sia il releasor sia l'IRM possono essere puramente interni all'animale, senza il coinvolgimento di alcuna percezione o stimolo esterno.

In alcuni casi la forza dello stimolo, filtrata dall'IRM, produce una maggiore o minore intensità (laddove possibile) del FAP; in altri casi il FAP pare largamente indipendente dalla forza dello stimolo.

Una nuova tendenza in campo etologico sta delineando il concetto di MAP, Modal Action Pattern, sequenze di azioni con parti fisse e parti variabili. In un articolo del professor George Barlow [123], "Ethological units of behavior", l'intero concetto di

FAP e di IRM viene messo in discussione nel dettaglio. Barlow critica fortemente l'insieme di rigidi criteri fissati per definire il FAP. In particolare:

a. il fatto che un FAP debba avere un fattore causale distinto da quello di qualsiasi altro FAP
b. il fatto che lo stimolo che dà il via al FAP, una volta rilasciato il comportamento, non abbia più alcun controllo su di esso.
c. Il fatto che tutti i componenti del FAP debbano apparire in una sequenza totalmente predicibile.

Gran parte dei comportamenti, nota Barlow, fornendo molti esempi, non sono caratterizzati da simili rigidi criteri: spesso alcune parti dei comportamenti avvengono in ordine diverso, e possono avere una intensità modulata durante l'esecuzione. Altri FAP non sono che versioni particolarmente stereotipate di comportamenti che hanno tante forme. Infine, alcuni comportamenti che definiamo FAP possono avere delle modulazioni che sfuggono alla nostra osservazione.

Insomma, la critica di Barlow è che i criteri di definizione dei FAP sono troppo dipendenti dalle caratteristiche osservate dagli esseri umani, più che dalle caratteristiche intrinseche dei comportamenti. Su tale base egli rifiuta l'idea dei FAP e suggerisce che l'ambiente possa modulare anche il comportamento più stereotipale. Egli definisce questa nuova prospettiva MAP, modal action pattern, ovvero "[a] spatio temporal pattern of coordinated movement, that clusters around some mode making it recognizable as a distinct behavior pattern".

Vorremmo notare quanto il modello FAP sia simile all'uso di pattern statici nell'intrusion detection tradizionale, mentre i MAP sono un modello flessibile di descrizione di unità comportamentali che, a nostro parere, non ha corrispondenze a livello informatico.

DISPLAY

Un sottoinsieme dei FAP prende il nome di "display", e costituisce l'insieme dei meccanismi di comunicazione di una specie. Si pensi ad esempio alla varietà di espressioni dei cani.

In un interessante circolo virtuoso, un "display" può essere il releasor di una risposta di qualche genere. Per esempio, pensate al comportamento di un uccello che nutre la sua nidiata di piccoli. Il genitore atterra sul bordo del nido, e muove il collo in un modo particolare. Quasi immediatamente, i piccoli iniziano a strillare e ad agitarsi per il cibo. Questo, a sua volta, istiga il genitore a nutrirli con un'altra sequenza di movimenti, e così via.

Nel campo dei display funziona un principio interessante denominato *principio di antitesi*, che significa che due display con significati diametralmente opposti saranno per quanto possibile diversi. Pensate al comportamento di un cane o di un gatto arrabbiato piuttosto che contento, e comprenderete esattamente cosa si intenda. Ad ogni modo, questo non è sempre vero, e alcuni animali tendono ad utilizzare dei display "mimetici" come tecniche di caccia o di difesa, per disorientare una preda o un aggressore. Purtroppo, questa tende ad essere la norma nel campo dell'Intrusion Detection. Utenti (quasi) innocui notoriamente riescono a comportarsi in modo bizzarro, mentre i più pericolosi aggressori faranno di tutto per cercare di nascondersi dietro una sequenza di attività apparentemente innocenti.

Alcuni studi suggeriscono [124] una profonda correlazione tra questo circolo virtuoso di comunicazione e l'evoluzione di sistemi di linguaggio. Anche se c'è una differenza abissale tra il nostro linguaggio (o quello dei computer) e le forme più semplici di comunicazione animale (i linguaggi sono prevalentemente forme di comunicazione *digitali* mentre la comunicazione animale è tipicamente *analogica*) c'è un cammino evolutivo ben delineato tra le due cose, promosso da un potente feedback positivo tra l'aumento di complessità della comunicazione, l'evoluzione di meccanismi di pensiero e coscienza, e l'evoluzione di una struttura sociale complessa. Questo tipo di analisi suggerirebbe anche una prospettiva interessante, secondo la quale le complesse comunicazioni e relazioni sociali presenti all'interno dei branchi e degli sciami di animali possono essere considerate le manifestazioni di una sorta di intelligenza distribuita. Ma stiamo divagando, e quindi ci fermiamo qui.

Ci limitiamo a far osservare come, interpretando i display come componenti di un linguaggio, e le sequenze di display come "frasi", ci troviamo a dover considerare che il nostro problema di rilevamento dei display, analisi degli stessi, ed elaborazione di ipotesi sulle motivazioni comportamentali, come analogo al problema di comprensione del linguaggio naturale, composto di esplorazione simbolica, analisi sintattica ed analisi semantica. Questo è indice della profondità del problema che ci siamo posti.

INDIVIDUAZIONE DEI COMPORTAMENTI

DEFINIZIONE DEL PROBLEMA

Cercheremo di sfruttare le analogie che abbiamo evidenziato, per applicare alcuni dei metodi dell'etologia al problema di *individuazione dei comportamenti* nello specifico settore della *intrusion detection*

È importante però sottolineare che questa classe di problemi si estende ben oltre il dominio della sicurezza informatica, e comprende tutte le situazioni in cui è importante studiare il comportamento di un soggetto dotato di autonomia decisionale e classificarlo. In alcuni problemi relativamente semplici, abbiamo solo due categorie tra cui distinguere, di solito "normale" ed "anomalo". In altri casi abbiamo una classificazione estesa, per esempio se vogliamo personalizzare il contenuto di un sito web in base al comportamento tipico di un utente; oppure nel problema molto studiato degli ambienti collaborativi di tipo *virtual classroom*¸ in cui vogliamo essere in grado di classificare il comportamento degli studenti in modo ampio e complesso, e spesso addirittura definibile da parte dell'utente finale del sistema (ovvero l'insegnante). In questi casi possiamo avere anche situazioni di intersezione tra varie classi. Per esempio, uno studente potrebbe essere sia "rapido nell'apprendere" che "timido nell'interazione sociale".

UNA PROPOSTA METODOLOGICA

Per realizzare un sistema di individuazione comportamentale dobbiamo affrontare una serie di sottoproblemi distinti. Innanzitutto, dobbiamo analizzare i *display* a nostra disposizione, eventualmente escludere quelli che non paiono attinenti, e costruire dei *sensori* adeguati per rilevarli e catalogarli.

Ad esempio, nel problema dell'intrusion detection potrebbe essere semplice raccogliere ed analizzare i dati di rete, piuttosto che i log di una workstation (più difficile invece, come abbiamo già avuto modo di osservare, garantire che tali dati non siano stati contraffatti o alterati); viceversa, individuare dei *display* significativi nell'osservazione di una classe virtuale può essere molto più problematico.

In secondo luogo, dobbiamo definire accuratamente il tipo di risultati che vogliamo: se ci interessa una classificazione netta tra "positivo" e "negativo" (anomalia/normalità), oppure se vogliamo una distinzione in classi multiple. In questo ultimo caso dobbiamo anche stabilire se tali classi sono disgiunte o possono avere delle intersezioni, e se vogliamo un sistema che dia in uscita "probabilità di appartenenza" piuttosto che certezze. In quest'ultimo caso potrebbero esserci delle opzioni *fail-safe*, ad esempio nell'incertezza se uno studente vada inserito in una classe ad un livello superiore o meno, potremmo decidere di lasciarlo in quella più semplice. Oppure, per non essere inondati da segnalazioni di un IDS, potremmo decidere che in caso di incertezza la scelta ricada su "comportamento normale".

Come terzo dato, dobbiamo esaminare la conoscenza di dominio che abbiamo a disposizione. In etologia, ad esempio, ci sono pochissime conoscenze di dominio su cosa o come un animale "pensi". Viceversa, sul comportamento degli intrusi nei

sistemi informatici abbiamo ampie conoscenze. Sui metodi di insegnamento possibili ci sono molte discussioni, è vero, ma in generale abbiamo anche qui approfondite conoscenze di dominio. Dalla risposta a questa domanda dipende anche la costruzione di un sistema più o meno "supervisionato": se le conoscenze sono poche, o contraddittorie, si cercherà di far costruire alla macchina un modello. Se viceversa abbiamo conoscenze, possiamo pensare di fornirgli già un modello di base sul dominio

Il passaggio finale ci porterà ad integrare le informazioni ricavate nei passaggi precedenti per costruire uno schema deduttivo di ipotesi che potranno essere costruite sopra i dati. Avremo bisogno di esso come riferimento durante la costruzione del sistema definitivo, che tuttavia potrebbe seguire uno schema diverso nei suoi processi deduttivi interni.

Proviamo ad applicare questo metodo alla creazione di un sistema di IDS "concettuale" basato sui comportamenti.

PASSO UNO: COSA OSSERVARE ?

I *display* del comportamento di un utente su un sistema informatico sono sostanzialmente le interazioni tra lui ed il sistema stesso. Queste interazioni avvengono, oggi, principalmente attraverso la rete, ma possono anche avvenire localmente al sistema stesso: pertanto le due fonti di dati sono (come abbiamo già avuto modo di osservare estensivamente) i pacchetti di traffico di rete e i log degli eventi di sistema.

Il vero problema di questi meccanismi di collezione di dati è la normalizzazione. Dati provenienti da fonti eterogenee sono diversi strutturalmente, non sono ordinati, e non sono, in generale, correlati direttamente ad uno specifico utente. Per questo devono venire compiute tutte le analisi di cui abbiamo già sottolineato le difficoltà, per cercare di normalizzare i dati acquisiti in un elenco, strutturalmente omogeneo, di *interazioni tra utenti e macchine*.

PASSO DUE: LE CATEGORIE

Per il nostro sistema di Intrusion Detection vogliamo semplificarci la vita al massimo, ed intendiamo distinguere tra comportamento *normale* e comportamento *anomalo* dell'utente. Vogliamo una distinzione netta, e che tipicamente (per evitare una valanga di falsi positivi) segnali un'intrusione solo quando c'è un notevole scostamento dal comportamento normale dell'utente

PASSO TRE: CONOSCENZE DI BASE SUL DOMINIO

Abbiamo già affrontato questi argomenti nel corso dell'esposizione, non ci dilunghiamo ulteriormente, rimandando al capitolo 2 del lavoro di tesi per tutte le informazioni.

Come si è fatto notare, il numero pur ampio di informazioni sul dominio delle tecniche di intrusione non si traduce, automaticamente, in conoscenze riproducibili in forma automatica sul riconoscimento degli intrusi.

ETOGRAMMI

Gli etologi hanno approcciato il problema della descrizione ed analisi dei comportamenti creando i cosiddetti "etogrammi". Un etogramma è un tentativo di descrivere correttamente e completamente l'insieme dei comportamenti di una specie.

Le osservazioni del comportamento degli animali sono compiute sul campo dai ricercatori, spesso con l'assistenza di software inferenziale appositamente creato [125]. I risultati vengono riassunti negli etogrammi, che non sono altro che cataloghi di comportamenti (FAP) tipicamente osservati, accompagnati dalla probabilità di tali comportamenti, e da osservazioni preliminari su possibili interpretazioni del loro significato.

Per esempio, una delle voci di un etogramma sviluppato per i gorilla [126] dice: "*Display* intimidatorio: riposarsi proni, battere sul petto, corsa breve ed esagerata, estroflessione delle labbra, e/o lancio di oggetti. Viene segnato quando due di questi comportamenti sono presenti contemporaneamente."

Un ricercatore osserverebbe a intervalli fissi una popolazione di gorilla, e segnerebbe in un etogramma i loro comportamenti, generando una distribuzione di probabilità e dati relativi alla sequenza del comportamento osservato.

CATENE COMPORTAMENTALI E MODELLI DI MARKOV

Nella ricerca sul campo in etologia viene fatto ampio uso di modelli e metodi statistici, tra cui le cosiddette "catene comportamentali" (behavior chains), che non sono altro che modelli a catena di Markov.

Il successo di questo modello è dovuto al fatto che esso è uno strumento semplificato di data mining che può essere applicato, con poca fatica, ai dati grezzi registrati su un etogramma in base alle osservazioni sul campo. Ci sono anche un discreto numero di pacchetti software che assistono nella costruzione di modelli di Markov (anche nascosti) a partire dai dati [127].

Un modello di Markov è costituito da un insieme finito di stati. Le transizioni tra questi stati sono governate da un insieme di probabilità dette probabilità di transizione. Un modello di Markov è detto del primo ordine se le probabilità di transizione da uno stato ad un altro dipendono solo dalla coppia di stati considerati, ovvero se le probabilità degli stati futuri dipendono completamente dallo stato presente. Modelli di ordine superiore hanno *memoria* degli stati attraversati in precedenza. È abbastanza evidente che molti sistemi del mondo reale saranno modellati meglio da un modello di ordine superiore. Tuttavia, la semplicità matematica dei modelli del primo ordine (detti anche *catene*) li rende utili come prima approssimazione, e in generale applicabili quando la ricerca che si sta compiendo non ha pretese di esattezza (come nel nostro caso).

Un modello di Markov nascosto (Hidden Markov Model, HMM) ha la stessa forma di un modello tradizionale, ma l'osservatore non è a conoscenza direttamente dello stato corrente. Viceversa, l'osservatore percepisce una o più *osservazioni*, che sono funzioni o più generalmente variabili aleatorie che vengono generate da una particolare funzione di probabilità associata con lo stato attuale.

Come si riconosce un sistema che può essere modellato con un modello di Markov, eventualmente nascosto ? Un processo nel dominio del tempo deve esibire la cosiddetta proprietà di Markov: la probabilità condizionata dell'evento corrente, dati tutti gli eventi presenti e passati, deve dipendere solo dai K eventi più recenti. Se K vale 1, il processo è modellabile con una catena di Markov del primo ordine. Ma se K ha un valore più alto, il modello del primo ordine può, generalmente, essere una buona approssimazione.

Per descrivere un HMM di ordine 1, dobbiamo definire:

- Il numero di stati del processo nascosto: N
- Il numero di simboli osservabili nell'alfabeto: M. Se le osservazioni sono continue, M è infinito.
- L'insieme delle probabilità di transizione, che è una matrice $\Lambda = \{a_{i,j}\}$ dove $a_{i,j} = p\{q_{t+1} = j \mid q_t = i\}, 1 \le i, j \le N$, dove q_t rappresenta lo stato all'istante t. Ovviamente, $a_{i,j} \ge 0,\ 1 \le i,j \le N$ e $\sum_{j=1}^{N} a_{i,j} = 1, 1 \le i \le N$. Usare un modello di ordine K significherebbe dover usare una matrice multidimensionale di

profondità k+1 per Λ. Questo è abbastanza improponibile, al di fuori della teoria. Il modello è stazionario, ovvero le funzioni di transizione di stato non dipendono dal tempo.

- Abbiamo poi la distribuzione di probabilità per i simboli osservati, per ogni stato, ovvero disponiamo delle probabilità discrete $B = \{b_j(k)\}$, tali che $b_j(k) = p\{o_t = v_k \mid q_t = j\}, 1 \le j \le N, 1 \le k \le M$ dove v_k è il k-esimo simbolo possibile nell'alfabeto e o_t è l'osservazione all'istante t. Ovviamente $b_j(k) \ge 0, 1 \le j \le N, 1 \le k \le M$ e $\sum_{k=1}^{M} b_j(k) = 1, 1 \le j \le N$.
- Abbiamo anche una probabilità di distribuzione iniziale, $\pi = \{\pi_i\}$ dove $\pi_i = p\{q_1 = i\}, 1 \le i \le N$

In notazione compatta, un HMM con probabilità discreta può essere indicato come $\lambda = (\Lambda, \mathrm{B}, \pi)$.

Per utilizzare funzioni d'osservazione continue, dovremo usare una funzione di densità di probabilità continua, invece di un insieme di probabilità discrete, di solito approssimandola con una somma pesata mediante dei coefficienti c_{jm} di M distribuzioni gaussiane: $b_j(o_t) = \sum_{m=1}^{M} c_{jm} H(\mu_{jm}, \Sigma_{jm}, o_t)$, in cui ovviamente $c_{jm} \ge 0, 1 \le j \le N, 1 \le m \le M$ e anche $\sum_{k=1}^{M} c_{jm} = 1, 1 \le j \le N$. In questa rappresentazione μ_{jm} sono I vettori delle medie, e Σ_{jm} le matrici di covarianza. Un modello del genere avrebbe dunque come parametri: $\lambda = (\Lambda, c_{jm}, \mu_{jm}, \Sigma_{jm}, \pi)$.

Si comprenderà ora l'utilità, per gli etologi ma anche per noi, di discretizzare il comportamento continuo dei soggetti osservati. Gli etologi utilizzano i FAP e i MAP. Noi potremmo intuitivamente utilizzare i comandi di una user session, o qualche altra forma di discretizzazione delle osservazioni.

I problemi basilari che possiamo riscontrare nell'uso di modelli HMM sono:

1) problema di **valutazione**: data una sequenza di osservazioni, $O = \{o_1, ...o_T\}$ ed un modello, $\lambda = (\Lambda, \mathrm{B}, \pi)$, qual è la probabilità che la sequenza osservata sia stata generata proprio da tale modello, $P\{O \mid \lambda\}$?
2) Problema di **decodifica**: data una sequenza di osservazioni $O = \{o_1, ...o_T\}$ e un modello $\lambda = (\Lambda, \mathrm{B}, \pi)$, quale sequenza di stati può aver più probabilmente generato tale sequenza ?

3) Problema di **apprendimento**: data una sequenza di osservazioni $O = \{o_1,...o_T\}$ ed un modello $\lambda = (\Lambda, \mathrm{B}, \pi)$, come possiamo adattare i parametri di tale modello per massimizzare la probabilità che la sequenza sia stata generata dal modello stesso ?

Ovviamente il problema del primo tipo si riscontra mentre cerchiamo di stabilire a quale classe di comportamento far risalire una sequenza di *display*, mentre un problema del terzo tipo è tipico della fase di *costruzione* e taratura del modello.

Il problema di valutazione non è semplice quanto sembra. Una soluzione ingenua sarebbe quella di calcolare $P\{O \mid \lambda\}$ sommando le probabilità su tutte le possibili sequenze di stati nel modello per la sequenza di T osservazioni: $P\{O \mid \lambda\} = \sum_{i=1}^{N} \prod_{j=1}^{T} a_{ij} b_j(o_t)$. Questo richiederebbe $O(N^T)$ operazioni, che può essere un valore inaffrontabile anche per valori piccoli di T

Esiste tuttavia un metodo, noto come algoritmo *forward*, di complessità $O(N^2 T)$ che rende calcolabile tale probabilità. Si usa una variabile ausiliaria, $\alpha_t(i)$, detta variabile di forward, e definita come la probabilità della sequenza parziale di osservazioni $o_1,...,o_t$, se terminasse all'istante t sullo stato i. Matematicamente, $\alpha_t(i) = p\{o_1,...o_t; q_t = i \mid \lambda\}$.

Vale per essa, banalmente, la relazione ricorsiva:

$$\alpha_{t+1}(j) = b_j(o_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{ij}, 1 \le j \le N, 1 \le t \le T-1$$

Ora, siccome: $\alpha_1(j) = \pi_j b_j(o_1), 1 \le j \le N$, usando la ricorsione calcoliamo $\alpha_T(i), 1 \le i \le N$, e la probabilità desiderata è proprio $P\{O \mid \lambda\} = \sum_{i=1}^{N} \alpha_T(i)$

Non discutiamo la soluzione completa del terzo problema, ma accenniamo che si utilizza una variabile detta *backward,* $\beta_t(i)$ che è la probabilità della sequenza di osservazioni parziali $o_{t+1},...,o_T$, se lo stato corrente è *i*. Matematicamente, $\beta_t(i) = p\{o_{t+1},...o_T \mid q_t = i, \lambda\}$. Esiste anche per essa una "scorciatoia" ricorsiva:

$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(o_{t+1}), 1 \le i \le N, 1 \le t \le T-1$, e dal momento che

$\alpha_t(i)\beta_t(i) = p\{O, q_t = i \mid \lambda\}, 1 \le i \le N, 1 \le t \le T$, possiamo calcolare

$P\{O \mid \lambda\} = \sum_{i=1}^{N} p\{O, q_t = i \mid \lambda\} = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i)$. Da questa formula si possono derivare

le formule utili per un "addestramento" del modello sulla base del gradiente. Tutto ciò sta alla base del famoso algoritmo di *Viterbi*.

CONCLUSIONI: UNA PROPOSTA DI MODELLO

Abbiamo dunque proposto un modello "concettuale" di sistema di rilevazione dei comportamenti degli utenti, che passa per l'individuazione di una serie di "azioni" (display), probabilmente le più frequenti (statisticamente) tra quelle rilevate in un ampio periodo di osservazione dai nostri sensori, opportunamente normalizzati.

Questo sistema, sfruttando le caratteristiche dei modelli di Markov nascosti, potrebbe essere addestrato, creando una serie di modelli corrispondenti a dei tipici comportamenti (o forse, più plausibilmente, a tipici comportamenti di specifici utenti). In seguito, per ogni sequenza di osservazioni, il sistema sarebbe in grado di determinare la probabilità statistica che essa sia stata generata dal profilo dell'utente considerato.

Come è stato fatto notare nell'ambito del lavoro di tesi, utilizzi più "ingenui" di modelli di Markov non nascosti sono stati già proposti ed esaminati nell'ambito di svariate ricerche sui sistemi IDS. La differenza della nostra proposta consiste nel rilevare, sulla scorta delle osservazioni degli etologi, che non necessariamente i *display* sono spiegabili con un modello Markoviano semplice, mentre si prestano molto più naturalmente all'uso di modelli Markoviani nascosti.

BIBLIOGRAFIA

[1] J. P. Anderson, *"Computer Security Threat Monitoring and Surveillance"*, technical report, James P. Anderson Company, Fort Washington, Pennsylvania (Aprile 1980)

[2] Department of Defense, "*Trusted Computer System Evaluation Criteria"*, (Dicembre 1985)

[3] V. D.Gligor, *"A Note on the Denial-of-Service Problem*", Proceedings of the IEEE Symposium on Security and Privacy, pag. 139 –149 (1983)

[4] E. G. Amoroso, *"Fundamentals of Computer Security Technology"*, Prentice-Hall PTR (1994)

[5] D. E. Bell, L. J. LaPadula, *"Secure Computer Systems: Unified Exposition and Multics Interpretation"*, technical report, Mitre TR-2997, Mitre Corporation, Bedford, MA (Marzo 1976)

[6] D. B. Baker, *"Fortresses Built Upon Sand",* in New Security Paradigms Workshop, pag. 148 –153 (Settembre 1996)

[7] O. Dahl, E. Dijkstra, C. A. Hoare, *"Structured Programming"*, Academic Press, London and New York (1972)

[8] B. P. Miller, L. Fredrikson, B. So, *"An empirical study of the reliability of UNIX utilities"*, Communications of the ACM, 33(12):32-44 (Dicembre 1990)

[9] P. G. Neumann, L. Robinson, K. N. Levitt, R. S. Boyer, A. R. Saxena, *"A Provably Secure Operating System"*, Technical Report SRI Project 2581, Stanford Research Institute (Giugno 1975)

[10] J. R. Williams, M. Schaefer, D. J. Landoll, *"Pretty Good Assurance"*, New Security Paradigms Workshop, pag. 82 –89, Arca Systems (Settembre 1996)

[11] http://online.securityfocus.com/sfonline/vulns/stats.shtml

[12] B. Schneier, *"Full Disclosure and the Window of Exposure"*, CryptoGram newsletter, n. 9 (15 settembre 2000) http://www.counterpane.com/crypto-gram-0009.html

[13] Si veda, a titolo d'esempio, il materiale dell'iniziativa "Human Firewall" su www.humanfirewall.org

[14] Computer Security Institute – Federal Bureau of Investigations, *"The 2002 CSI/FBI computer crime and security survey"*, disponibile online, http://www.gocsi.com/press/20020407.html

[15] Il resoconto, un po' di parte in verità, si può trovare in: T. Shimomura, J. Markoff, *"Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw - By the Man Who Did It*", Hyperion, New York (1996)

[16] S. Garfinkel, G. Spafford, *"Web Security &Commerce"*, O'Reilly (1997)

[17] D. L. Lough, *"A taxonomy of computer attacks with applications to wireless networks",* tesi di dottorato, Virginia Polytechnic Institute and State University, (Aprile 2001)

[18] J. D. Howard, *"An Analysis Of Security Incidents On The Internet, 1989 – 1995",* tesi di dottorato, università di Pittsburgh (7 aprile 1997) http://www.cert.org/research/JHThesis/Start.html

[19] Sun Tzu, *"L'arte della guerra"*, SuperBUR Classici, RCS Milano (Aprile 1999)

[20] M. Strano, *"Computer crime"*, edizioni Apogeo, Milano (2000)

[21] D. A. Wheeler, *"Secure Programming for Linux and Unix HOWTO"* http://www.dwheeler.com/secure-programs

[22] E. "Aleph1" Levy, *"Smashing the stack for fun and profit"*, Phrack magazine, vol. 7, issue 49 (Novembre 1996)
http://www.phrack.org/phrack/49/P49-14

[23] http://www.sans.org/top20

[24] E. H. Spafford, K. A. Heaphy, D. J. Ferbrache, *"A Computer Virus Primer"*, in *"Computers Under Attack: Intruders, Worms, and Viruses*", Peter J. Denning, ed., pp. 316-355, ACM Press, New York (1990)

[25] http://project.honeynet.org/

[26] A. Ghirardini, *"Social Engineering: una guida introduttiva"*, technical white paper #3, Italian Black Hats Association
http://www.blackhats.it/it/papers/social_engineering.pdf

[27] B. Mukherjee, L.T. Heberlein, K.N. Levitt, "*Network Intrusion Detection*", IEEE Network, Maggio/Giugno 1994

[28] S. Axelsson, *"Intrusion Detection Systems: A Survey and Taxonomy"*, Technical Report 99-15, Depart. of Computer Engineering, Chalmers University (Marzo 2000)

[29] The Honeynet Project,*"Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community"*, Addison-Wesley (Agosto 2001)

[30] NEPED del gruppo di hacking spagnolo Apostols, www.apostols.org, successivamente migliorato dal gruppo italiano s0ftpj in PROscan:
http://www.s0ftpj.org/tools/proscan.c

[31] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, *"A Network Security Monitor*", Proceedings of the IEEE Symposium on Research in Security and Privacy, pag. 296-304 (Maggio 1990)
http://seclab.cs.ucdavis.edu/papers/pdfs/th-gd-90.pdf

[32] T. Bass, *"Intrusion detection systems & multisensor data fusion: Creating cyberspace situational awareness"* , Communications of the ACM, Aprile 2000

[33] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, T. Grance, L. T. Heberlein, Che-Lin Ho, K. N. Levitt, B. Mukherjee, D. L. Mansur, K. L. Pon, S. E. Smaha *"A system for distributed intrusion detection"*, COMPCOM Spring '91 Digest of Papers, pag. 170-176 (Febbraio/Marzo 1991)

[34] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isaco, E. H. Spafford, D. Zamboni, *"An Architecture for Intrusion Detection Using Autonomous Agents*", Proceedings of ACSAC '98, pag. 13-24 (1998)

[35] http://www.intrusion.com/products/productcategory.asp?lngCatId=10

[36] http://www.sikurezza.org:8000/angel/

[37] V. Paxson, *"Bro: A System for Detecting Network Intruders in Real-Time"*, 7th Annual USENIX Security Symposium (Gennaio 1998) http://citeseer.nj.nec.com/article/paxson98bro.html

[38] http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz

[39] T. H. Ptacek, T. N. Newsham, "*Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection"*, Technical Report, Secure Networks (Gennaio 1998) http://citeseer.nj.nec.com/ptacek98insertion.html

[40] Ad esempio: http://www-rnks.informatik.tu-cottbus.de/~sobirey/ids.html

[41] T. F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D. L. Edwards, H. S. Javitz, A. Valdes, *"IDES: The Enhanced Prototype - A Real-Time Intrusion-Detection Expert System"*, technical report SRI-CSL-88-12, SRI International, Menlo Park, CA (Ottobre 1988) http://www.sdl.sri.com/papers/csl-88-12/

[42] J. R. Winkler, "A UNIX Prototype for Intrusion and Anomaly Detection in Secure Networks", Proceedings of the 13th National Computer Security Conference, pag. 115-124 Washington D. C. (Ottobre 1990)

[43] M. M. Sebring, E. Sellhouse, M. E. Hanna, R. A. Whitehurst, *"Expert system in intrusion detection: A case study"*, Proceedings of the 11th National Computer Security Conference, pag. 74-81, Baltimora (Ottobre 1988)

[44] H. S.Vaccaro, G. E. Liepins, *"Detection of anomalous computer session activity"*, Proceedings of the Symposium on Research in Security and Privacy, Oakland, pag. 280-289 (Maggio 1989)

[45] M. Roesch, *"Snort —Lightweight intrusion detection for networks"*, Proceedings of LISA '99 13th USENIX Systems Administration Conference, Seattle (Novembre 1999)

[46] C. Kruegel, T. Toth, *"Automatic rule clustering for improved signature-based intrusion detection"*, Technical Report http://www.infosys.tuwien.ac.at/snort-ng/snort-ng.pdf

[47] T. Lane, C. E. Brodley, *"Sequence matching and learning in anomaly detection for computer security"* AAAI Workshop: AI Approaches to Fraud Detection and Risk Management, pag. 43-49, AAAI Press (Luglio 1997). Espanso in una tesi di dottorato: T.D. Lane, *"Machine Learning Techniques For The Computer Security Domain Of Anomaly Detection"*, Purdue University (1998)

[48] M. Theus, M. Schonlau, *"Intrusion Detection Based on Structural Zeroes"*, Statistical Computing & Graphics Newsletter, Vol. 9, Nr. 1, pag. 12-17, American Statistical Association (1998) http://citeseer.nj.nec.com/theus98intrusion.html

[49] J. Ryan, M. Lin, R. Miikkulainen, *"Intrusion Detection with Neural Networks"*, in Advances in Neural Information Processing Systems 10, M. Jordan e altri, editors, pag. 943-949, MIT Press (1998)

[50] L. Mè, *"Security audit trail analysis using genetic algorithms"*, Proceedings of the 12th International Conference on Computer Safety, Reliability and Security, pag. 329 – 340, Poznan, Poland, (Ottobre 1993)

[51] Tra tutti i lavori che gli autori hanno pubblicato sul tema, suggeriamo: W. Lee, S. J. Stolfo, K. W. Mok, *"Adaptive Intrusion Detection: A Data Mining Approach"*, Artificial Intelligence Review Issues on the Application of Data Mining, 14:533-567, Kluwer (2000)

[52] Uno dei migliori testi in proposito è: J. Cannady, *"Artificial neural networks for misuse detection"*, Proceedings of the 1998 National Information Systems

Security Conference (NISSC'98), pag. 443-456, (Ottobre 1998)
http://citeseer.nj.nec.com/cannady98artificial.html

[53] S. Hofmeyr, S. Forrest, *"Architecture for an Artificial Immune System"* Evolutionary Computation 7(1), Morgan-Kaufmann, pp. 1289-1296 (2000), http://cs.unm.edu/~forrest/publications/hofmeyr_forrest.pdf

[54] L. Deri, *"Ntop: a Lightweight Open-Source Network IDS"*, technical report, (Settembre 2000)
http://luca.ntop.org/LightweightIDS.pdf

[55] L. Deri, S. Suin, G. Maselli, *"Design and Implementation of an Anomaly Detection System: an Empirical Approach"*, technical report
http://luca.ntop.org/ADS.pdf

[56] http://www.ll.mit.edu/IST/ideval/data/data_index.html

[57] K. Kendall, *"A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems"*, Tesi di Master of Science, Massachusetts Institute of Technology (1998)

[58] K. M. C. Tan, B. S. Collie, *"Detection and Classification of TCP/IP Network Services"*, Proceedings of the Computer Security Applications Conference, pag. 99-107 (1997)

[59] J. A. Hartigan, *"Clustering Algorithms"*, Wiley (1975)

[60] J. Han, M. Kamber, *"Data Mining: concepts and techniques"*, Morgan-Kauffman (2000)

[61] J. Frank, *"Artificial intelligence and intrusion detection: Current and future directions"*, Proceedings of the 17th National Computer Security Conference, (Ottobre 1994)
http://citeseer.nj.nec.com/frank94artificial.html

[62] D. Hawkins, *"Identification of Outliers"*, Chapman and Hall, London (1980)

[63] S. Breunig, H.-P. Kriegel, R. Ng, J. Sander, *"LOF: Identifying Density-Based Local Outliers"*, ACM SIGMOD International Conference on Management of Data, Dallas (2000)
http://citeseer.nj.nec.com/breunig00lof.html

[64] C. Aggarwal, P. Yu., *"Outlier detection for high dimensional data"*, ACM SIGMOD Conference (2001)
http://citeseer.nj.nec.com/aggarwal01outlier.html

[65] A. K. Jain, M. N. Murty, P. J. Flynn, "*Data Clustering: A Review"*, ACM Computing Surveys, Vol. 31, Nr. 3, pag. 264-323 (1999)

[66] Z. Huang, *"Extensions to the k-means algorithm for clustering large data sets with categorical values"*, Data Mining and Knowledge Discovery, vol.2, no.3, p. 283-304, Kluwer Academic Publishers 1998

[67] A. Likas, N. Vlassis, J. J. Verbeek, *"The Global K-Means Clustering Algorithm"*, comparirà su Pattern Recognition 36(2) (2003)

[68] T. Kohonen, *"Self-Organizing Maps"*, 3a edizione, Springer-Verlag, Berlino (2001)

[69] A. Rauber, *"LabelSOM: On the labeling of selforganizing maps"*, Proceedings of the International Joint Conference on Neural Networks, Washington D .C. (1999)
http://citeseer.nj.nec.com/rauber99labelsom.html

[70] L. Kaufman, *"Finding groups in data: an introduction to cluster analysis"*, Wiley, New York (1990)

[71] R. T. Ng, J. Han, *"Efficient and Effective Clustering Methods for Spatial Data Mining"*, Proceedings of the 20th VLDB Conference, pag. 144-155, Santiago (1994)
http://citeseer.nj.nec.com/ng94efficient.html

[72] T. Zhang, R. Ramakrishnan, M. Livny: *"BIRCH: An Efficient Data Clustering Method for Very Large Databases"*, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pag. 103-114, Montreal (1996)

[73] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, J. French *"Clustering Large Datasets in Arbitrary Metric Spaces"*, Proceedings of the 15th International Conference on Data Engineering (ICDE'99), Sydney (Marzo 1999)
http://www.cs.virginia.edu/~cyberia/papers/ICDE99.pdf

[74] S. Guha, R. Rastogi, K. Shim, *"CURE: an efficient clustering algorithm for large databases"*, ACM SIGMOD Record, vol. 27, nr. 2, pag. 73-84 (Giugno 1998)

[75] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, *"Automatic subspace clustering of high dimensional data for data mining applications"*, Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data, Seattle (Giugno 1998)
http://citeseer.nj.nec.com/agrawal98automatic.html

[76] R. A. Reckhow, J. Culberson, *"Covering simple orthogonal polygon with a minimum number of orthogonally convex polygons*" Proceedings of the ACM 3rd Annual Computational Geometry Conference, pag. 268-277 (1987)

[77] U. Feige, *"A threshold of ln(n) for approximating set cover"*, Proceedings of the 28th Annual ACM Symposium on Theory of Computing, pag. 314-318 (1996)

[78] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, J. S. Park, *"Fast algorithms for projected clustering"*, Proceedings of ACM SIGMOD International Conference on the Management of Data, pag. 61-72 (1999)
http://citeseer.nj.nec.com/aggarwal99fast.html

[79] C. C. Aggarwal, P. S. Yu, *"Finding generalized projected clusters in high dimensional spaces"*, Proceedings of the ACM SIGMOD International Conference on the Management of Data, pag. 70–81 (2000)

[80] H. Nagesh, S. Goil, A. Choudhary, *"MAFIA: Efficient and scalable subspace clustering for very large data sets"*, Technical Report 9906-010, Northwestern University (Giugno 1999)

[81] A. Hinneburg, D. A. Keim, *"Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering"*, Proceedings of the 25th International Conference on Very Large Data Bases, pag. 506–517 (1999)

[82] W. Wang, J. Yang, R. Muntz, *"STING: A statistical information grid approach to spatial data mining"*, Proceedings of the 23rd International Conference on Very Large Data Bases, pag. 186-195, Atene (1997)
http://citeseer.nj.nec.com/wang97sting.html

[83] G. Sheikholeslami, S. Chatterjee, A. Zhang, *"WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases"*, Proceedings of the 24th International Conference on Very Large Data Bases, pag. 428-439, New York (1998)
http://citeseer.nj.nec.com/sheikholeslami98wavecluster.html

[84] E. H. Han, G. Karypis, V. Kumar, B. Mobasher, *"Clustering in a high-dimensional space using hypergraph models"*, Technical Report TR-97-063, Department of Computer Science, University of Minnesota, Minneapolis (1997)
http://citeseer.nj.nec.com/article/han97clustering.html

[85] P. Cheeseman, J. Stutz, *"Bayesian Classification (AutoClass): Theory and Results"*, in Advances in Knowledge Discovery and Data Mining, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, editors (1996)
http://citeseer.nj.nec.com/cheeseman96bayesian.html

[86] D. E. Goldberg, *"Genetic Algorithms in Search, Optimization and Machine Learning"*, Addison-Wesley (1989)

[87] D. Jones, M. A. Beltramo, *"Solving partitioning problems with genetic algorithms"*, Proceedings of the Fourth International Conference on Genetic Algorithms, 442–449 (1991)

[88] D. Whitley, T. Starkweather, D. Fuquay, *"Scheduling problems and traveling salesman: the genetic edge recombination"*, Proceedings of the Third International Conference on Genetic Algorithms, pag. 133–140, Morgan Kaufmann (1989)

[89] S. K. Mishra, V. V. Raghavan, *"An empirical study of the performance of heuristic methods for clustering"*. In "Pattern Recognition in Practice", E. S. Gelsema, L. N. Kanal, editors, pag 425–436 (1994)

[90] G. A. Carpenter, S. Grossberg, *"ART2: Self-organization of stable category recognition codes for analog input patterns"*, Applied Optics, vol. 26, nr. 23, pp. 4919–4930 (1987)

[91] J. R. Williamson, *"Gaussian ARTMAP: A neural network for fast in-cremental learning of noisy multidimensional maps"*, Neural Networks, vol. 9, nr. 5, pag. 881–897 (1996)

[92] A. Baraldi, E. Alpaydın, *"Constructive Feedforward ART Clustering Networks—Part II"*, IEEE Transactions on neural networks, vol. 13, nr. 3 (Maggio 2002)

[93] D. A. Patterson, J. L. Hennessy, *"Computer Organization and Design: The Hardware/software Interface"*, Morgan-Kaufmann (1994)

[94] www.tcpdump.org

[95] windump.polito.it e winpcap.polito.it

[96] www.ethereal.com

[97] J. Postel, *"RFC 791: Internet Protocol"* (1981)
ftp://ftp.rfc-editor.org/in-notes/rfc791.txt

[98] J. Postel, *"RFC 793: Transmission Control Protocol"* (1981)
ftp://ftp.rfc-editor.org/in-notes/rfc793.txt

[99] J. Postel, *"RFC 768: User Datagram Protocol"* (1980)
ftp://ftp.rfc-editor.org/in-notes/rfc768.txt

[100] J. Postel, *"RFC 792: Internet Control Message Protocol"* (1981)
ftp://ftp.rfc-editor.org/in-notes/rfc792.txt

[101] I. Morlini, *"Multivariate outliers detection with Kohonen networks: an useful tool for routine exploration of large data sets"*, NTTS'98 (1998)
http://europa.eu.int/en/comm/eurostat/research/conferences/ntts-98/

[102] Growing Hierarchical SOM Toolbox for Matlab
http://www.erg.abdn.ac.uk/users/alvin/ghsom/index.html

[103] A. Rauber, *"LabelSOM: On the labeling of selforganizing maps"*, Proceedings of the International Joint Conference on Neural Networks, Washington D.C. (1999)
http://citeseer.nj.nec.com/rauber99labelsom.html

[104] D. Boley, V. Borst, M. Gini, *"An Unsupervised Clustering Tool for Unstructured Data"*, IJCAI 99 International Joint Conference on Artificial Intelligence, Stoccolma (Agosto 1999)

[105] S. M. Savaresi, D. L. Boley, S. Bittanti, G. Gazzaniga, *"Cluster Selection in Divisive Clustering Algorithms"*, Proceedings of the 2nd SIAM International Conference on Data Mining, Arlington (Aprile 2002)
http://www.siam.org/meetings/sdm02/proceedings/sdm02-18.pdf

[106] G. Golub, A. van Loan, *"Matrix Computations"*, John Hopkins University Press (1996)

[107] http://www.daimi.au.dk/~rmunk/PROPACK/propack.tar.gz

[108] M. Brand, *"Incremental singular value decomposition of uncertain data with missing values"*, Proceedings of the 2002 European Conference on Computer Vision, Springer Lecture Notes in Computer Science volume 2350 (2002)

[109] Qui seguiamo T. Lane, C.E. Brodley, *"Temporal Sequence Learning and Data Reduction for Anomaly Detection"*, ACM Transactions on Information and System Security, Vol. 2, Nr. 3, Pag. 295–331 (Agosto 1999)

[110] P. Lichodzijewski, A. N. Zincir-Heywood, M. I. Heywood, *"Dynamic Intrusion Detection Using Self Organizing Maps"*, 14th Annual Canadian Information Technology Security Symposium (Maggio 2002)

[111] K. Labib, R. Vemuri, *"NSOM: A Real-Time Network-Based Intrusion Detection System Using Self-Organizing Maps"*, Technical Report, Department of Applied Science, University of California, Davis

[112] L. Girardin, *"An Eye on Network IntruderAdministrator Shootouts"*, Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring (1999)
http://citeseer.nj.nec.com/girardin99eye.html

[113] M. V. Mahoney, P. K. Chan, *"Detecting Novel Attacks by Identifying Anomalous Network Packet Headers"*, Florida Institute of Technology Technical Report CS-2001-2 (2001)

[114] E. Eskin, *"Anomaly detection over noisy data using learned probability distributions"*, Proceedings of the Seventeenth International Conference on Machine Learning (2000)
http://citeseer.nj.nec.com/eskin00anomaly.html

[115] D.-Y. Yeung, C. Chow, *"Parzen-Window Network Intrusion Detectors"*, Proceedings of the 16th International Conference on Pattern Recognition, vol. 4, pag. 385-388, Quebec City (Agosto 2002) http://citeseer.nj.nec.com/505644.html

[116] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[117] K. Yamanishi, J. Takeuchi, G. Williams, *"On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms"*, Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pag. 320-324, Boston (Agosto 2000) http://citeseer.nj.nec.com/yamanishi00line.html

[118] www.whitehats.com/ids

[119] J. P. Yoo, C. C. Pettey, S. Yoo *"A hybrid conceptual clustering system"*, Proceedings of the 24th ACM annual conference on Computer science (1996)

[120] N. Tinbergen, *"The hierarchical organization of nervous mechanisms underlying instinctive behaviour"*, Symposium for the Society for Experimental Biology, 4, 305-312 (1950)

[121] T. Tyrell, *"Computational mechanisms for action selection"*, tesi di dottorato, University of Edinburgh (1993)

[122] A. K. Seth, *"Evolving Action Selection and Selective Attention without Actions, Attention, or Selection",* Technical Report, University of Sussex

[123] G. W. Barlow, *"Ethological units of behavior"*, in "The Central Nervous System and Fish Behavior", pp. 217-232, University of Chicago Press (1968)

[124] J. P. Marler, *"On the Origin of Speech from Animal Sounds"*, in "The Role of Speech in Language", J.F. Kavanagh, J. Cutting, editors, MIT Press (1975)

[125] Per esempio in GSEQ: http://www.gsu.edu/~psyrab/sg_e_gsw.htm

[126] *"Collection of Gorilla Ethograms"*, compilato a cura del Gorilla Behavior Advisory Group

[127] Ad esempio, il toolbox per Matlab sviluppato da Tapas Kanungo: http://www.cfar.umd.edu/~kanungo/software/software.html

oppure i file del Vision Project di Compaq Research:
http://crl.research.compaq.com/projects/vision/pmt.html