Università degli Studi di Modena

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E INFORMATICHE
Corso di Laurea in Informatica

Tesi di laurea

Analisi di un flow-based Intrusion Detection System basato su machine learning e sua estensione a formati di netflow differenti: il caso di Stratosphere

Candidato:
Michele Murgolo
Matricola 101851

Relatore:
Prof. Mirco Marchetti

Anno Accademico 2017-2018

Indice

Capitolo 1

Introduzione

La continua digitalizzazione nel mondo sta mettendo le aziende a rischio di attacchi informatici più che mai. Negli ultimi anni, grazie alla crescente adozione di servizi cloud e mobili, la sicurezza delle informazioni ha subito un profondo cambio di paradigma, influenzando sia i tradizionali strumenti di protezione (come antivirus e firewall), sia le piattaforme dedicate all'individuazione di attività dannose all'interno delle reti aziendali [58]. I metodi di attacco sempre più sofisticati utilizzati dai criminali informatici in diverse recenti violazioni della sicurezza su larga scala mettono chiaramente in mostra l'inadeguatezza dei tradizionali approcci per la sicurezza delle informazioni [53, 54]. Con attacchi sempre più avanzati e persistenti e il semplice fatto che ogni organizzazione deve proteggersi da molteplici varietà di attacchi mentre un aggressore ha bisogno solo di un tentativo riuscito per compromettere un'intera azienda, è necessario che le organizzazioni rivalutino, modifichino e aggiornino i propri sistemi di difesa. Lo stato dell'arte mette in evidenza come l'analisi dei dati costituisca un elemento chiave per migliorare drasticamente le performance dei meccanismi difensivi, che consente un aumento sensibile della rilevazione di azioni malevole [8].

Big data security analytics è l'approccio alla base di questo miglioramento del rilevamento. Il rilevamento deve essere in grado di identificare i modelli di utilizzo in continuo cambiamento, ed eseguire analisi complesse su una varietà di fonti di dati che vanno dai registri di server e applicazioni agli eventi di rete e alle attività degli utenti. Ciò richiede di eseguire analisi su grandi quantità di dati correnti e storici.

Negli ultimi anni è emersa una nuova generazione di soluzioni di analisi della sicurezza, in grado di raccogliere, archiviare e analizzare enormi quantità di dati [56, 57]. Questi dati vengono analizzati applicando vari algoritmi di correlazione per rilevare le anomalie e quindi identificare possibili attività dannose. L'industria ha finalmente raggiunto il punto in cui gli algoritmi di intelligenza artificiale per

l'elaborazione di dati su larga scala sono diventati accessibili utilizzando framework prontamente disponibili [55]. Ciò consente di combinare analisi storiche e in tempo reale e identificare nuovi incidenti che potrebbero essere correlati ad altri che si sono verificati in passato. Insieme a fonti di intelligence di sicurezza esterne [59] che forniscono informazioni aggiornate sulle ultime vulnerabilità, ciò può facilitare notevolmente l'identificazione di attacchi informatici in corso sulla rete [61].

È con l'intenzione di utilizzare queste tecnologie che viene presentato in questa tesi una suite di software per la rilevazione di intrusioni nella rete, che fanno uso dei più recenti meccanismi per i processi di cattura e analisi di enormi quantità di dati. Questa tesi metterà in particolare evidenza le difficoltà nella gestione delle diverse tipologie di formati richieste da queste tecnologie. Verranno sottolineate l'eterogeneità dei software che hanno sempre contraddistinto l'informatica e le soluzioni scelte per risolvere tali problemi. A questo scopo, è stato creato un algoritmo per la conversione del traffico di rete tra due formati differenti. Tale algoritmo ha richiesto la ricerca e lo sviluppo di soluzioni originali per il raggiungimento di un duplice obiettivo: (i) la preservazione dei dati durante le conversioni; (ii) l'ottenimento di una elevata efficienza in termini di velocità di esecuzione del programma. Infatti, la produzione di un algoritmo di conversione con perdita di dati (*lossy*) porterebbe inevitabilmente ad una riduzione delle performance di rilevazione. Allo stesso modo, la necessità di effettuare analisi in tempo reale (al fine di rilevare attacchi il prima possibile) richiede tempistiche di esecuzione il più rapide possibili. Per lo sviluppo della soluzione proposta, si è pertanto fatto ricorso ad un approccio di calcolo parallelo per sfruttare la potenza dei più recenti processori multi core, velocizzando notevolmente i tempi di analisi. Il programma prodotto è stato sottoposto ad un'ampia serie di esperimenti, volti a misurarne le performance di esecuzione. Tali esperimenti dimostrano che la parallelizzazione nell'esecuzione raggiunge uno speedup lineare grazie al quale è possibile convertire migliaia di files in pochi secondi. Infine, al fine di facilitare l'utilizzo del software analizzato, verranno anche spiegate in dettaglio tutte le procedure e operazioni necessarie per il corretto deployment della piattaforma di *Intrusion Detection System* (IDS) analizzata.

La parte rimanente di questa tesi è strutturata come segue.

Nel secondo capitolo verranno in primo luogo presentate le minacce provenienti dalla rete, con particolare enfasi sui tipi di attacchi su larga scala. Successivamente verrà presentato lo stato dell'arte dei sistemi di difesa utilizzati ad oggi per contrastare questi tipi di attacchi. In conclusione sarà fatta una introduzione al machine learning usato da questi sistemi.

Nel terzo capitolo verrà introdotto uno speciale tipo di file che sarà il principale punto di enfasi di questa tesi. Quindi verranno presentati i differenti tipi di formati

che questo file può assumere e le problematiche dovute ai diversi standard in uso oggi. Dopo una descrizione dettagliata delle varie differenze tra i formati verrà introdotto il software che farà uso di questi file e le problematiche legate all'utilizzo di software avanzati ma ancora in fase di alpha. Si presenterà poi l'utilizzo che questi file hanno in relazione alle tecnologie utilizzate. Infine troverà spazio la presentazione del problema da affrontare.

Nel quarto capitolo si descriveranno le scelte effettuate per risolvere il problema del deployment di un software non ancora pubblicato e come si è modificato il codice sorgente per poterlo adattare alle proprie esigenze. Successivamente verrà analizzato l'uso di diversi formati di file e l'automatizzazione di tale soluzione. Infine saranno presentati i diversi metodi atti a perfezionare l'automatizzazione per renderla efficiente e la scelta che è stata effettuata.

Nel quinto capitolo verranno mostrati i risultati dei test e i benchmark effettuati con lo scopo di confermare al livello pratico quanto mostrato sotto forma teorica nel capitolo precedente. Saranno descritte in modo dettagliato le condizioni sotto le quali sono stati effettuati i test e limiti della scalabilità della soluzione. Il tutto verrà seguito da grafici esplicativi.

Nel sesto capitolo infine, verranno presentate le conclusioni e possibili strade per lavoro futuro.

Capitolo 2

Stato dell'arte

In questo capitolo verrà presentato lo stato dell'arte per le soluzioni di analisi ed individuazione di minacce ai sistemi informatici. Si procede col fornire un'introduzione preliminare sulle tipologie di attacco affrontate in questa tesi.

2.1 Malware

Il termine malware è una combinazione delle parole *malicious* e *software*. Con questo termine si identificano tutti quei programmi software progettati per danneggiare o effettuare azioni illecite su un sistema informatico. [22]

Gli obiettivi che può avere un malware sono molteplici e sono in continua evoluzione. Il malware, a seconda dello scopo per cui è stato creato e alle sue caratteristiche, viene classificato nei seguenti modi:

Virus Prendono il nome dai virus in campo biologico e si comportano in modo analogo, sono programmi che si replicano sul computer che hanno infettato e si predispongono ad infettare nuovi computer mediante mezzi di trasmissione quali email e chiavette USB. [33]

Spyware Il termine Spyware è una combinazione delle parole *Spy* e *Software*. È un software che viene installato sul computer della vittima a sua insaputa e che raccoglie informazioni. Uno spyware è oggetto di controversia perchè può essere utilizzato negli ambienti lavorativi per controllare le ricerche dei dipendenti o per controllare l'attività dei propri figli su internet. Anche se utilizzato per scopi più innocui

può comunque violare la privacy dell'utente. [37]
Usi più scorretti di programmi spyware prevedono di tracciare la cronologia internet di un utente per inviare pubblicità mirata, accedere alle password degli account in uso sul computer infetto e/o alle informazioni bancarie. Le informazioni raccolte attraverso l'uso di spyware possono essere utilizzate in vari modi, l'uso più frequente e più remunerativo ad oggi è quello di rivendere tali informazioni a dei terzi. [48]

Backdoor Tradotto letteralmente come *porta sul retro*, è un metodo utilizzato per avere un accesso privilegiato e spesso segreto che aggira il sistema di autenticazione previsto. Lo scopo di una backdoor è quello di permettere una connessione in remoto al computer vittima per prenderne il controllo.

Trojan Il cavallo di Troia fu una macchina da guerra che, secondo la leggenda, fu usata dai greci per espugnare la città di Troia. Questo termine è entrato nel linguaggio comune per indicare uno stratagemma con cui penetrare le difese. Nell'ambito dei malware il trojan è un software che si nasconde all'interno di un altro programma all'apparenza innocuo e che, se eseguito, esegue anche il codice del trojan [43]. Oggi col termine trojan ci si riferisce principalmente ai malware ad accesso remoto. Spesso vengono utilizzati per installare backdoor sui sistemi bersaglio. [47]

I malware erano inizialmente usati per compiere azioni dolose sia da hacker malintenzionati che dai governi per sottrarre informazioni personali, inviare spam e commettere frodi. [12] [23]

L'evoluzione e lo sviluppo di internet hanno portato ad un incremento degli utenti connessi sempre maggiore. Questa crescita di internet ha spostato l'obiettivo dei malware che vengono usati sempre di meno per compiere azioni dolose. Fin dal 2003 la maggior parte dei malware sono stati creati per prendere il controllo dei computer dell'utente vittima per scopi illeciti [23]. Solitamente, le macchine infette vengono impiegate per l'invio di email di spam o per effettuare attacchi distribuiti Denial of Service (DDoS).

2.2 Botnet

Nella sua forma più semplice una Botnet è un gruppo di computer che sono stati infettati da un malware che consente ad un attaccante, detto anche *botmaster*, di avere il controllo sulle macchine infettate. Le Botnet sono usate dal botmaster per compiere operazioni illecite ad insaputa della vittima. Una volta infetto, il computer della vittima prende il nome di *zombie* (o, più semplicemente, "bot"). [3]
Per aggiungere un computer ad una botnet si infetta il computer vittima con un malware che installa una backdoor in grado di consentire al botmaster di avere accesso remoto al computer infettato dal malware. Infettando molteplici host, il botmaster avrà così accesso ad un sistema gigantesco di computer zombie pronti ad essere attivati ed eseguire i suoi comandi. Le botnet rilevate e studiate nella letteratura dimostrano come questi sistemi possano arrivare a contenere anche milioni di computer infetti. [3]

I componenti di una botnet sono i seguenti:

Botmaster Il botmaster rappresenta l'utente (o gli utenti) che detiene il il controllo remoto dell'intera botnet. La macchina utilizzata dal botmaster per comunicare con i vari bot viene tipicamente definita come *Command and Control* (C&C). I comandi (e le eventuali risposte) vengono inviati utilizzando protocolli e canali specifici, atti a minimizzare le possibilità di identificazione.

Control protocol Il protocollo utilizzato dal master per comunicare con i computer zombie.

Computer zombie Computer connesso ad internet che è stato infettato attraverso dei virus o trojan e che può essere utilizzato in modo remoto.

Nella figura 2.1 viene rappresentato una possibile architettura di una botnet, in cui il botmaster controlla in modo remoto i computer zombie attraverso dei server C&C.

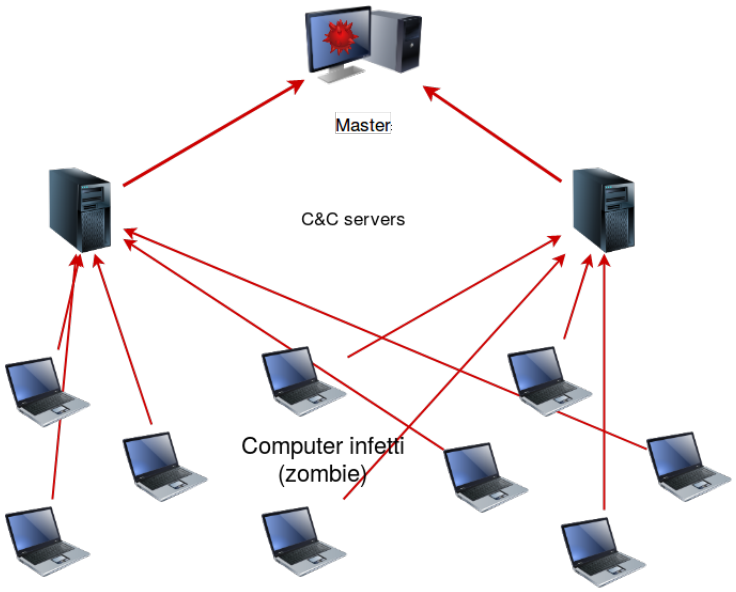

Figura 2.1: Esempio architettura botnet

I principali attacchi informatici effettuati attraverso l'utilizzo delle botnet sono vari: possono essere spam [24], DDoS [18], click fraud [45], phishing [78] e bitcoin mining [52] per citarne alcuni.

Spam È l'invio tramite posta elettronica di messaggi contenenti truffe o pubblicità. Può essere di tipo *orizzontale*, in cui si inviano più messaggi a molti utenti, o *verticale*, in cui si puntano bersagli specifici con email mirate. Lo spam spesso confluisce nel phishing.

Phishing Con l'aumento di transazioni effettuate su internet il phishing è diventato sempre più comune [75]. Il phishing è un tipo di truffa utilizzato per ottenere informazioni da utenti non esperti attraverso l'impersonazione di fonti attendibili [29].

DDoS Acronimo di Distributed Denial of Service, è un tipo di attacco in cui si mira a far esaurire le risorse di un sistema informatico affinchè non riesca più a fornire un servizio in maniera ottimale. Per fare ciò si inviano moltissime richieste al sistema in un breve periodo di tempo utilizzando i computer zombie. L'obiettivo è saturare la risorse disponibili (come la banda o la CPU), causando rallentamenti o il blocco completo del sistema.

Click fraud Questo tipo di frode si verifica sulla pubblicità su internet di tipo pay per click. Questo tipo di pubblicità genera quantità di denaro in base al numero di click effettuati sull'annuncio pubblicitario. Le botnet sfruttano questo tipo di pubblicità utilizzando computer zombie per cliccare in massa gli annunci pubblicitari.

Bitcoin mining I bitcoin [9] vengono creati risolvendo operazioni matematiche complesse. Una botnet può utilizzare i computer zombie come unità di calcolo a cui far eseguire queste computazioni.

2.3 Botnet Detection

Le botnet sono diventate il metodo preferito per lanciare attacchi su internet. Rappresentano una seria minaccia poichè possono inviare attacchi in modo coordinato e istantaneo da numerosi bot [79]. È stimato che ci sono milioni di bot attivi internet ogni giorno, di conseguenza è possibile identificare con le botnet una delle minacce più critiche per le organizzazioni moderne [79].

Le botnet sono difficili da individuare dal momento che l'host autore dell'attacco non è visibile direttamente dalla vittima perchè nascosto da un layer di zombie. Un esempio di attacco è rappresentato in Figura 2.2: il computer vittima non vede un attacco unico, ma un insieme di attacchi da una moltitudine di computer diversi che nascondono il botmaster.

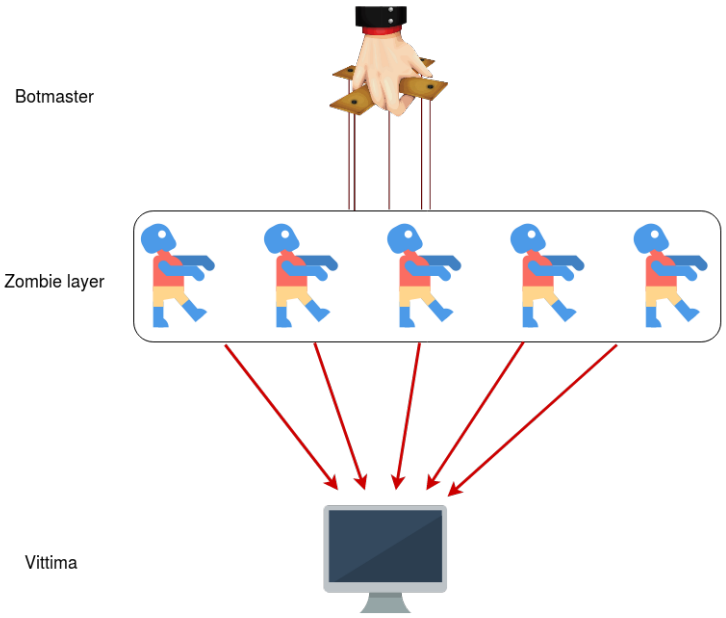

Figura 2.2: Zombie layer

Esistono in letteratura diversi metodi per la rilevazione delle botnet. Ad esempio, gli autori di [73] utilizzano tecniche di clustering dei network flow per rilevare il traffico di rete prodotto dalle botnet, ma ottenendo un consistente quantitativo di falsi positivi. L'approccio descritto in [76] è orientato al rilevamento delle botnet tramite l'utilizzo di signature, di fatto impedendo di identificare le botnet di nuova generazione di cui non si conoscono le caratteristiche. Esistono anche numerosi metodi basati sul data mining [64], che sebbene siano teoricamente in grado di individuare nuove botnet, hanno la tendenza ad essere molto imprecisi. Un altro articolo [67] presenta un elenco di metodi per il rilevamento delle botnet basato sui DNS; tuttavia non tutte le botnet fanno uso del DNS per comunicare con il rispettivo CnC server [63]. Infine, si segnalano gli approcci basati sugli honeypot [74], che possono essere utili in caso di attaccanti poco esperti, ma che sono facilmente aggirabili in caso di avversari più competenti. In questa tesi ci si è focalizzato su tecniche basate sull'analisi dei modelli comportamentali.

2.4 Intrusion Detection System

In questa sezione verranno presentati i meccanismi utilizzati per identificare accessi non autorizzati ai sistemi informatici, che trovano anche impiego come strumenti per la rilevazione di botnet [79].

Un'intrusione si verifica quando un aggressore tenta di entrare o interrompere le normali operazioni di un sistema informativo [72]. Esistono varie classi di intrusione che vanno rilevate: si possono verificare situazioni in cui un utente ruba una password, utenti legittimi che abusano dei loro privilegi o hacker che usano script trovati in rete per attaccare il sistema.

I sistemi di rilevamento delle intrusioni sono gli "allarmi antifurto" del campo della sicurezza informatica. L'obiettivo è difendere un sistema attraverso la segnalazione di allarmi, che vengono emessi ogni volta che viene rilevata un'attività sospetta [62]. Tali sistemi prendono il nome di *Intrusion Detection System* (IDS). Un IDS è un dispositivo software o hardware utilizzato per identificare intrusioni a singoli host o reti locali.

2.4.1 Tipi di IDS

Gli IDS possono essere di due tipi in base al target su cui vengono applicati: possono essere basati su rete o su host.

Host-Based IDS Un IDS basato su host, detto HIDS, risiede su un particolare computer o server e monitora l'attività solo su quel sistema.

Network-Based IDS Un IDS basato sulla rete, detto NIDS, è installato su un computer o dispositivo collegato ad un segmento della rete e ne monitora il traffico alla ricerca di indicazioni di attacchi. Un IDS basato su rete può rilevare molti più attacchi rispetto ad un IDS basato su host, ma richiede una configurazione e manutenzione più complessa.

Un IDS è composto da quattro componenti [72]:

- **Sensori**, sono utilizzati per ricevere informazioni dalla rete o dai computer.
- **Console**, utilizzata per monitorare lo stato della rete e dei computer.
- **Motore**, analizza i dati prelevati dai sensori e provvede a individuare eventuali falle nella sicurezza.
- **Database**, memorizza le regole utilizzate per identificare violazioni di sicurezza.

L'immagine 2.3 descrive i componenti di un NIDS. I pacchetti in entrata e uscita dalla rete sono ricevuti da un sensore che raccoglie il traffico dati, successivamente il motore analizza i dati prelevati dai sensori con le regole presenti nel database.

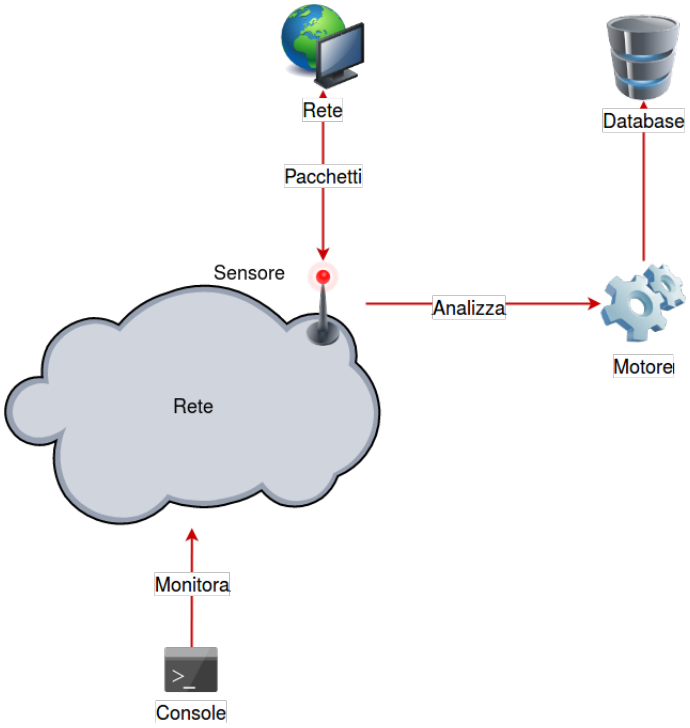

Figura 2.3: Esempio di un IDS

Un'attuale estensione degli IDS sono gli Intrusion Prevention Systems, detti IPS. Tali strumenti, unitamente alla rilevazione di intrusioni, possono impedire l'esecuzione di azioni dannose per il sistema monitorato. Gli IPS usano diverse tecniche di risposte attive, che possono essere suddivise nei seguenti gruppi [69]:

- **terminare** la connessione internet o la sessione utente che sta eseguendo l'attacco
- **bloccare** l'accesso all'obiettivo e agli obiettivi simili all'utente o indirizzo IP che sta tentando un'intrusione
- **cambiare** l'ambiente di sicurezza modificando la configurazione di altri controlli di sicurezza per interrompere l'attacco, come la riconfigurazione delle regole di un firewall. Alcuni IPS possono anche applicare della patch di sicurezza a degli host se l'IPS rileva che l'host presenta delle vulnerabilità.

Poiché i due sistemi coesistono spesso, il termine combinato Intrusion Detection and Prevention System (IDPS) viene generalmente utilizzato per descrivere le attuali tecnologie anti-intrusione [69].

2.4.2 Perchè usare IDS

Dispositivi di tipo IDS sono utili per le difese di sistemi informatici per molteplici ragioni:

- **defense in depth**, l'utilizzo di un dispositivo IDS unito a firewall [19], controlli di accesso e autenticazione [60], e antivirus [65] permette la realizzazione di un meccanismo di protezione multi-livello
- **documentazione**, i dati acquisiti sono utili anche per il miglioramento continuo della qualità. Gli IDS raccolgono costantemente informazioni sugli attacchi che hanno compromesso con successo gli strati esterni dei controlli di sicurezza, per esempio di un firewall. Queste informazioni possono essere utilizzate per identificare e riparare le vulnerabilità esposte dagli attacchi, di fatto aiutando l'organizzazione ad accelerare il processo di risposta agli incidenti e ad apportare miglioramenti continui. Inoltre, sebbene un IDS non sia in grado di assicurare la prevenzione completa delle intrusioni, è comunque in grado di assistere nella revisione fornendo informazioni su come si è verificato l'attacco, assuieme ai dettagli sulle operazioni compiute dall'intruso. Gli IDS possono anche essere utilizzati in ambito forense, risultando utili anche per motivi legali [69]
- **insider threat**, gli attacchi non sempre vengono effettuati da un attaccante esterno alla rete: spesso infatti sono gli utenti interni all'organizzazione che compiono azioni illegittime tentando di compromettere la rete dall'interno. Un IDS è in grado di segnalare anche questo tipo di "intrusioni" [69]

Si segnala anche l'importante funzione di deterrente svolta dalla presenza di un IDS. Se gli utenti esterni e interni sanno che un'organizzazione utilizza un sistema di rilevamento e prevenzione delle intrusioni, saranno meno propensi ad effettuare azioni malevole verso tale azienda [69].

2.4.3 Metodi di rilevamento

Gli IDS utilizzano vari metodi di rilevamento per monitorare e valutare il traffico di rete. I metodi principali sono [62]:

Signature detection Questo metodo di rilevamento delle intrusioni esamina il traffico di rete alla ricerca di pattern che corrispondono a firme note, ovvero pattern di attacco preconfigurati e predeterminati. Un vantaggio di questo approccio è che, avendo un modello con

cui confrontare il comportamento che si sta osservando, il tasso di falsi positivi è molto ridotto. È inoltre veloce e semplice in quanto deve soltanto effettuare confronti con modelli illegali già noti. Uno svantaggio di questo approccio è che si possono verificare molti falsi negativi, cioè i comportamenti dannosi che non vengono segnalati. Ciò si verifica perchè questi sistemi sfruttano esclusivamente le regole per rilevare tipologie di attacco note; pertanto, in presenza di attacchi che presentano caratteristiche differenti dalle signature utilizzate per il confronto, non verrà segnalato alcun allarme. Ne segue che è quindi estremamente importante mantenere aggiornato il database delle signature con tutte le versioni più recenti di possibili comportamenti malevoli [34].

Anomaly detection Questo tipo di rilevamento si basa sull'idea che le azioni più sospette – e quindi di maggiore interesse per un operatore della sicurezza – sono quelle che presentano caratteristiche che deviano dalla normalità; pertanto, tali tecniche si basano sulla rilevazione di anomalie sistema analizzato. E' quindi necessario uno studio anticipato del sistema al fine di definirne il suo comportamento normale. In seguito, verrà poi stabilito quanto sia possibile discostarsi dal tipo di attività normale. Il vantaggio principale di questo approccio consiste nella possibilità di rilevare anche attacchi non noti in precedenza, rendendoli efficaci contro attacchi che riescono ad eludere i sistemi basati su signature, e in alcuni casi anche contro attacchi di tipo zero-day [49]. Gli svantaggi sono la difficoltà nella creazione dei modelli di "normalità"; e la tendenza di queste tecniche a generare un numero considerevole di falsi positivi [42].

2.5 Machine learning

Il machine learning rappresenta uno degli sviluppi più prolifici dell'intelligenza artificiale moderna [66]. L'enfasi del machine learning è sulla capacità del sistema di adattarsi o cambiare, in genere in risposta a qualche forma di esperienza fornita al sistema. Dopo l'apprendimento ci si aspetta che il sistema migliori le prestazioni future sullo stesso compito o lavori simili.

Negli ultimi anni il machine learning ha preso sempre più piede ed è ormai utilizzato quasi ovunque [66]. Alcuni esempi di applicazioni di machine learning nel mondo reale sono il controllo di robot autonomi in grado di navigare da soli, si pensi alle macchine con pilota automatico che stanno entrando in commercio negli

ultimi anni come la Tesla di Elon Musk [16]; il filtro dello spam nelle email [10], il riconoscimento della calligrafia impiegato da software *Optical Character Recognition* [27], il riconoscimento vocale degli assistenti presenti su dispositivi come Google Home [15], Amazon Alexa [20] o Apple HomePod [4]; il rilevamento dei volti nelle fotocamere digitali [21] e così via. Per problemi come il riconoscimento vocale gli algoritmi basati sul machine learning superano tutti gli approcci che sono stati tentati fino ad oggi [70].

Generalmente esistono tre paradigmi di machine learning [77].

Supervised learning Questo paradigma prevede una fase di training in cui viene fornito un apposito dataset definito da una serie di features[1], insieme alle corrispondenti label[2]. L'obiettivo è ottenere un modello in grado di effettuare previsioni su samples non precedentemente noti, analizzandone le features e confrontandole con la conoscenza acquisita in fase di training.

Unsupervised learning Questo paradigma si differenzia dal Supervised Learning in quanto non richiede una fase di training con dataset labellati. Vengono forniti in input dei samples che consistono di sole features, ma non viene fornita alcuna label. L'obiettivo è scoprire nuove proprietà nei dati forniti in input attraverso l'ottimizzazione di un principio di apprendimento.

Reinforced learning Paradigma che si basa sul principio dell'*esplorazione*. Osservando lo stato attuale dell'ambiente e analizzandone le feature, il modello compie un'azione che cambia lo stato dell'ambiente, ricevendo poi una valutazione positiva o negativa sull'azione compiuta. L'obiettivo è compiere una serie di azioni tali da massimizzare la valutazione ricevuta.

2.5.1 Supervised learning

Al computer supervisionato vengono forniti dei dati chiamati *training data*. Questi dati sono dei campioni, ognuno dei quali consiste in una serie di features insieme all'output desiderato.

[1] In ambito di machine learning, si utilizza il termine *feature* per indicare una proprietà misurabile di un determinato sample (dato). Ogni sample fornito in input ad un qualsiasi algoritmo di machine learning sarà quindi identificato da una serie di features.

[2] Le labels vengono utilizzate per indicare cosa rappresenta un determinato sample

In base al valore di output si distinguono due diversi problemi:

- **Classificazione** Se il valore di output è discreto, ad esempio l'appartenenza o la non appartenenza ad una determinata classe.
- **Regressione** Se il valore dell'output è un valore reale continuo in un determinato range.

In entrambi i problemi si vuole trovare una funzione h, che dato un input non conosciuto stima il valore dell'output. L'obiettivo dei problemi di supervised learning è quello di fare una predizione basata su proprietà conosciute apprese durante la fase di training.

La figura 2.4 descrive un problema di supervised learning, dove l'algoritmo di machine learning prende in input un insieme di esempi da cui impara una funzione che gli permette di fare previsioni.

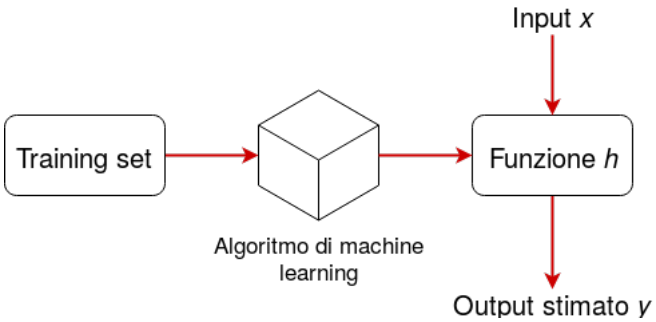

Figura 2.4: Supervised learning

Un esempio di algoritmo supervised sono le reti neurali.

2.5.1.1 Reti neurali

Una rete neurale è un algoritmo che si ispira ai neuroni nel cervello umano. Ogni neurone è responabile della risoluzione di una parte molto piccola del problema, che nel complesso permette al cervello di risolvere problemi difficili. In modo simile una rete neurale è composta da cellule che lavorano insieme per produrre un risultato desiderato. La rete inizia con un input che attraversa i neuroni. Se un particolare neurone riceve abbastanza stimoli invia un messaggio a qualsiasi altro neurone a cui è collegato.

La figura 2.5 rappresenta una semplice neurone che controlla se una persona è viva. Controlla due input: il battito e il respiro. Se il neurone riceve uno stimolo da entrambi restituisce in output che la persone è viva.

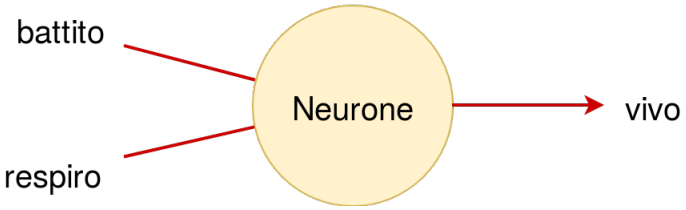

Figura 2.5: Esempio di neurone

Questi algoritmi possono trovare applicazione in IDS [68].

2.5.1.2 Modelli probabilistici

I metodi probabilistici sono fondamentali per il machine learning [66]. La funzione h, per stimare il valore dell'output, sceglie i parametri che massimizzano la probabilità dei dati che lega i valori degli input ai valori degli output.

2.5.1.3 Modelli di Markov

Le serie temporali e, più in generale le sequenze, sono una forma di dati strutturati che rappresenta un elenco di osservazioni per le quali è possibile definire un ordine completo, come ad esempio il tempo in una sequenza temporale.

Sia una sequenza di lunghezza T, $y_n = y_1, \ldots, y_T$. Il termine y_t indica l'osservazione t-esima rispetto all'ordine totale. Siccome gli elementi che compongono la sequenza non sono indipendenti e identicamente distribuiti, la distribuzione congiunta di un modello probabilistico per y è $P(Y_1, \ldots, Y_T)$.

Per osservazioni y_t a valore discreto la distribuzione congiunta cresce esponenzialmente con le dimensioni del dominio di osservazione. Per ridurre la parametrizzazione si usano le *Catene di Markov*. Le Catene di Markov semplificano assumendo che un'osservazione che si verifica in una posizione t della sequenza dipende solo da un numero limitato di predecessori rispetto all'ordine completo. Ciò significa che in una serie temporale un'osservazione al tempo presente tiene conto soltanto di un numero limitato di osservazioni del passato.

Le Catene di Markov stanno alla base dei *Modelli di Markov nascosti* (HMM).

2.5.1.4 Catena di Markov

Una Catena di Markov è un processo stocastico per sequenze. Presuppone che un'osservazione y_t a tempo t dipenda solo da un insieme finito di predecessori $L \geq 1$. Il numero di predecessori L che influenzano la nuova osservazione è detto ordine della Catena di Markov.

Una catena di Markov di ordine L è una sequenza di variabili casuali $\mathbf{Y} = Y_1, \ldots, Y_T$ tali che $\forall t \in \{1, \ldots, T\}$

$$P\left(Y_t = y_t | Y_1, \ldots, Y_{t-1}, Y_{t+1}, \ldots, Y_T\right) = P\left(Y_t = y_t | Y_{t-L}, \ldots, Y_{t-1}\right)$$

2.5.1.5 Modelli di Markov nascosti

Le Catene di Markov modellano i dati sequenziali assumendo che gli elementi della sequenza siano generati da un processo stocastico completamente osservabile. Nella maggior parte dei sistemi reali l'unica informazione disponibile può essere il risultato del processo stocastico ad ogni osservazione y_t mentre lo stato del sistema rimane nascosto.

2.5.2 Unsupervised learning

In questa categoria vengono forniti dei samples con relative feature in input, ma non viene fornita alcuna label. L'obiettivo di questi algoritmi è quello di trovare delle caratteristiche comuni nei dati forniti in input. Se si vogliono identificare dei raggruppamenti di elementi simili, è un problema di Clustering.

Clustering È un tipico esempio di unsupervised learning in cui, dato un insieme di features senza labels, l'obiettivo è quello di individuare dei raggruppamenti, detti cluster, quanto più possibile coerenti. Un esempio di algoritmo di Clustering è il K-Means.

2.5.2.1 K-Means

L'algoritmo K-Means è uno degli algoritmi di clustering più conosciuti e più utilizzati. Permette di suddividere un insieme di oggetti in K gruppi sulla base dei loro attributi. Si assume che gli attributi degli oggetti possano essere rappresentati come vettori, e che quindi formino uno spazio vettoriale. Dal *dataset* fornito vengono inizializzati n punti detti centroidi per ogni cluster che è necessario identificare.

In figura 2.6 vengono inizializzati due centroidi con una croce rossa e blu.

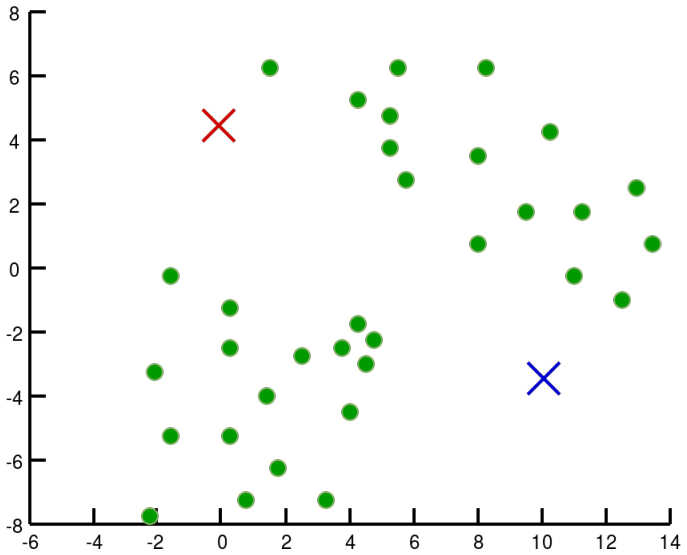

Figura 2.6: Algoritmo k-means - Inizializzazione

L'algoritmo segue una procedura iterativa che ripete due passi, il primo è il passo di assegnazione ai cluster, mentre il secondo è il passo di spostamento dei centroidi. Il primo passo di assegnazione ai cluster controlla la distanza di ogni punto del dataset dai centroidi e assegna ciascun punto al centroide che gli è più vicino. La figura 2.7 rappresenta questo primo passo: i punti più vicini al centroide rosso sono stati colorati di rosso, mentre quelli più vicini al centroide blu sono stati colorati di blu.

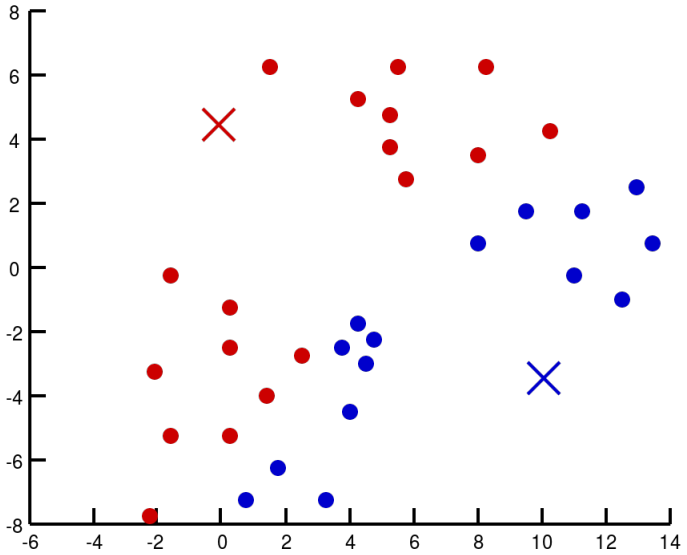

Figura 2.7: Algoritmo k-means - Passo 1

Il secondo passo consiste nel calcolare la media di tutti i punti nel dataset che appartendono ad un cluster e muovere in quella posizione il centroide del cluster. Questo passo viene descritto in figura 2.8

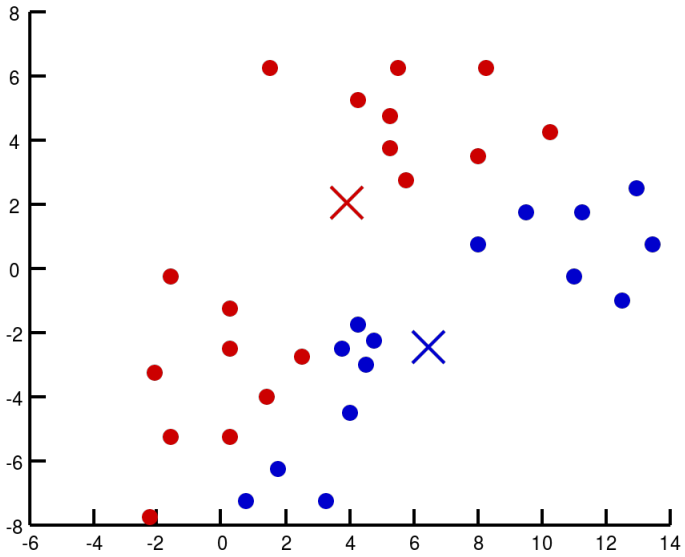

Figura 2.8: Algoritmo k-means - Passo 2

L'algoritmo continua a ripetere questi due passi fino a quando i centroidi non si spostano più, il che significa che l'algoritmo ha raggiunto la convergenza e i due cluster sono stati individuati.

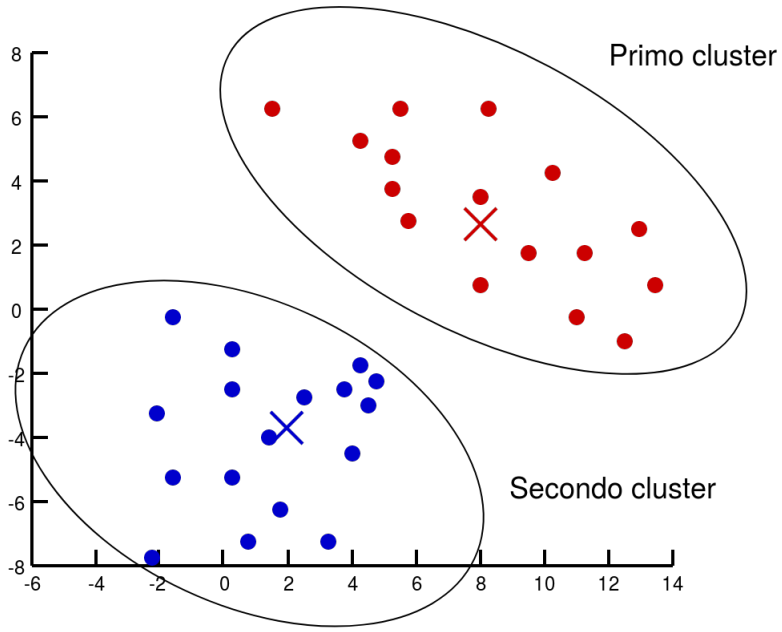

Figura 2.9: Algoritmo k-means - Convergenza

Algoritmi di clustering sono usati per fare botnet detection dagli IDS [73].

2.5.3 Reinforced learning

A differenza dell'unsupervised learning, il reinforced learning viene indirizzato dalla valutazione esterna. Inoltre a differenza del supervised learning in cui viene specificato l'output corrispondente ad un input, nel reinforced learning viene fornita solo una valutazione sull'azione compiuta.

In aggiunta, questo tipo di machine learning è caratterizzato da un processo dinamico in fasi discrete: ad ogni passo, osservando l'ambiente circostante e ottenendo qualche input, il computer compie un'azione e si sposta in un nuovo stato, venendo premiato o punito in base all'azione effettuata [66].

2.5.3.1 Processo decisionale di Markov

Il reinforced learning è strettamente correlato con il processo decisionale di Markov [66], che consiste in una serie di stati $s_0, s_1, \ldots, s_t, s_{t+1}, \ldots$. Allo stato s_t, un'azione

$a_t = \pi(s_t)$ è selezionata da un insieme A di azioni intraprese in funzione dello stato π, che fa muovere l'ambiente per raggiungere lo stato successivo s_{t+1} e la valutazione v_{t+1} associata con la transizione (s_t, a_t, s_{t+1}) viene assegnata. L'obiettivo è ottenere la valutazione più alta possibile, cioè di massimizzare

$$R = \sum_{t=0}^{N-1} r_{t+1}$$

con N istante di tempo in cui lo stato finale è raggiunto.

Capitolo 3

Analisi

3.1 Analisi traffico di rete

Nell'era digitale il traffico di rete è aumentato notevolmente e con esso gli attacchi di tipo informatico, per questo motivo è necessario trovare soluzioni per analizzare questo grande quantitativo di pacchetti che attraversano i dispositivi di rete delle aziende di grosse dimensioni. Nel mondo informatico di oggi è molto importante poter determinare rapidamente e con precisione l'origine e la portata di un potenziale attacco su una rete al fine di poterlo contrastare in modo efficace. Per fare ciò viene costruito un *audit trail*[1] di informazioni collezionate dal traffico di rete usando una combinazione di network flow e packet capture.

3.1.1 Packet Capture

Per packet capture (PCAP) si intende la cattura di traffico internet che attraversa un dispositivo di rete; nel caso la cattura avvenga su un border router, è così possibile ispezionare tutto il traffico di rete generato da un'organizzazione. I software che si occupano di effettuare la cattura del traffico operano intercettando i singoli pacchetti, offrendo anche la possibilità di archiviarli su memoria persistente. Esempi di questi programmi sono la libreria **libpcap** per i sistemi operativi Unix, mentre sui sistemi Windows si fa utilizzo di **WinPcap** [50].

[1] Un audit trail è un file che contiene una registrazione cronologica di attività relative alla sicurezza per consentire la ricostruzione e l'esame di eventi [7].

3.1.2 Network Flow

Un network flow è una sequenza di pacchetti inviati da una sorgente ad una destinazione che presentano degli attributi in comune [71], quali:

- indirizzo IP sorgente
- indirizzo IP destinazione
- porta sorgente
- porta destinazione
- protocollo

Tutti i pacchetti che condividono gli stessi valori di questi attributi possono essere raggruppati in un unico flow. Risulta quindi evidente come le analisi basate sui network flow presentino numerosi vantaggi rispetto a quelle basate sui PCAP. Ad esempio, poichè i flow rappresentano un'aggregazione di pacchetti, è possibile diminuire drasticamente lo spazio richiesto per la memorizzazione. Di conseguenza, anche l'ispezione di tali dati risulterà più rapida e richiederà meno risorse computazionali. Va però sottolineato che i network flow non consentono di salvare il payload dei singoli pacchetti: questa caratteristica li rende da un lato meno dettagliati rispetto ai full packet capture; dall'altro, consente di effettuare l'analisi dei dati senza doversi preoccupare di questioni legate alla privacy degli utenti monitorati.

3.1.2.1 NetFlow

NetFlow [1] è un protocollo di analisi di rete introdotto da Cisco [25] che offre la possibilità di raccogliere informazioni dettagliate sul traffico che attraversa un'interfaccia. I dispositivi di rete che supportano NetFlow possono raccogliere statistiche sul traffico ed esportarle come record verso un NetFlow collector, un server che esegue l'analisi del traffico.

Cisco definisce un flow come una sequenza unidirezionale di pacchetti che condividono tutti i seguenti 7 valori [1].

- interfaccia di ingresso
- indirizzo IP sorgente
- indirizzo IP destinazione

- protocollo IP
- porta sorgente TCP o UDP, 0 per altri protocolli
- IP Type of Service

La figura 3.1 descrive un esempio di architettura basata su NetFlow.

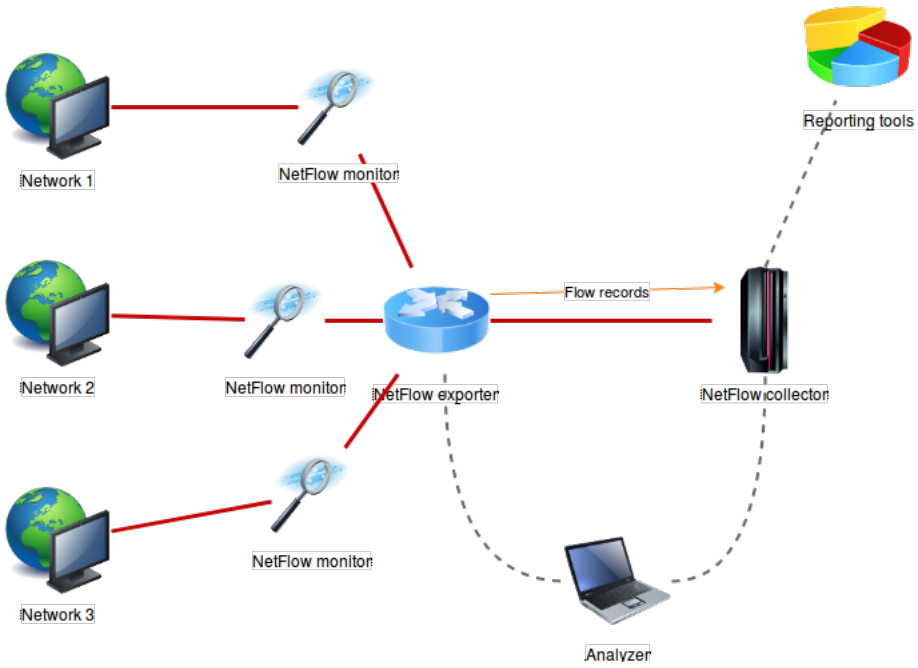

Figura 3.1: Architettura NetFlow

NetFlow monitor Un componente applicato a un'interfaccia che raccoglie informazioni sui flow. I NetFlow monitor sono costituiti da un record e una cache.

NetFlow exporter Aggrega i pacchetti in flows e ne esporta i record verso uno o più *flow collectors*. Quando dei pacchetti arrivano al NetFlow exporter, vengono ispezionati singolarmente per uno o più attributi che vengono utilizzati per determinare se il pachetto è univoco o è simile agli altri pacchetti. Se il pacchetto presenta attributi simili viene classificato nello stesso flow. Dopo aver esaminato questi attributi, il NetFlow exporter li aggrega in record di flow e li salva in un database che può essere una cache NetFlow o un NetFlow collector.

La figura 3.2 rappresenta il funzionamento di un NetFlow exporter.

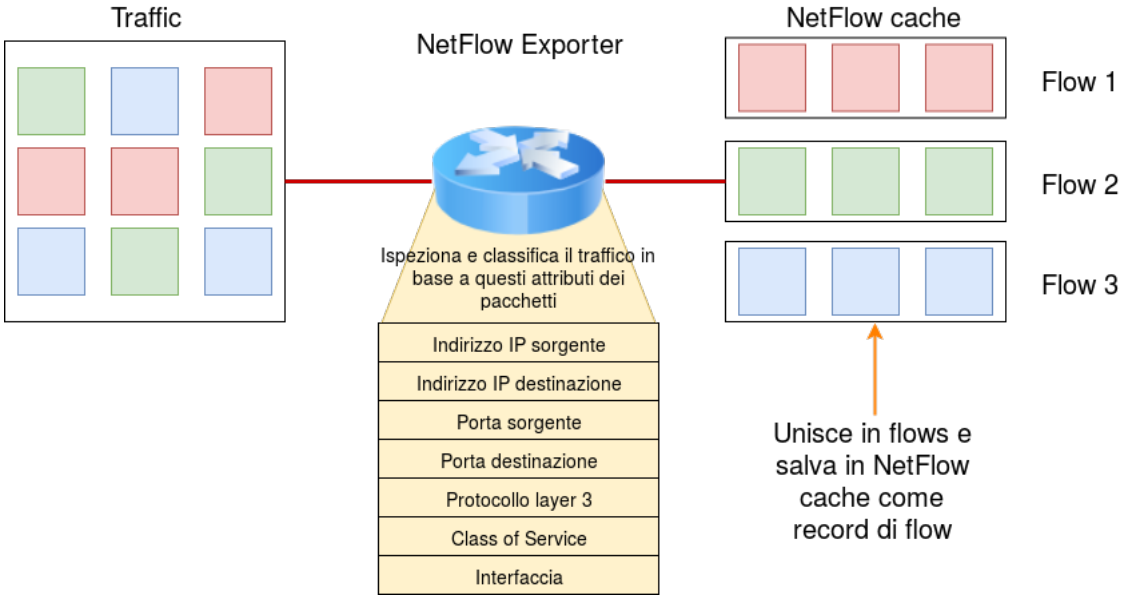

Figura 3.2: NetFlow exporter

NetFlow collector Responsabile della ricezione, conservazione e pre-elaborazione dei dati di un flow ricevuti da un *flow exporter*. Solitamente è un software separato in esecuzione su un server di rete. I record NetFlow vengono esportati in un NetFlow collector tramite protocollo UDP.

Analysis application Analizza i dati dei flows ricevuti nel contesto del rilevamento delle intrusioni o del profilo di traffico. Un esempio di Analysis application possono essere degli IDPS. L'immagine 3.3 rappresenta SolarWinds NetFlow Traffic Analyzer, un software per l'analisi dei flows [35].

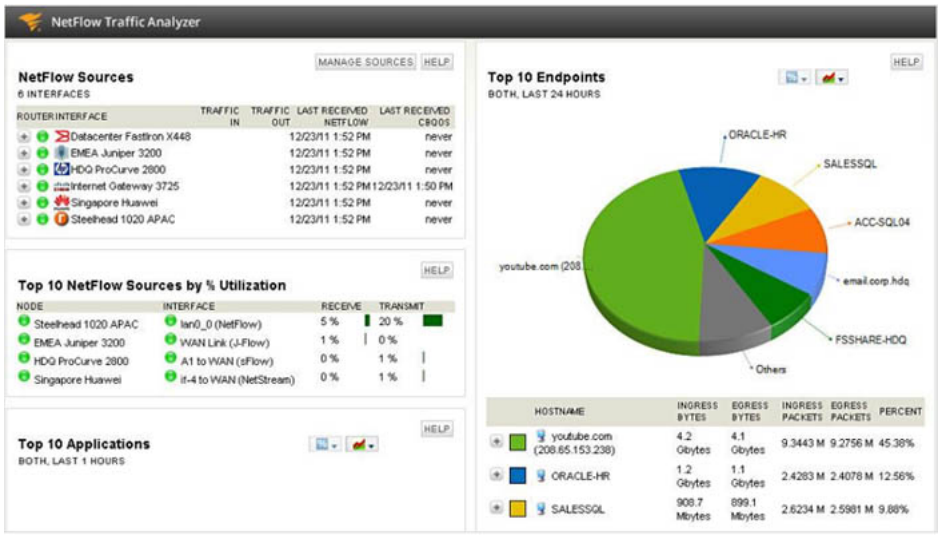

Figura 3.3: SolarWinds NetFlow Traffic Analyzer

3.2 Strumenti di monitoraggio e analisi del traffico di rete

In questa sezione verrano presentati i software per la cattura dei network flows che saranno oggetto di analisi di questa tesi: Argus e nProbe. Questi software sono Open Source e sono stati usati su una distribuzione GNU/Linux.

3.2.1 Audit Record Generation and Utilization System

Argus (Audit Record Generation and Utilization System) è stato la prima implementazione del monitoraggio dei flow, è un progetto open source e multipiattaforma [5]. Quest'ultima particolarità lo rende molto interessante poichè supportando molti sistemi operativi, tra i quali Windows, MacOSX, Linux, Solaris, FreeBSD, OpenBSD, IRIX e OpenWrt, può essere adoperato in quasi tutte le reti comprendendo la maggior parte degli host. La sua architettura è di tipo server/client. Il server recupera i pacchetti ricevuti da una o più interfacce di rete disponibili su una una macchina e Argus assembla poi questi pacchetti in dati binari che rappresentano dei flow. Lo scopo dei client è di quello di leggere i dati dei flow.

Argus viene utilizzato da molte università e aziende [6] per registrare dei flow che vengono utilizzati sia nell'analisi immediata dell'utilizzo della rete, sia nell'analisi storica.

I file di Argus sono compressi con algoritmi che offrono un rapporto di compressione fino 10.000:1. La compressione di questi file è molto importante perchè tendono ad essere molto grandi e possono raggiungere dimensioni difficili da gestire.

La tabella 3.1 descrive i campi dei flow che argus salva su file. La prima colonna rappresenta il nome del campo del pacchetto di rete, la seconda colonna è una descrizione del campo.

campo	**descrizione**
StartTime	record start time
Dur	record total duration
Proto	transaction protocol
SrcAddr	source IP address
Sport	source port number
Dir	direction of transaction
DstAddr	destination IP address
Dport	destination port number
State	transaction state
sTos	source TOS byte value
dTos	destination TOS byte value
TotPkts	total transaction packet count
TotBytes	total transaction bytes
SrcBytes	src ->dst transaction bytes
srcUdata	source user data buffer
dstUdata	destination user data buffer
Label	metadata label

Tabella 3.1: Campi di Argus

3.2.2 nProbe

Negli ambienti commerciali, NetFlow è probabilmente lo standard de facto per la gestione del traffico di rete. nProbe è un software in grado di reccogliere, analizzare ed esportare report sul traffico di rete utilizzando il formato standard Cisco NetFlow [26]. È disponibile per la maggior parte dei sistemi operativi sul mercato.

La tabella 3.2 descrive i principali campi dei flow che nProbe è in grado di collezionare: nella colonna di sinistra si riporta il nome del campo, mentre viene fornita una breve descrizione corrispondente nella colonna di destra.

campo	descrizione
IPV4_SRC_ADDR	IPv4 source address
IPV4_DST_ADDR	IPv4 destination address
IPV4_NEXT_HOP	IPv4 next hop address
INPUT_SNMP	input interface SNMP idx
OUTPUT_SNMP	output interface SNMP idx
IN_PKTS	incoming flow packets (src ->dst)
IN_BYTES	incoming flow bytes (src ->dst)
FIRST_SWITCHED	SysUptime (msec) of the first flow pkt
LAST_SWITCHED	SysUptime (msec) of the last flow pkt
L4_SRC_PORT	IPv4 source port
L4_DST_PORT	IPv4 destination port
TCP_FLAGS	cumulative of all flow TCP flags
PROTOCOL	IP protocol byte
SRC_TOS	Type of service byte
SRC_AS	source BGP AS
DST_AS	destination BGP AS
IPV4_SRC_MASK	IPv4 source subnet mask
IPV4_DST_MASK	IPv4 dest subnet mask
L7_PROTO	layer 7 protocol (numeric)
BIFLOW_DIRECTION	1=initiator, 2=reverseInitiator
FLOW_START_SEC	seconds (epoch) of the first flow packet
FLOW_END_SEC	seconds (epoch) of the last flow packet
OUT_PKTS	outgoing flow packets (dst ->src)
OUT_BYTES	outgoing flow bytes (dst ->src)
FLOW_ID	serial flow identifier
FLOW_ACTIVE_TIMEOUT	activity timeout of flow cache entries
FLOW_INACTIVE_TIMEOUT	inactivity timeout of flow cache entries
IN_SRC_MAC	source MAC address
OUT_DST_MAC	destination MAC address

Tabella 3.2: Campi di nProbe

Rispetto ad Argus, nProbe è un software scalabile ed è stato progettato per restare al passo con le velocità multi-Gbit su hardware comune. Usando una CPU dual-core nProbe può essere usato per catturare pacchetti a 1 Gbit con perdita di pacchetti inferiore al 1%. Inoltre nProbe permette una configurazione più dettagliata e personalizzabile, permette di scegliere molti campi per l'analisi dei network flow [26].

Di contro, Argus è compatibile con un maggiore numero di sistemi: Mentre nProbe è disponibile solo per Unix, Windows o MacOs X, Argus funziona anche su FreeBSD, OpenBSD, NetBSD, AIX, HP-UX, VxWorks, IRIS e OpenWrt. inoltre è stato portato su molte piattaforme hardware accelerate come Bivio, Pluribus, Arista, Tilera [5].

3.3 Stratosphere Suite

In questa tesi si è analizzata la Stratosphere Suite [38], un insieme di software open source disponibile sia per sistemi operativi Windows che per le distribuzioni GNU/-Linux. La Stratosphere Suite si compone di due parti distinte: Stratosphere IPS e Stratosphere Testing Framework. Stratosphere IPS è un sistema di rilevazione e prevenzione delle intrusioni orientato all'analisi comportamentale basata sui network flow. L'obiettivo di Stratosphere IPS è quello di creare un Intrusion Prevention System in grado di rilevare e bloccare comportamenti malevoli all'interno di una rete. Si sottolinea che, per questa tesi, è stata utilizzata la versione di Stratosphere IPS per sistemi Linux (slips). Stratosphere Testing Framework (stf) si occupa della generazione dei modelli comportamentali che saranno poi usati da Stratosphere IPS per effettuare le analisi del traffico e segnalare le possibili intrusioni.

3.3.1 Stratosphere Linux IPS

Slips si occupa della parte di rilevazione delle attività malevole impiegando algoritmi di machine learning che analizzano network flow. Questi algoritmi utilizzano i network flow in ingresso per elaborare dei modelli comportamentali in real-time di ciascun host monitorato: tali modelli comportamentali saranno quindi confrontati con dei modelli predefiniti che includono attività malevole. Nel caso slips riveli somiglianze significative tra i modelli "attuali" e quelli utilizzati per il confronto, genererà un allarme. Il processo di cattura dei network flow è effettuato attraverso Argus. L'idea di base è quella di leggere i flow da Argus e di inviarli a Slips. I modelli comportamentali da fornire a slips per effettuare il confronto vengono creati da Stratosphere Testing Framework.

La figura 3.4 descrive l'architettura della suite di Stratosphere. Su una rete interna o host singolo è presente un'istanza di Argus che cattura il traffico e crea dei network flow. La suite di Stratosphere protegge la rete interna tramite Slips che analizza il traffico catturato da Argus e stf che crea dei modelli comportamentali dai flows di Argus.

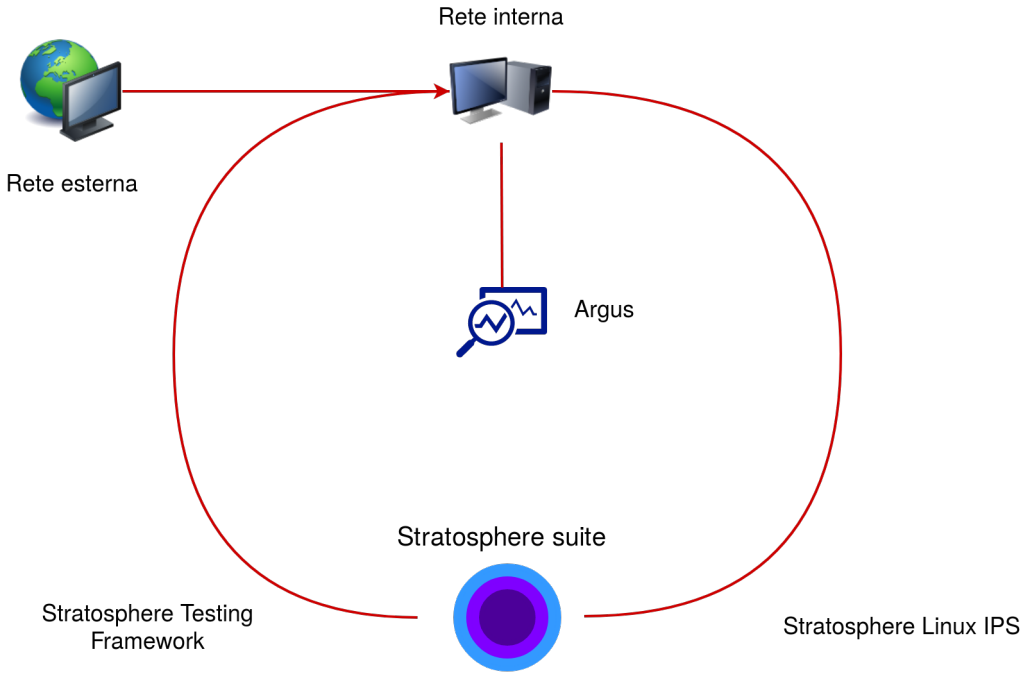

Figura 3.4: Architettura Stratosphere

3.3.2 Stratosphere Testing Framework

Stf è un framework di ricerca sulla sicurezza della rete per analizzare i modelli comportamentali delle connessioni di rete. Il suo obiettivo è aiutare i ricercatori a trovare nuovi comportamenti malware, etichettare tali comportamenti, creare i loro modelli di traffico e verificare gli algoritmi di rilevamento. Una volta creati e verificati i migliori modelli comportamentali di malware, questi verranno utilizzati da slips per il rilevamento. Stratosphere Testing Framework usa algoritmi di machine learning sui modelli comportamentali.

Il nucleo di *Stratosphere IPS* è composto dai modelli comportamentali di reti e algoritmi di rilevamento. I modelli comportamentali rappresentano ciò che una connessione specifica fa nella rete durante la sua vita. Per stf una connessione è un singolo flow, ovvero una sequenza di pacchetti che condividono dei valori. Il comportamento è costuito analizzando la sua periodicità, le dimensioni e la durata di ciascun flusso. Sulla base di queste caratteristiche a ciascun flusso viene assegnata una lettera e il gruppo di lettere caratterizza il comportamento della connessione.

Prendiamo come esempio una connessione generata da una botnet che ha il seguente modello comportamentale

```
1 88*y*y*i*H*H*H*y*0yy*H*H*H*y*y*y*y*y*H*h*y*h*h*H*H*h*H*y*y*y*y*H*
```

In questo caso ci dice che i flussi sono altamente periodici (lettere h, i), con qualche periodicità persa vicino all'inizio (lettere y). I flussi hanno anche una grande dimensione con una durata media. I simboli tra le lettere sono correlati al tempo trascorso tra i flussi. In questo caso il simbolo '*' significa che il flusso è separato da meno di un'ora. Con l'utilizzo di questo tipo di modelli siamo in grado di generare le caratteristiche comportamentali di un gran numero di azioni dannose.

L'immagine 3.5 mostra i criteri di assegnazione delle lettere per i modelli comportamentali.

	Size Small			Size Medium			Size Large		
	Dur. Short	Dur. Med.	Dur. Long	Dur. Short	Dur. Med.	Dur. Long	Dur. Short	Dur. Med.	Dur. Long
Strong Periodicity	a	b	c	d	e	f	g	h	i
Weak Periodicity	A	B	C	D	E	F	G	H	I
Weak Non-Periodicity	r	s	t	u	v	w	x	y	z
Strong Non-Periodicity	R	S	T	U	V	W	X	Y	Z
No Data	1	2	3	4	5	6	7	8	9

Symbols for time difference:

Between 0 and 5 seconds:	.
Between 5 and 60 seconds:	,
Between 60 secs and 5 mins:	+
Between 5 mins and 1 hour:	*
Timeout of 1 hour	0

Figura 3.5: Tabella modelli comportamentali

3.4 Analisi del problema

Questa tesi si occupa di risolvere un duplice problema: come installare ed utilizzare correttamente la suite di Stratosphere; come poter estendere la suite di Stratosphere utilizzando formati di netflow differenti da quelli prodotti da Argus.

3.4.1 Deployment di Stratosphere IPS

La suite di Stratosphere è attualmente pubblicata in alpha [38]. Un programma in alpha è un programma che è ancora nelle prime fasi di test e in genere include bug significativi e problemi di usabilità, pertanto se ne sconsiglia l'utilizzo in fasi di produzione [2].

Per l'installazione e l'utilizzo di questo software è stato necessario scaricare il codice sorgente da GitHub. Quando si esegue il programma scritto in Python *slips.py* con il comando 4.2

```
cat file.binetflow | ./slips.py -f models -d
```

Listing 3.1: comando per eseguire slips

L'output generato è vuoto e il programma non termina l'esecuzione. Si è provato con diversi file di netflow con gli stessi risultati, il programma sembra andare in loop senza mostrare alcun tipo di output. Si è quindi ispezionato il codice con tecniche di debug alla ricerca di una possibile soluzione per il corretto deployment del software.

3.4.2 Problematiche dovute all'utilizzo di due diversi formati

I file prodotti da nProbe e Argus, oltre ad avere formati diversi, hanno anche campi eterogenei. Stratosphere IPS si appoggia ad Argus per la cattura di file e di conseguenza può leggere file solo in formato binetflow. Se una azienda si appoggia a Cisco utilizzando nProbe non può quindi fare utilizzo di questo software, poichè non riesce a leggere file in formato differente. I file prodotti da nProbe hanno una struttura gerarchica fissa ben definita: ci sono 4 livelli di subdir, in cui il primo livello indica l'anno, il secondo il mese, il terzo il giorno e il quarto l'ora. All'interno dell'ultima subdir, quella delle ore, ci sono 60 file uno per ogni minuto della giornata. La immagine 3.6 descrive la struttura gerarchica dei file.

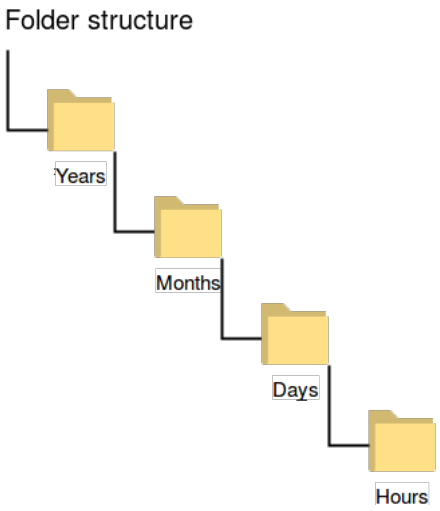

Figura 3.6: Folder Structure

Questa incompatibilità ha portato alla necessità di una conversione: i file prodotti da nProbe devono essere convertiti in file con formato usato da Argus. Questa

conversione deve essere precisa (i.e., evitare al più possibile la perdita di informazioni) ed efficiente (i.e., essere eseguita in breve tempo in modo tale da poter utilizzare i flow convertiti per analisi in quasi real-time).

Capitolo 4

Soluzione proposta

In questo capitolo viene presentato il programma di conversione. La prima parte descrive le scelte effettuate per convertire i campi nell'altro formato e come automatizzare la procedura. La seconda parte introduce il concetto di programmazione parallela e le possibili soluzioni per rendere efficiente la conversione.

4.1 Conversione

Per convertire i file di nProbe nel formato utilizzato da Argus si è fatto per prima cosa uno studio sui campi degli header per individuarne le differenze. Alcuni campi sono presenti in entrambi i formati seppure con nome diverso, altri sono stati ottenuti come combinazione, mentre quelli che nProbe utilizza in più rispetto ad Argus sono stati scartati.

Il primo campo di Argus è il campo *StartTime*. Il corrispondente campo usato da nProbe è *FIRST_SWITCHED*. In figura 4.1 vengono descritte le formattazione dei 2 campi.

Argus	nProbe
StartTime	**FIRST_SWITCHED**
1970/01/01 01:00:00.000000	1496354368.886618
1970/01/01 01:00:12.442467	1496354369.137147
1970/01/01 01:01:11.901878	1496354368.905439
1970/01/01 01:01:12:938923	1496354368.905350
1970/01/01 01:02:20:303943	1496354369.996835

Figura 4.1: Formattazione delle date

Argus salva la data seguita da uno spazio bianco, le ore i minuti e i secondi e i microsecondi. nProbe invece non salva la data ma soltanto i secondi nel formato 10^{10} con i microsecondi. Per effettuare una conversione precisa dei 2 campi la data nel formato YYYY/MM/DD è stata presa dalla struttura gerarchica delle cartelle guardando in modo ricorsivo la posizione del file all'interno del file system. Per i secondi si è convertito i 10 gigasecondi in secondi, mentre per i microsecondi non c'è stato bisogno di conversione.

Il secondo campo di Argus è *Duration*. Non esistendo un campo di nProbe che indica la durata si è usata la differenza tra i campi *LAST_SWITCHED* e *FIRST_SWITCHED*. In figura 4.2 viene mostrata l'operazione di differenza.

Argus	nProbe		
Duration	**LAST_SWITCHED**		**FIRST_SWITCHED**
1.249548	1496354370.136166		1496354368.886618
0.999100	1496354370.136213	—	1496354369.137147
1.246654	1496354370.152093		1496354368.905439
1.387396	1496354370.292746		1496354368.905350
0.420741	1496354370.417576		1496354369.996835

Figura 4.2: Durata

Dopo aver fatto la differenza tra i 2 campi si converte il risultato in secondi e microsecondi per rispettare il formato di Argus.

Il terzo campo di Argus è *Proto* che indica il protocollo usato dalla connessione. Argus salva in questo campo la keyword del protocollo mentre nProbe salva il numero decimale. La figura 4.3 mostra la differenza tra i 2 formati.

Argus	nProbe
Proto	**PROTOCOL**
l2tp	115
ipcv	71
ipv6	41
udp	17
icmp	1

Figura 4.3: Protocolli

Per effettuare la conversione tra numeri e keyword si è usato un *dictionary*, un costrutto del linguaggio Python. Questa particolare struttura dati è un array associativo in cui una chiave viene associata ad un valore. Come chiave si è usato il numero decimale del protocollo e come valore la keyword. Il costo medio di accesso ad un valore con questa struttura dati è $\mathcal{O}(1)$. La figura 4.1 rappresenta la sintassi di un dictionary in Python.

```
dict_protocols = {
    "0":"hopopt",
    "1":"icmp",
    "2":"igmp",
    "3":"ggp",
    "4":"ipv4",
    "5":"st",
    "6":"tcp",
    "7":"cbt",
    "8":"egp",
    "9":"igp",
    "10":"bbn-rcc-mon",
```

Listing 4.1: Esempio di dictionary

Il quarto campo di Argus è *SrcAddr* che indica l'indirizzo IP sorgente. Questo campo è presente nei campi di nProbe, di conseguenza non è necessario effettuare una conversione.

Il quinto campo di Argus è *Sport* che indica la porta di origine della connessione. Il campo corrispettivo di nProbe è *L4_SRC_PORT* e condividono entrambi lo stesso formato, non è pertanto necessario alcun tipo di conversione per questi 2 campi.

Il sesto campo di Argus si chiama *Dir* e rappresenta la direzione della connessione. Per la conversione si è utilizzato il campo di nProbe *BIFLOW_DIRECTION*. In nProbe si ha come valore del campo BIFLOW_DIRECTION solo 1 e 2. Il nu-

mero 1 indica un flow monodirezionale che va dall'indirizzo sorgente al indirizzo destinazione, mentre il valore 2 indica un flow bidirezionale. In Argus si hanno più valori:

- **-** la connessione è normale
- **|** la connessione è stata resettata
- **o** la connessione è scaduta
- **?** la direzione della connessione è sconosciuta

La figura 4.4 mostra come è stata effettuata la conversione.

Argus	nProbe
Dir	**BIFLOW_DIRECTION**
<-	1
<->	2
<?>	1
->	1
who	1

Figura 4.4: Direzione connessione

Nel primo caso si può convertire con 1 ma bisogna invertire i campi src e dst perchè la connessione viene inizializzata a sinistra. Nel secondo caso si converte con 2 perchè il flow è bidirezionale. Nel terzo caso la direzione della connessione è sconosciuta, si è scelto di convertire con 1 per sicurezza e non lasciare il campo vuoto. Nel quarto caso il flow è monodirezionale e quindi si converte con 1. Nell'ultimo caso *who* è una keyword che indica una comunicazione *icmp* che non viene utilizzata da Stratosphere, si è scelto di convertire con 1 per sicurezza.

Il settimo campo di Argus è *DstAddr* e rappresenta l'indirizzo IP di destinazione. Il campo corrispondente di nProbe si chiama *IPV4_DST_ADDR*. Come nel caso dell'indirizzo IP sorgente anche qui non è necessaria la conversione.

L'ottavo campo di Argus è *Dport* e indica la porta di destinazione. Argus usa una formattazione decimale, stessa cosa il campo *DST_PORT* di nProbe.

Il nono campo di Argus è *State* e denota lo stato della connessione. Questo campo non è generabile a partire dai file di nProbe. Per effettuare questa conversione si

è prima verificato l'impatto di questo campo nella generazione dei modelli di STF inserendo un valore non corretto. I modelli generati cambiando i valori di questo campo con valori non corretti restano invariati, pertanto il campo State non ne influenza la generazione. Per la conversione si è quindi inserito un valore costante "CON" per sicurezza. La figura 4.5 mostra esempi di valori del campo State di Argus.

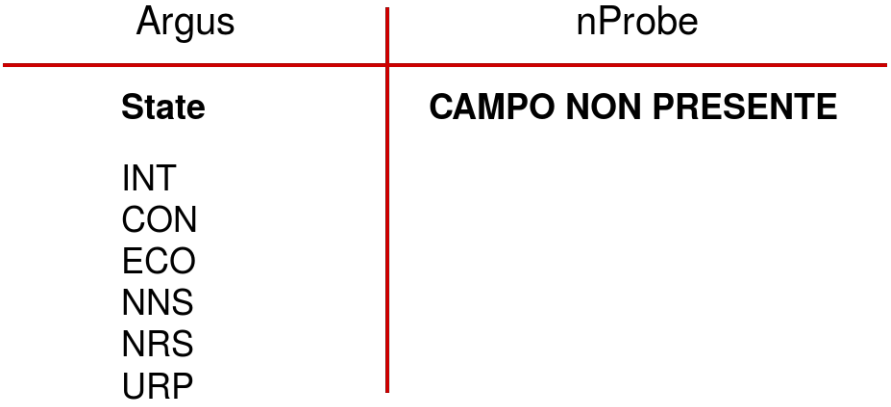

Figura 4.5: Stato connessione

Il decimo e undicesimo campo di Argus sono *sTos dTos* e rappresentano il *Type of Service* sorgente e destinazione di un pacchetto. Questi campi sono presenti in nProbe e condividono la stessa formattazione di Argus, non è pertanto necessaria una conversione.

Il dodicesimo campo di Argus è *TotPkts* e rappresenta i pacchetti totali transitati nella connessione. Questo campo non compare in nProbe che ha però i campi *IN_PKTS* e *OUT_PKTS*. Per fare la conversione da nProbe ad Argus si sono sommati questi 2 valori per ottenere il totale dei pacchetti. La figura 4.6 mostra l'operazione di somma e la formattazione dei dati.

Argus	nProbe	
TotPkts	**IN_PKTS**	**OUT_PKTS**
466	423	43
57	54	3
477	432	45
9	3	6
7	4	3

Figura 4.6: Pacchetti totali transitati nella connessione

Il tredicesimo campo di Argus è *TotBytes* e rappresenta i bytes totali transitati nella connessione. Come visto per il campo TotPkts, in nProbe questo campo non campare direttamente ma sono presenti *IN_BYTES* e *OUT_BYTES*. Per fare la conversione questi 2 campi vengono sommati per ottenere il totale. La figura 4.7 mostra l'operazione di somma e la formattazione dei dati.

Argus	nProbe	
TotBytes	**IN_BYTES**	**OUT_BYTES**
141	44	97
412	67	345
558	2	556
49	45	4
76	56	6

Figura 4.7: Bytes totali transitati nella connessione

Il quattordicesimo campo di Argus è *SrcBytes* che indica i bytes in entrata dal sorgente. Per fare questa conversione si è usato il campo di nProbe *IN_BYTES* in figura 4.7.

Il quindicesimo e ultimo campo di Argus è il campo label che serve per i metadata. Questo campo non è presente in nProbe e non è necessario per la generazione dei modelli, viene quindi lasciato vuoto.

4.2 Automatizzazione conversione

La conversione è stata automatizzata con la scrittura di uno script in Python. Si è scelto di scrivere un programma usando questo linguaggio per la facilità di utilizzo nel lavorare con i file e per le performance.

4.2.1 Lettura dei file

Il problema richiede lo sviluppo di un programma che converte file in modalità batch. La figura 4.8 rappresenta l'operazione da eseguire per la lettura dei file. I file, che sono organizzati in cartelle per data, sono presi in modo ricorsivo e inseriti in una coda. Al termine della lettura dei file si ha una coda con l'insieme dei path dei file da leggere.

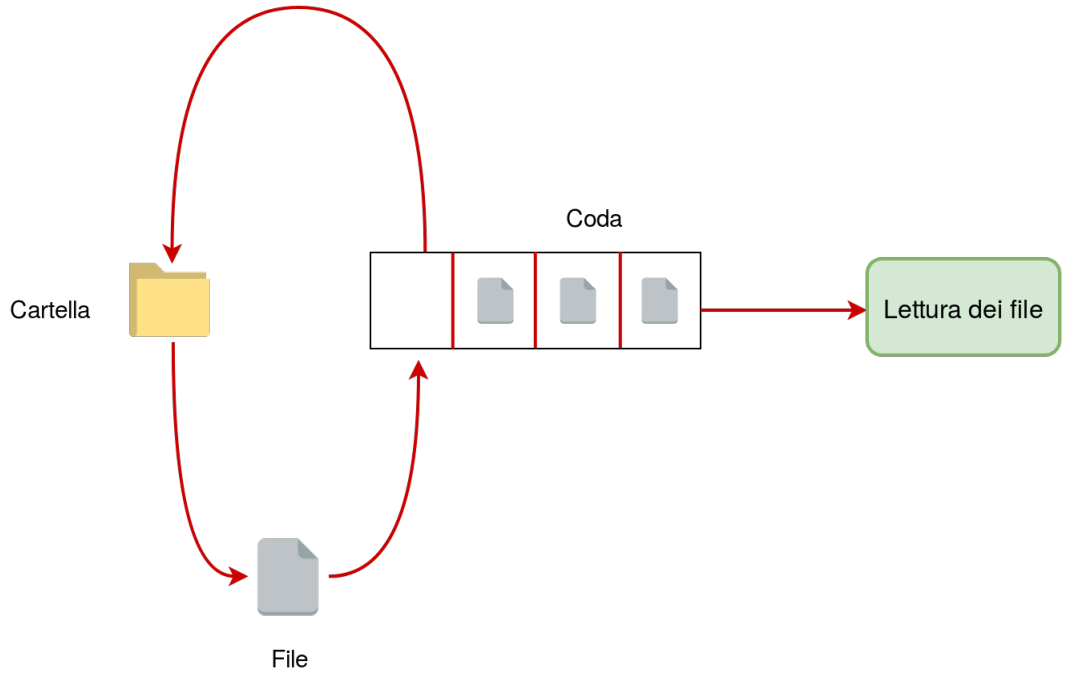

Figura 4.8: Lettura dei file ricorsiva

Lo pseudocodice 1 descrive l'immagine 4.8. Alla riga 1 si assegna ad una variabile il path della cartella in cui sono contenuti i file e nella seconda riga si crea un oggetto coda [32]. Si creano poi 2 cicli *for* annidati che scorrono la struttura gerarchica della cartella e per ogni ogni file all'interno della cartella *minuti* si mette in coda il file. Quando l'esecuzione dei 2 cicli *for* finisce rimane una coda con all'interno tutti i path dei file.

Algorithm 1 Inserimento file in una coda

Input: la variabile pathCartella è il percorso della cartella che fornisce l'utente

```
1: cartellaRoot ←pathCartella
2: Coda ←Queue()                                  ▷ inizializza una coda
3: for mesi, giorni, minuti in cartellaRoot do
4:     for file in minuti do
5:         Coda.append() ←file                    ▷ inserisce il file nella coda
```

4.2.2 Lettura righe dei file

Una volta che la coda è completa si passa alla lettura dei file. Per questa operazione si prende un file file dalla coda e si legge una riga alla volta. Da ogni riga vengono estratti i campi utili alla conversione. La figura 4.9 descrive questa operazione. Si è scelto di leggere i file per righe e non per intero perchè sono troppo grandi e rischiano di saturare la RAM, soprattutto leggendo più file in contemporanea. La soluzione scelta permette di eseguire l'algoritmo anche su computer con poca RAM.

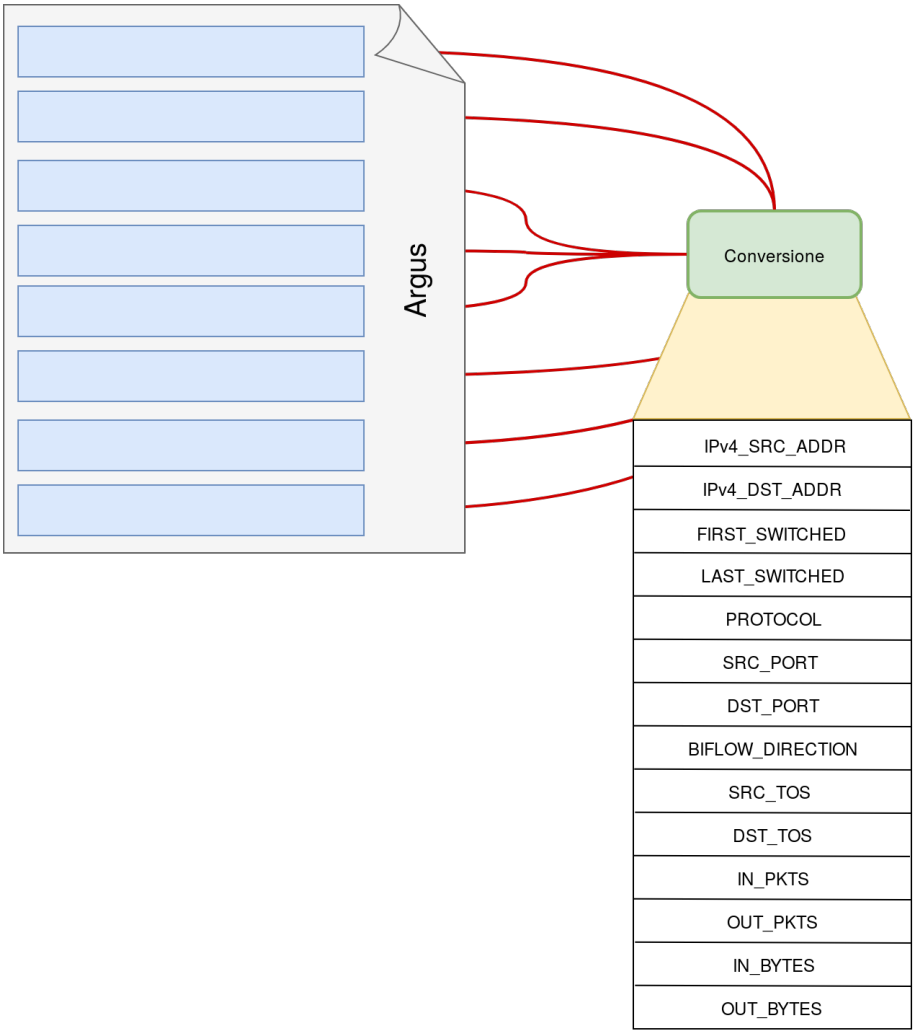

Figura 4.9: Lettura delle righe

Lo pseudocodice 2 descrive l'immagine 4.9. Si inizializza una lista per ogni campo di nProbe da leggere e si prende un file dalla coda. Per ogni riga del file si aggiunge il campo letto alla lista. Finito il ciclo for le liste contengono tutti i campi del file.

Algorithm 2 Lettura righe file

```
1: inizializzazione di una lista per ogni campo
2: file ←Coda.get()                          ▷ prende un file dalla coda
3: for riga in file do
4:     Assegna ad ogni lista il valore letto dalla riga
```

4.2.3 Conversione dei dati

Lo pseudocodice 3 descrive il processo di conversione che si effettua sui valori letti nel codice 2. La data si ricava dal nome delle cartelle scomponendo il path, la figura 4.10 mostra l'acquisizione della data partendo dal percorso del file. Per ogni riga viene poi fatta una addizione dei pacchetti e dei byte in entrata e in uscita.

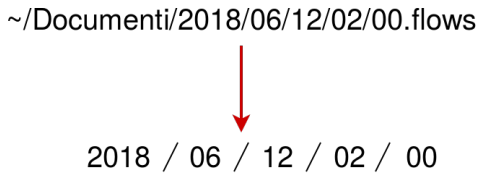

Figura 4.10: Acquisizione data dal percorso del file

Algorithm 3 Conversione dei campi

```
1: scomposizione del percorso file per ricavare la data
2: while i < lunghezza del file do
3:     last ←last_switched[i]          ▷ assegna il valore i-esimo alla variabile
4:     first ←first_switched[i]
5:     tot_pkts ←in_pkts[i] + out_pkts[i]
6:     tot_bytes ←in_bytes[i] + out_bytes[i]
```

4.2.4 Scrittura su file

La figura 4.11 mostra il procedimento di scrittura su file. Questo passaggio viene effettuato finita la conversione dei file.

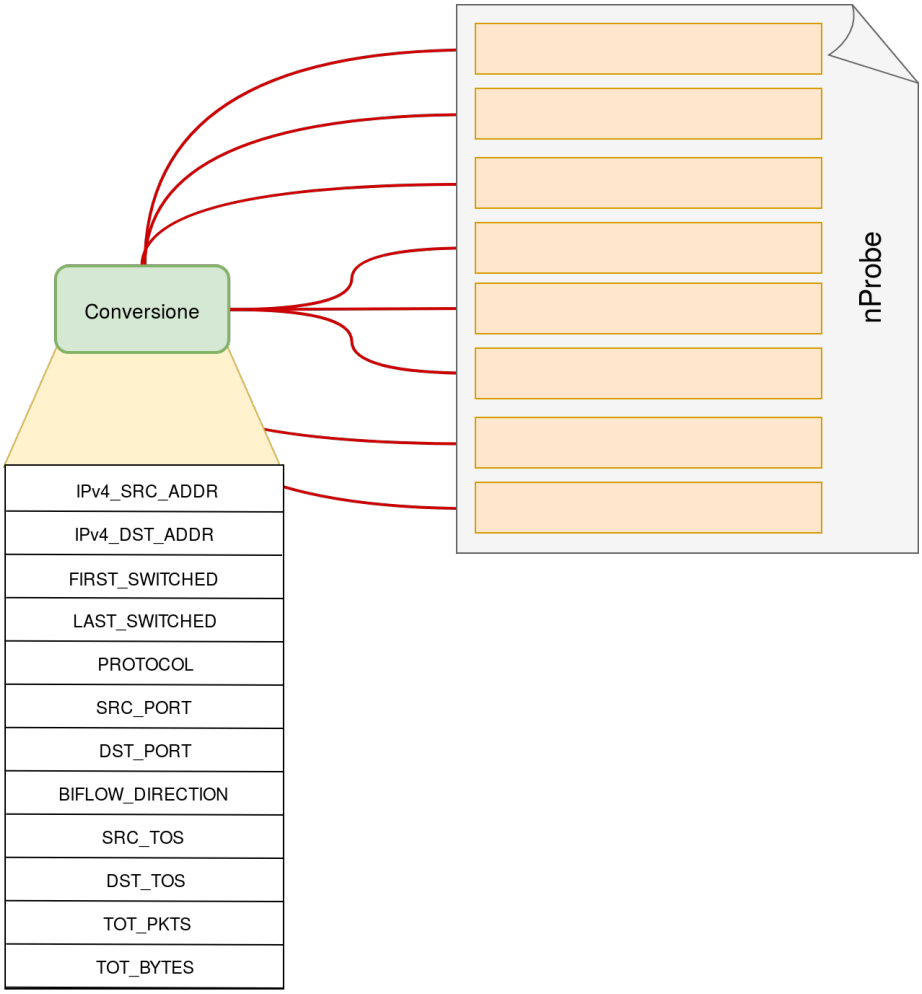

Figura 4.11: Scrittura su file nProbe

Lo pseudocodice 4 descrive il processo di scrittura su file. All'interno di un ciclo *while* si effettua una scrittura di tutti i campi convertiti nel passaggio precedente.

Algorithm 4 Scrittura su file

1: **while** $i < lunghezzadel file$ **do**
2: scrive la conversione su file

4.2.5 Algoritmo completo

La figura 4.12 mostra lo schema completo per la risoluzione dell'algoritmo.

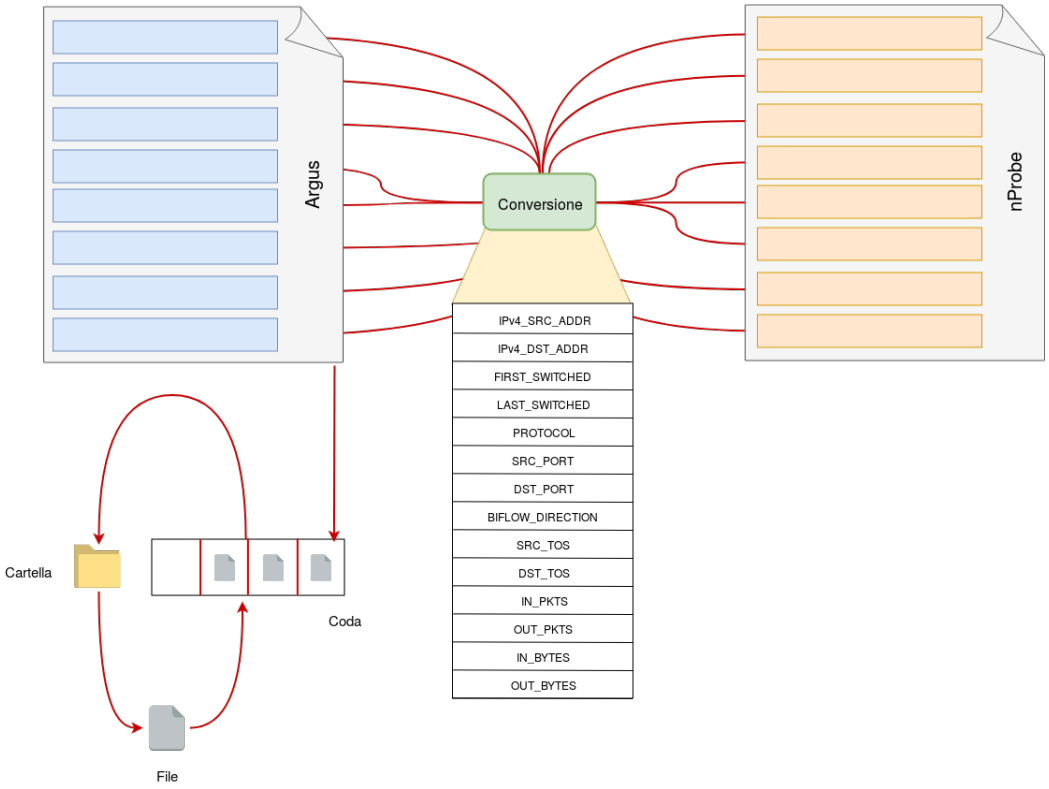

Figura 4.12: Schema completo

L'algoritmo 5 scritto in pseudocodice descrive una soluzione single core al problema

Algorithm 5 Pseudocodice completo

```
1: cartellaRoot ←pathCartella
2: Coda ←Queue()                                   ▷ inizializza una coda
3: for mesi, giorni, minuti in cartellaRoot do
4:     for file in minuti do
5:         Coda.append() ←file                     ▷ inserisce il file nella coda
6: inizializzazione di una lista per ogni campo
7: file ←Coda.get()                                ▷ prende un file dalla coda
8: for riga in file do
9:     Assegna ad ogni lista il valore letto dalla riga
10: scomposizione del percorso file per ricavare la data
11: while i < lunghezza del file do
12:     last ←last_switched[i]                     ▷ assegna il valore i-esimo alla variabile
13:     first ←first_switched[i]
14:     tot_pkts ←in_pkts[i] + out_pkts[i]
15:     tot_bytes ←in_bytes[i] + out_bytes[i]
16: while i < lunghezzadelfile do
17:     scrive la conversione su file
```

4.3 Rendere efficiente la conversione

Nel programma descritto in precendenza viene generato un solo file di output in cui vengono convertiti tutti i file dati in input. Questa soluzione è comoda perchè da migliaia di file si ottiene un solo file con i dati convertiti, ma presenta il problema di creare un file con dimensioni enormi e di difficile gestione (il file può raggiungere dimensioni tali da rendere difficile anche solo aprirlo in lettura).

Il programma è inoltre inefficiente poichè è single core e ha come collo di bottiglia la scrittura su un unico file. La soluzione proposta seppure sia teoricamente corretta non può avere un'applicazione nel mondo reale. Bisogna cambiare quindi strategia per rendere la conversione più veloce sfruttando le macchine multi core e per avere in output file di dimensioni accettabili.

4.3.1 Possibili soluzioni

Si possono pensare diverse soluzioni che migliorerebbero in modo significativo il programma visto in precedenza:

- Lavorare su chunk di file
- Meccanismi di lock
- Scrittura su file 1:1

Lavorare su chunk di file Per velocizzare il programma e sfruttare i processori disponibili si potrebbe assegnare ad ogni processore un file da leggere e convertire. Quando il processore termina la conversione dei dati scrive sul file in output. In questo modo si dividrebbe il tempo di esecuzione sul numero di processori disponibili. Questa soluzione rende il più veloce possibile la conversione dei file ma i processori finiscono per scrivere sullo stesso file senza avere nessuna regola di precedenza, questo crea problemi perchè le scritture in output non sono ordinate e non è possibile creare modelli comportamentali affidabili. Le scritture su file devono essere ordinate e sequenziali, inoltre questa soluzione ha il problema di scrivere sempre su unico file e, come detto in precedenza, questo tipo di soluzione non è possibile.

Meccanismi di lock Un modo per risolvere i problemi precedenti è quello di utilizzare il concetto di semaforo. In informatica un semaforo è un tipo di dato astratto gestito da un sistema operativo multitasking per sincronizzare l'accesso a risorse condivise tra processi. È composto da una variabile intera e da una coda di processi. Quando un processo apre il file per scriverci viene impostato un semaforo che segnala che la risorsa è occupata, se un altro processore prova ad aprire lo stesso file per scriverci gli sarà negato l'accesso dal semaforo fino a quando l'altro processo non rilascerà la risorsa. In questo modo si risolve il problema delle scritture ordinate, ma c'è da tener conto che i semafori riducono la velocità di esecuzione dell'algoritmo poichè mentre un processore occupa la risorsa tutti gli altri processori devono mettersi in coda per aspettarne il rilascio. Nonostante questa soluzione rallenti l'esecuzione del programma rispetto alla soluzione precedente è comunque molto più veloce della versione single core perchè, sebbene la scrittura su file sia rallentata dai semafori, si guadagna molto tempo nella lettura e conversione dei dati dove non ne è necessario l'utilizzo, e che quindi viene effettuata alla massima velocità possibile. Questa soluzione risolve anche il problema dell'ordinamento delle scritture. Rimane il problema della scrittura su un unico file che però può essere risolto facilmente decidendo di scrivere su un nuovo file quando raggiunge una dimensione specificata. Se si

implementa la divisione del file di output questa soluzione può considerarsi efficiente anche se rimane il collo di bottiglia introdotto dai semafori che costringe i processi ad aspettare in coda quando una risorsa è impegnata.

Scrittura su file 1:1 In questa soluzione ogni processore apre un file, lo legge, lo converte e scrive su un proprio file. Questa soluzione è decisamente la più semplice tra quelle proposte ed è anche la più efficiente poichè i processori non entrano mai in conflitto tra di loro cosicchè da effettuare letture, conversioni e scritture alla massima velocità possibile dalla macchina. Un problema che può creare questa soluzione è la produzione di una grande quantità di file in output. Si pensi che una sola settimana di traffico di rete, sono circa 10 mila file.

4.3.2 Scelta effettuata

Si è scelto di procedere con l'ultima soluzione presentata perchè data la grande quantità di informazioni da convertire si preferisce l'approccio più veloce a discapito dell'ordinamento finale.

Per realizzare questa soluzione si effettua una lettura analoga dei file vista nella sezione 4.2.1 per creare una coda che contiene il percorso di tutti i file presenti in un cartella. Si chiama poi una funzione che prende un numero designato di core da assegnare ad una funzione. Ogni core prende un file all'interno della coda, lo legge, lo converte e scrive su un file in modo indipendente dagli altri. La figura 4.13 mostra la parallelizzazione dell'algoritmo.

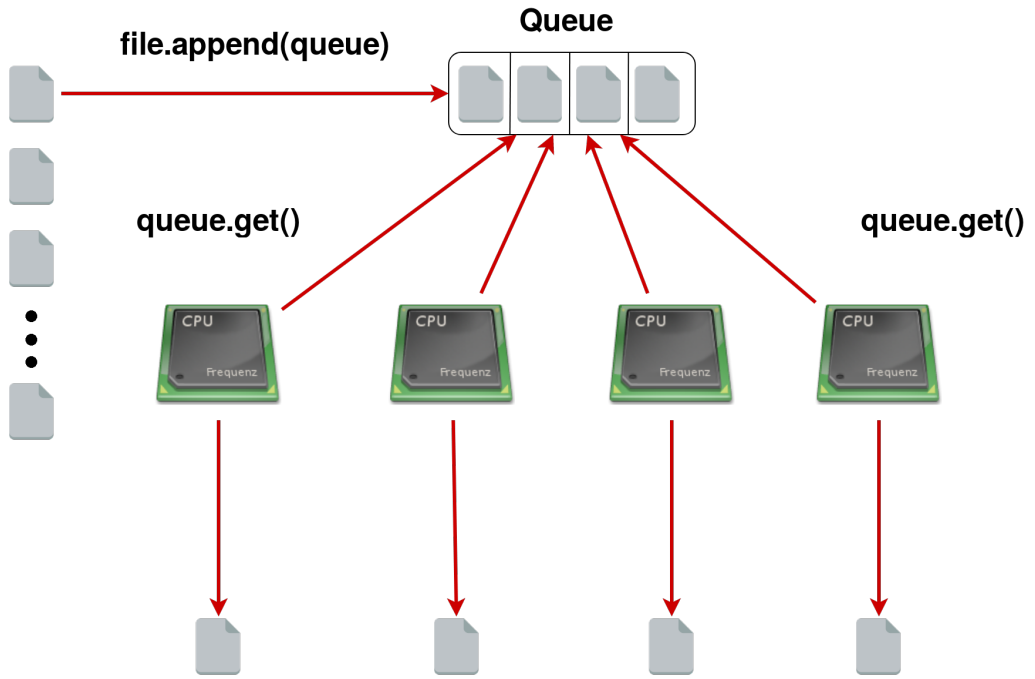

Figura 4.13: Multi core version

Lo pseudocodice 6 descrive questa soluzione.

Algorithm 6 Multi core version

```
1: procedure HYDRA
2:     for all file in path do
3:         Queue[] ← file
4:         legge i dati dal file
5:         prende n-core
6:         while Queue[] not empty do
7:             filename ←Queue.get()
8:             converte i dati nel nuovo formato
9:             scrive i dati in un nuovo file
```

4.4 Correzione del codice sorgente

Il codice sorgente è stato modificato per poter effettuare una installazione funzionante. Per la scrittura di questa tesi si è scaricata la versione più aggiornata disponibile sulla pagina GitHub di slips [40]. Questa versione, essendo in alpha, non funziona *out of the box*. È stato perciò necessario apportare delle modifiche al codice.

Per provare il corretto funzionamento di slips si è passato per prima cosa un file di flows in input per verificarne il corretto funzionamento. Il codice 4.2 mostra il comando utilizzato

```
cat file.binetflow | ./slips.py -f models -d
```

Listing 4.2: comando per eseguire slips

L'output generato dal comando 4.2 è vuoto per qualsiasi tipo di file.

Il codice 4.3 rappresenta la funzione *run* del file *slips.py*. Questo pezzo di codice viene eseguito su più processori: ogni core prende una riga dalla coda fino a quando non si svuota ed esegue il codice all'interno del try. La riga numero 5 prende una riga del file dalla coda e alla riga numero 7 controlla se il contenuto della riga letta è diverso da *stop* per continuare. Questo controllo, inserito probabilmente in fase di test per eseguire la funzione su pezzi di codice prestabiliti, va sostituito con un controllo che esegue il codice solo se la riga non è vuota.

Questo pezzo di codice ha inoltre dei controlli di debug che eseguono la funzione solamente se la riga letta ha un particolare indirizzo IP o una porta. Alla riga numero 8 c'è un controllo sugli indirizzi IP 10.0.2.15 e 31.13.91.6 e alla riga numero 11 un controllo sulla porta 80 e indirizzo IP 10.0.2.15. Questi controlli sono stati tolti per permettere alla funzione di eseguire su qualsiasi indirizzo IP e porta.

```
1  def run(self):
2      try:
3          while True:
4              if not self.queue.empty():
5                  line = self.queue.get()
6                  #print "IN THE PROCESS AT:{} flow: *{}*".format(
       datetime.now(), line)
7                  if 'stop' != line:
8                      if '10.0.2.15' in line or '31.13.91.6' in line:
9                          # Process this flow
10                         column_values = self.parsingfunction(line)
11                         if column_values[7] == "80" or column_values[6]
       == "'10.0.2.15'":
12                             print(column_values[6], column_values[7])
13                             try:
```

Listing 4.3: codice originale

Il codice 4.4 mostra le modifiche apportate al codice 4.3. È stato risolto anche un bug che lascia in esecuzione il programma quando finisce di eseguire. La riga numero 3 del codice 4.3 non permetteva alla funzione di uscire dal ciclo *while*, è stato risolto con la riga numero 3 del codice 4.4.

```
1      def run(self):
```

```
2         try:
3             while not queue.empty():
4                 line = self.queue.get()
5                 #print "IN THE PROCESS AT:{} flow: *{}*".format(
    datetime.now(), line)
6                 if line != '':
7                     # Process this flow
8                     column_values = self.parsingfunction(line)
9                     #print(column_values[6], column_values[7])
10                    try:
```

Listing 4.4: codice modificato

Capitolo 5

Sperimentazioni

In questo capitolo vengono presentati gli esperimenti effettuati e le prestazioni ottenuti dallo script di conversione, successivamente i test fatti con Stratosphere IPS sulla creazione di modelli comportamentali per verificare l'accuratezza della conversione dei flows.

5.1 Realizzazione di una piattaforma dedicata

Per lo sviluppo del programma per la conversione dei file e per l'utilizzo di Stratosphere IPS è stata creata una piattaforma dedicata alla Security Analytics tramite l'utilizzo di *VirtualBox* con una installazione del sistema operativo Ubuntu 16.04 LTS.

> **VirtualBox** è un software gratuito e open source per l'esecuzione di macchine virtuali che supporta Windows, GNU/Linux e macOS come sistemi operativi host ed è in grado di eseguire Windows, GNU/Linux, OS/2 Warp, BSD come ad esempio OpenBSD, FreeBSD e infine Solaris e OpenSolaris come sistemi operativi guest [46].

5.2 Installazione Stratosphere IPS

Sulla macchina virtuale è stato installato il framework di Stratospehere IPS. Di seguito i passaggi effettuati per l'installazione [41].

- Installazione del programma git 2.7.4 [13]

```
sudo apt install git
```

Il pacchetto *git* è necessario per interagire con il repository GitHub su cui è presente Stratosphere.

- Clonazione repository GitHub del framework

```
git clone https://github.com/stratosphereips/
   StratosphereTestingFramework
```

- Installazione del programma python-pip [30]

```
sudo apt install python-pip
```

Questo pacchetto serve per installare i pacchetti di Python e verrà usato per installare pacchetti successivi.

- prettytable 0.7.2-3 [31]

```
sudo apt install python-prettytable
```

PrettyTable è una libreria di Python usata per rappresentare tabelle ASCII.

- transaction 1.4.3-3 [44]

```
sudo apt install python-transaction
```

Questo pacchetto contiene l'implementazione delle transazioni per Python ed è usato principalmente da *Zodb*.

- persistent 4.1.1-1build2 [28]

```
sudo apt install python-persistent
```

Il pacchetto contiene l'implementazione della persistenza dei dati per database come *Zodb*.

- zodb 5.4.0 [51]

```
sudo pip install zodb
```

È il database ad oggetti utilizzato dal framework.

- sparse 1.1-1.3build1 [36]

```
sudo apt install python-sparse
```

Libreria che fornisce vettori multi dimensionali sparsi.

- dateutil 2.4.2-1 [11]

```
sudo apt install python-dateutil
```

Modulo che fornisce un'estensione al modulo standard di Python *datetime*.

5.3 Installazione di Argus

Per generare i netflow dal traffico di rete è necessario avere una istanza di Argus sul computer. I seguenti passaggi sono necessari per la corretta installazione di Argus [41].

- libpcap 1.7.4-2

```
sudo apt install libpcab-dev
```

- bison 3.0.4

```
sudo apt install bison
```

- flex 2.6.0-11

```
sudo apt install flex
```

- Installazione dell'ultima versione di argus 3.0.8.2 dal sito http://qosient.com/argus/dev/argus-latest.tar.gz
- Installazione dell'ultima versione di argus-client 3.0.8.2 dal sito http://qosient.com/argus/dev/argus-clients-latest.tar.gz

5.4 Utilizzo del programma STF

In questa sezione viene presentato il framework di Stratoshpere le la creazione dei modelli comportamentali, STF.

Per eseguire il programma si usa il comando

```
./stf.py
```

La figura 5.1 mostra l'interfaccia a console del programma STF

```
    Stratosphere Testing Framework
            _   __
           | | / _|
        ___| |_| |_
       / __| __|  _|
       \__ \ |_| |
    ... |___/\__|_|  ...
    0.1.2alpha


[*] Amount of experiments in the DB so far: 0
[*] Amount of datasets in the DB so far: 0
[*] Amount of groups of connections in the DB so far: 0
[*] Amount of groups of models in the DB so far: 0
[*] Amount of notes in the DB so far: 0
stf >
```

Figura 5.1: Interfaccia a console di STF

Per caricare un dataset si utilizza il comando

```
datasets -c /absolute/path/file.binetflow
```

Per generare la connessione si utilizza il comando

```
connections -g
```

Infine, per generare i modelli, il comando

```
models -g
```

Per visualizzare il behavioral model si utilizza il comando

```
models -L [id]
```

5.5 Tempi di esecuzione

Come visto nel capitolo 4, il programma di conversione proposto è efficiente. In questa sezione vengono effettuati dei *benchmark* per analizzarne le prestazioni e studiare la scalabilità di una soluzione che prevede l'utilizzo di più CPU in parallelo.

5.5.1 Benchmark

Per le analisi sono stati usati dei dataset di network flow generati da una rete di grandi dimensioni con migliaia di host attivi quotidionamente. Per misurare i tempi di esecuzione del programma sono stati usati 3 dataset:

- 10 giorni di flows di dimensione totale 1,7 GB

- 5 giorni di flows di dimensione totale 700 MB
- 1 giorno di flows di dimensione totale 70 MB

Le misure sono state realizzate con il programma /usr/bin/time. I grafici sono stati creati con il programma GNU Octave per GNU/Linux [14].

Il computer su cui sono stati effettuati i benchmark ha le seguenti caratteristiche:

- Linux Mint 19 Cinnamon
- Kernel 4.15.0-20-generic
- Processore Intel Core i7-3770 @ 3.40GHz
- RAM 8 GB
- HDD 2 TB 7200 RPM

I benchmark sono stati effettuati convertendo il dataset con l'algoritmo proposto nel capitolo 4. I benchmark sono stati ripetuti in funzione del numero di core disponibili sulla CPU. I tempi di esecuzione sul dataset di 1,7 GB sono rappresentati in minuti, secondi e centesimi:

- 1 core: 11:35:46
- 2 core: 6:02:65
- 3 core: 4:12:62
- 4 core: 3:09:79
- 5 core: 3:04:47
- 6 core: 3:04:66
- 7 core: 2:59:22
- 8 core: 2:59:33

Come è possibile constatare dai risultati, per i primi 4 core si ha un guadagno lineare, mentre gli ultimi 4 forniscono un miglioramento non più lineare. Questo è dovuto al numero di core della CPU usata per i benchmark, l'i7-3770 ha 4 core fisici e 4 virtuali sfruttando la tecnologia *Hyper Threading* [17] di Intel.

La figura 5.2 rappresenta il grafico che è stato costruito sui risultati dell'esecuzione dei 3 diversi dataset: ha sull'asse delle x il numero di core e sull'asse delle y

il tempo impiegato per convertire i file. Il colore blu identifica il dataset di dimensione 1,7 GB, il colore rosso 700 MB e quello verde 70 MB. Come si può vedere il tempo impiegato scende in modo quasi lineare per i primi quattro core nel dataset più grosso. Il grafico verde rimane quasi costante perchè per dataset di dimensioni molto ridotte aggiungere core è quasi ininfluente.

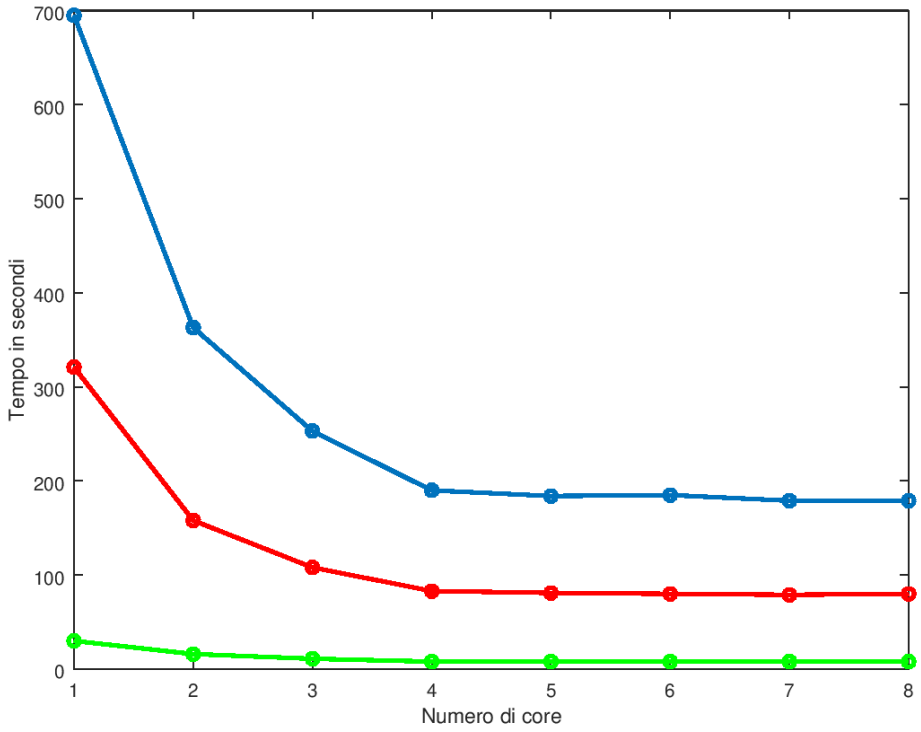

Figura 5.2: Andamento del tempo in funzione del numero di core

5.5.2 Metrica delle prestazioni parallele

Sia T(p) il tempo di esecuzione in secondi di un certo algoritmo su p processori. Di conseguenza sia $T(1)$ il tempo di esecuzione del codice parallelo su 1 processore. La *misura di scalabilità* o *speedup* relativo di un algoritmo parallelo eseguito su p processori si calcola come:

$$S(p) = \frac{T(1)}{T(p)}$$

In un sistema ideale, in cui il carico di lavoro potrebbe essere perfettamente partizionato su p processori, lo speedup relativo dovrebbe essere uguale a p. In questo caso si parla di speedup lineare.

Con i risultati ottenuti in precedenza si è calcolato lo speedup per ogni numero di core. La figura seguente mostra l'andamento dello speedup relativo in funzione del numero di core. Si vede come la curva è inizialmente prossima ad uno speedup lineare, poi si ha un punto di saturazione dovuto ai costi di comunicazioni tra i vari cori che cominciano a prevalere sugli altri costi. Si ha che

$$\lim_{p\to\infty} S(p) = 0$$

C'è una soglia, che dipende dall'algoritmo e dall'architettura del sistema, oltre la quale è controproducente aumentare il numero di core. La figura 5.3 descrive l'andamento dello speedup con i risultati ottenuti.

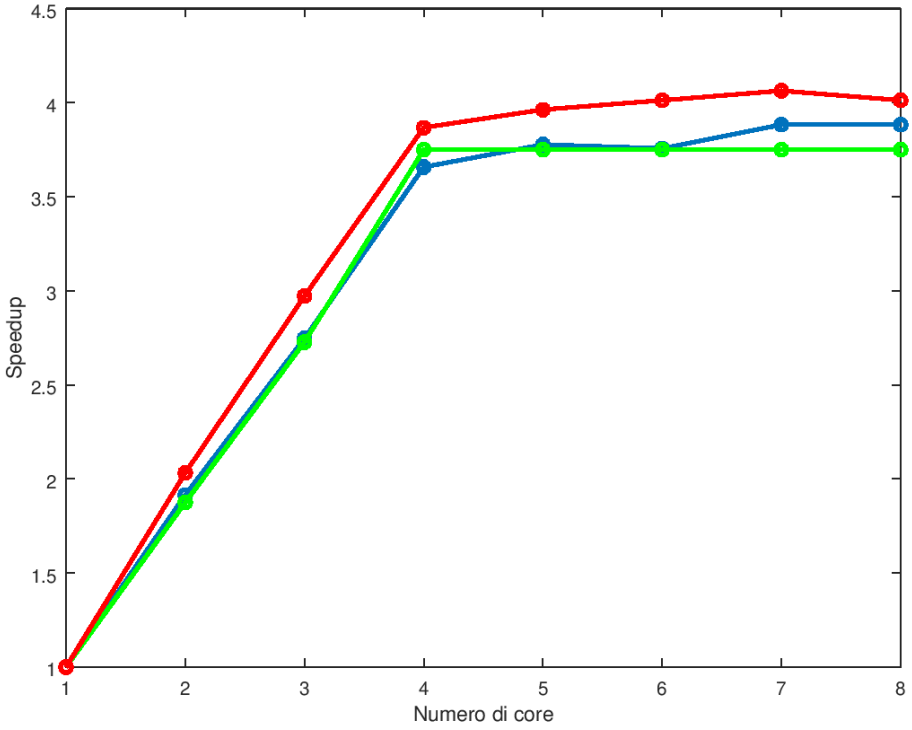

Figura 5.3: Andamento dello speedup in funzione del numero di core

Si definisce *efficienza* il rapporto

$$E(p) = \frac{S(p)}{p}$$

Idealmente, se l'algoritmo avesse uno speedup lineare, si avrebbe $E(p) = 1$. Nella pratica $E(1) = 1$ mentre $E(p)$, per $p > 1$, è una funzione decrescente. Più l'efficienza

si allontana da 1, peggio stiamo sfruttando le risorse di calcolo disponibili nel sistema parallelo.

La figura 5.4 mostra l'andamento dell'efficienza del programma di conversione al variare del numero di processori. Usando quattro core si ha un'efficienza leggermente maggiore di 0.9 mentre con l'aggiunta di altri core si può notare chiaramente come l'efficienza cali.

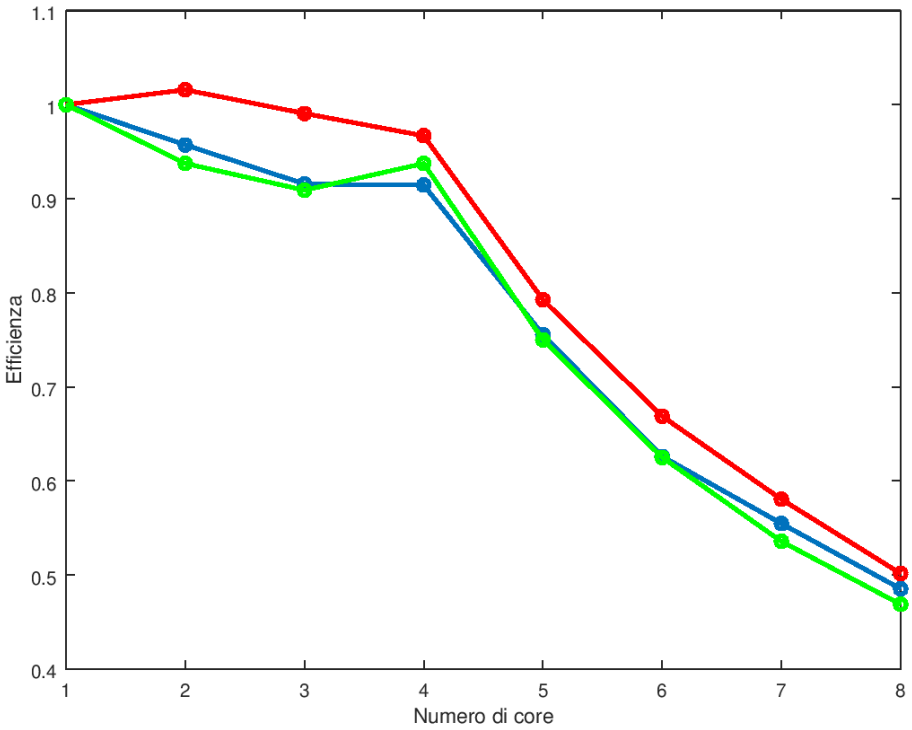

Figura 5.4: Andamento dell'efficienza in funzione del numero di core

Per determinare il numero ottimale di core da utilizzare per un certo algoritmo bisogna cercare il numero di core con efficienza e speedup maggiore. Si osserva che:

- $E(p)$ ha il massimo per $p = 1$
- $S(p)$ ha il massimo in corrispondenza del punto di saturazione, in cui però l'efficienza è piuttosto bassa

Per trovare il numero ottimo si usa la *funzione di Kuck*

$$F(p) = E(p)S(p)$$

Questa funzione restituisce un numero che mette in relazione l'efficienza e lo speedup. Il numero ottimale di core p_F con cui eseguire un algoritmo è il numero in corrispondenza del massimo della funzione di Kuck.

$$p_F = argmax F(p)$$

Nella figura 5.5 è rappresentata la funzione di Kuck

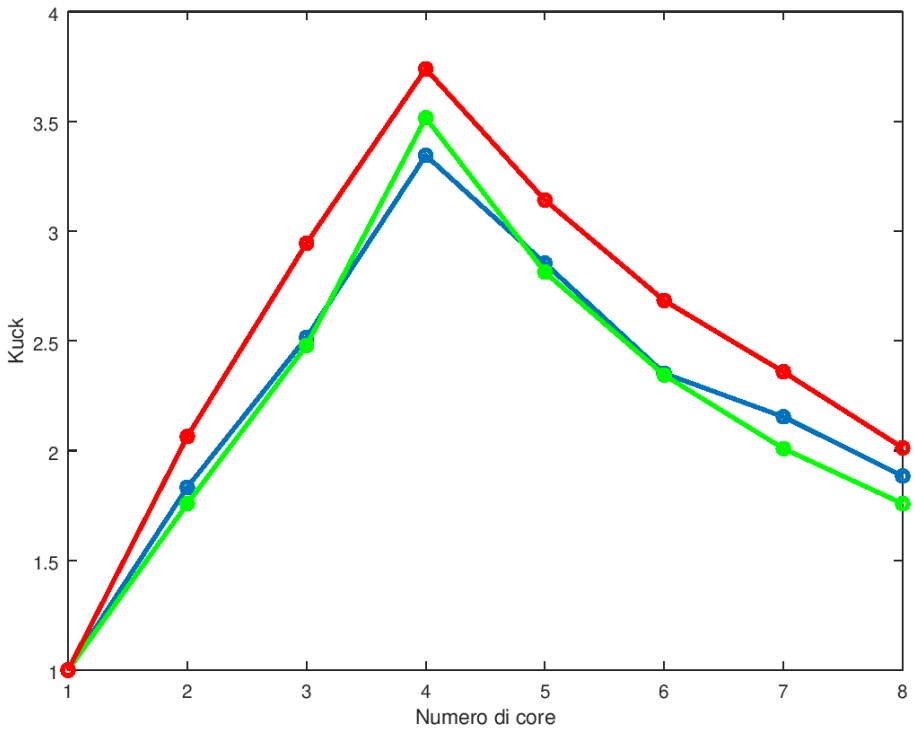

Figura 5.5: Funzione di Kuck al variare del numero di core

La conversione in parallelo risulta molto efficiente, intorno allo 0.9. Con gli esperimenti effettuati ci si aspetta uno speedup lineare sul numero di core fisici. Utilizzando un solo core la conversione impiega 12 minuti, mentre con quattro core si divide in quattro il tempo arrivando a 3 minuti. Le CPU di nuova generazione come AMD Ryzen vantano 32 core fisici e 32 virtuali, con questo numero di core fisici ci si può aspettare di dividere il tempo per 32, arrivando ad effettuare la conversione in 23 secondi.

5.6 Precisione della Conversione

Per controllare se la conversione viene effettuata correttamente si è usato il software Stratosphere IPS per creare e utilizzare dei modelli impiegati per fare *anomaly detection* su del traffico di rete. I flows sono stati convertiti nel formato compatibile da Argus ad nProbe e infine di nuovo ad Argus per controllare che non ci siano delle perdite di informazioni vitali che alterino il funzionamento dell'IDS.

La figura 5.6 descrive il processo di conversione. Ad ogni conversione effettuata è prevista una lieve perdita di informazioni dovuta al tipo di campi utilizzati nei diversi formati e alla loro formattazione.

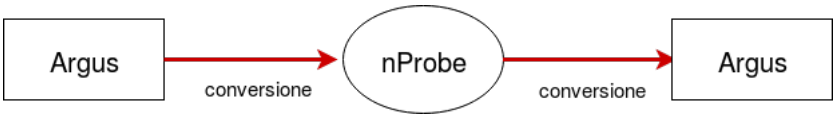

Figura 5.6: Processo di conversione

Sono stati usati dei flow dati in input a slips per rilevare del traffico malevolo, successivamente convertiti in un altro formato e riconvertiti per verificare che l'output sia identico pre e post conversione. I flows e i modelli comportamentali utilizzati per il test sono stati presi dal repository GitHub di slips [39].

Nel test effettuato si è utilizzato il file "2016-11-4_win11.binetflow" con il comando 5.1

```
1 cat test-flows/Malicious/2016-11-4_win11.binetflow | ./slips.py -f
    models/ -d
```

Listing 5.1: comando di test

l'output di slips è rappresentato in figura 5.8. Sono stati rilevati 3 indirizzi IP malevoli dal file di flow dato in input.

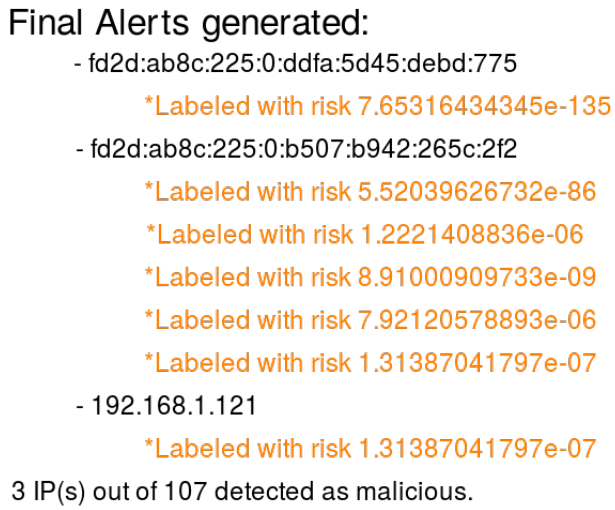

Figura 5.7: Output slips - file originale

Ci si aspetta quindi di ottenere lo stesso output dopo il processo di conversione, rilevando gli stessi IP malevoli con lo stesso valore di rischio. Per controllare se la perdita di informazioni è significativa o meno si esegue nuovamente il test utilizzando il file convertito con il comando 5.1. L'output di slips è rappresentato in figura 5.8

Figura 5.8: Output slips - file convertito

I due output sono identici, si può quindi dire con certezza che, nonostante sia presente una perdita di informazioni, questa non altera il rilevamento di slips.

Capitolo 6

Conclusioni

In questa tesi si è analizzato un software di Intrusion Detection System basato su machine learning. Inizialmente si è discusso dei vantaggi che porta la cattura dei pacchetti per monitorare lo stato di una rete e la loro importanza nella prevenzione e risposta ad attacchi di tipo informatico. Quindi stati studiati dei programmi (Argus e nProbe) che rappresentano lo stato dell'arte per la cattura del traffico di rete e successiva generazione di network flows, mostrandone le caratteristiche e mostrando i pro e i contro di ciascuna soluzione. In particolare, è stato messo in evidenza come questi programmi – nonostante si prefissino il medesimo obiettivo – consentino di catturare informazioni differenti, risultando in formati di file diversi in output. Pertanto, al fine di facilitare l'adozione di entrambi i software per la produzione e successiva analisi di network flow in contesti specifici, è stato proposto un apposito algoritmo di conversione bidirezionale. Tale algoritmo è stato progettato in modo tale da rispettare due requisiti fondamentali nell'ambito dell'analisi dei dati: limitare la perdita di informazioni durante la conversione; effettuare l'esecuzione in tempi rapidi e compatibili con analisi online.

La correttezza della conversione è stata verificata utilizzando lo stesso file convertito tra due formati per fare anomaly detection.

Per garantire un'elevata efficienza del programma, sono stati utilizzati concetti di calcolo parallelo che hanno contribuito drasticamente ad incrementare la velocità della conversione. Le prestazioni raggiunte introducendo concetti di calcolo parallelo, misurate attraverso un grosso sono ottime e si è dimostrato che è possibile raggiungere uno speedup lineare sui core fisici della macchina, che nello sviluppo di programmi in parallelo è un'obiettivo importante in quanto dimostra che il numero di core è sfruttato bene e in modo efficiente. Con il rapido sviluppo e miglioramento della tecnologia ci si aspetta che le CPU diventino sempre più veloci e potenti, rendendo l'esecuzione dell'algoritmo proposto in questa tesi sempre più veloce.

Le principali difficoltà riscontrate durante la realizzazione di questa tesi, hanno avuto a che fare con l'analisi di Stratosphere. Questo software infatti, essendo pubblicato come alpha è ancora in fase sperimentale. Le difficoltà riscontrate sono state superate con un'analisi approfondita per capirne il funzionamento e tramite dei processi di debugging. Nella tesi viene esposta una guida *step-by-step* su come effettuare il deployment di un sistema basato su Stratosphere.

Possibili punti su cui basare eventuale lavoro futuro sono l'estensione dell'algoritmo di conversione anche ad altri formati, con possibilità di creare un tool di conversione universale. Inoltre, si ritiene interessante estendere Stratosphere IPS integrando il supporto ad altri algoritmi di machine learning orientati all'analisi comportamentale.

Bibliografia

[1] https://pliki.ip-sa.pl/wiki/Wiki.jsp?page=NetFlow.

[2] Alpha software definition. https://techterms.com/definition/alpha_software.

[3] Anatomy of a botnet. https://web.archive.org/web/20170201233855/https://www.scribd.com/document/179124526/Anatomy-of-a-Botnet-WP-pdf. Accessed: 2017-02-01.

[4] Apple explains how 'hey siri' works using a deep neural network and machine learning. https://9to5mac.com/2017/10/18/how-hey-siri-works/.

[5] Argus. https://qosient.com/argus/.

[6] Argus – audit record generation and utilization system. https://en.wikipedia.org/wiki/Argus_%E2%80%93_Audit_Record_Generation_and_Utilization_System.

[7] Atis telecom glossary - audit trail. https://web.archive.org/web/20130313232104/http://www.atis.org/glossary/definition.aspx?id=5572.

[8] Big data security analytics: A weapon against rising cyber security attacks? https://bi-survey.com/big-data-security-analytics.

[9] Bitcoin. https://bitcoin.org/it.

[10] Comparison of machine learning techniques in email spam detection. https://dev.to/matchilling/comparison-of-machine-learning-techniques-in-email-spam-detection.

[11] dateutil - python. https://pypi.org/project/python-dateutil/.

[12] Federal trade commission- consumer information, "malware". https://www.consumer.ftc.gov/articles/0011-malware. Accessed: November 2015.

[13] git - distributed is the new centralized. https://git-scm.com/about.

[14] Gnu octave. https://www.gnu.org/software/octave/.

[15] Google assistant and artificial intelligence – the future of simulated communication. https://campaignsoftheworld.com/digital/google-assistant-and-artificial-intelligence/.

[16] How tesla is ushering in the age of the learning car. http://fortune.com/2015/10/16/how-tesla-autopilot-learns/.

[17] Hyper - threading. https://it.wikipedia.org/wiki/Hyper-Threading.

[18] Inside the infamous mirai iot botnet: A retrospective analysis. InsidetheinfamousMiraiIoTBotnet:ARetrospectiveAnalysis.

[19] Intrusion detection and prevention: More than a firewall. https://searchcio.techtarget.com/tip/Intrusion-detection-and-prevention-More-than-a-firewall.

[20] Machine learning in practice: How does amazon's alexa really work? https://www.forbes.com/sites/bernardmarr/2018/10/05/how-does-amazons-alexa-really-work/#227507bf1937.

[21] Making facial recognition smarter with artificial intelligence. https://www.forbes.com/sites/jenniferhicks/2018/09/30/making-facial-recognition-smarter-with-artificial-intelligence/#565964e1c8f1.

[22] Malware definition. https://techterms.com/definition/malware. Accessed: 27-04-2016.

[23] Malware revolution: A change in target. https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc512596(v=technet.10). Published: 05/20/2008.

[24] Mcafee: due sole botnet per (quasi) tutto lo spam del mondo. https://www.punto-informatico.it/mcafee-due-sole-botnet-per-quasi-tutto-lo-spam-del-mondo-2/.

[25] Netflow - wikipedia. https://en.wikipedia.org/wiki/NetFlow.

[26] nprobe. https://www.ntop.org/products/netflow/nprobe/.

[27] Optical character recognition. https://en.wikipedia.org/wiki/Optical_character_recognition.

[28] persistent = python. https://pypi.org/project/persistent/.

[29] Phishing definition - urban dictionary. https://www.urbandictionary.com/define.php?term=phishing.

[30] pip - python. https://pypi.org/project/pip/.

[31] prettytable - python. https://pypi.org/project/PrettyTable/.

[32] Queue - python docs. https://docs.python.org/2/library/queue.html.

[33] Sicurezza: Virus, worm, trojan... http://www.bloomriot.org/91/sicurezza-virus-worm-trojan.html.

[34] Snort - network intrusion detection and prevention system. https://www.snort.org/.

[35] Solarwinds. https://www.solarwinds.com/network-performance-monitor.

[36] sparse - python. https://pypi.org/project/sparse/.

[37] spyware. https://searchsecurity.techtarget.com/definition/spyware. Accessed: september 2016.

[38] Stratosphere ips - protecting the civil society through high quality research. https://www.stratosphereips.org/stratosphere-ips-suite.

[39] Stratosphere ips for linux. https://github.com/stratosphereips/StratosphereLinuxIPS.

[40] Stratosphere linux ips latest commit. https://github.com/stratosphereips/StratosphereLinuxIPS/tree/d9f42edf728dc7be21a5f441b6851e24071b966a.

[41] Stratospheretestingframework - github.

[42] Suricata - open source ids. https://suricata-ids.org/.

[43] Tojan horse definition. https://techterms.com/definition/trojanhorse.

[44] transaction - python. https://pypi.org/project/transaction/.

[45] Video ad fraud botnet bags up to 1.3 million daily. https://www.bankinfosecurity.com/video-ad-fraud-botnet-generated-up-to-13-million-day-a-10470.

[46] Virtualbox. https://www.virtualbox.org/.

[47] What is a trojan virus? - definition. https://usa.kaspersky.com/resource-center/threats/trojans#.VvD3eOLhDtQ.

[48] What is spyware? - definition. https://www.kaspersky.com/resource-center/threats/spyware.

[49] What is zero day exploit? https://www.kaspersky.com/resource-center/definitions/zero-day-exploit.

[50] Wikipedia, the free encyclopedia - pcap. https://en.wikipedia.org/wiki/Pcap.

[51] Zodb - a native object database for python. http://www.zodb.org/en/latest/.

[52] "mining" botnets are back - infecting thousands of pcs, generating hundreds of thousands of dollars for criminals. https://www.kaspersky.com/about/press-releases/2017_mining-botnets-are-back-infecting-thousands-of-pcs.

[53] WikiLeaks Vault7: Archimedes documentation. https://wikileaks.org/vault7/#Archimedes, visited in Jun. 2017.

[54] Analysis of the Cyber Attack on the Ukrainian Power Grid. http://www.nerc.com/pa/CI/ESISAC/Documents/E-ISAC_SANS_Ukraine_DUC_18Mar2016.pdf, visited in Jun. 2018.

[55] Apache Spot. http://spot.incubator.apache.org/, Jun. 2018.

[56] DarkTrace. https://www.darktrace.com/en/press/2017/204/, visited in Sep. 2018.

[57] IBM Watson. https://www.ibm.com/security/artificial-intelligence, Visited in Sep. 2018.

[58] Internet Security Threat Report 2018. https://www.symantec.com/security-center/threat-report, visited in Jun. 2018.

[59] VirusTotal. https://www.virustotal.com, visited in Aug. 2018.

[60] K. Rajalakshmi A. Hyils Sharon Magdalene. Intrusion detection prevention system using acl and aaa. Technical report, ManonmaniamSundaranar University, Tirunelveli India, April 2014.

[61] Giovanni Apruzzese, Mirco Marchetti, Michele Colajanni, Gabriele Gambigliani Zoccoli, and Alessandro Guido. Identifying malicious hosts involved in periodic communications. In *Network Computing and Applications (NCA), 2017 IEEE 16th International Symposium on*. IEEE, 2017.

[62] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, University of Technology Goteborg Sweden, 14 March 2000.

[63] Dennis Brown. Resilient botnet command and control with tor. *DEF CON*, 18, 2010.

[64] D. Karahoca E. Alparslan, A. Karahoca. Botnet detection: Enhancing analysis by using data mining techniques. Technical report, September 2012.

[65] U. Aickelin J. Greensmith. Firewall, intrusion detection systems and anti-virus scanners. Technical report, University of Nottingham, UK, February 2005.

[66] Witold Pedrycz (eds.) Janusz Kacprzyk. *Springer Handbook of Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 1 edition, 2015.

[67] A. Manasrah M. Kadhum K. Alieyan, A. Almomani. A survey of botnet detection based on dns. Technical report, University Sains Malaysia, Al-Balqa' Applied University, December 2015.

[68] K. D. R. Assis R. C. Almeida L. P. Dias, J. J. F. Cerqueira. Using artificial neural network in intrusion detection systems to computer networks. Technical report, Federal University of Bahia, Salvador Brazil, September 2017.

[69] Herbert J. Mattord Michael E. Whitman. *Principles of Information Security*. Course Technology, 4 edition, 2011.

[70] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, 1 edition, 1997.

[71] G. Ruth N. Brownlee, C. Mills. Traffic flow measurement: Architecture. Technical report, University of Auckland, October 1999.

[72] Robert Newman. *Computer Security: Protecting Digital Resources*. 2009.

[73] M. Araghizadeh P. Amini, R. Azmi. Botnet detection using netflow and clustering. Technical report, Malek-Ashtar University of Technology Tehran Iran, Alzahra University Tehran Iran, University of Tehran, March 2014.

[74] R. Cunningham C. Zou P. Wang, L. Wu. Honeypot detection in advanced botnet attacks. Technical report, University of Central Florida, North Carolina State University, February 2010.

[75] Mark Stamp Peter Stavroulakis. *Handbook of Information and Communication Security*. Springer, 1st edition. edition, 2010.

[76] A. Singh Brar S. Behal, K. Kumar Saluja. Signature-based botnet detection and prevention. Technical report, Deptt of CSE Ferozepur India, Deptt of CSE Ludhiana India.

[77] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[78] Danielle Anne Veluz. Tequila botnet leads to phishing attack. `http://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/48/tequila-botnet-leads-to-phishing-attack`, June 2010.

[79] David Dagon Wenke Lee, Cliff Wang. *Botnet Detection: Countering the Largest Security Threat (Advances in Information Security)*. Springer, softcover reprint of hardcover 1st ed. 2008 edition, 2010.