POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Monitoraggio e classificazione di traffico cifrato con TLS

Relatori
prof. Cataldo Basile
prof. Antonio Lioy

Candidato
Simone Pastore

Relatore esterno
sig. Danilo Massa

Anno accademico 2017-2018

† Al nonno Aldo, alla nonna Rosanna e allo zio Lelle

Ringraziamenti

Un ringraziamento e un pensiero va alle persone che mi sono state accanto in questi anni universitari:

- a mio padre e a mia madre, che mi hanno permesso di non poter far altro che pensare a studiare, dato che a tutto il resto hanno pensato loro;
- alla mia fidanzata, che ha contribuito a trasmettermi la felicità anche nei momenti più difficili;
- ai miei nonni e zii, che mi hanno supportato moralmente e psicologicamente durante il percorso;
- ai miei compagni di studio e di corso, che hanno contribuito alla mia crescita personale e caratteriale;
- ai ragazzi della mia squadra di calcio, che mi hanno motivato a non mollare e a continuare a perseguire i miei obiettivi;
- a tutti i dipendenti di Aizoon, che mi hanno dato la possibilità di svolgere la tesi presso di loro e di prendere parte alle loro dinamiche aziendali.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Obiettivi di tesi

La tesi si è svolta presso un'azienda di consulenza informatica sita in Torino di nome Aizoon, presso il loro ufficio di Cyber Security.

L'obiettivo della mia tesi è il monitoraggio passivo di traffico SSL/TLS col fine di classificarne il tipo di traffico cifrato trasportato. La tesi è stata strutturata in diversi step.

Per prima cosa ho dovuto raccogliere informazioni attraverso l'analisi della situazione reale circa la classificazione di traffico over SSL/TLS. Per fare ciò ho letto alcuni articoli oppure tesi già scritte che mi sono state passate dalle persone che mi seguivano presso l'azienda oppure che ho cercato personalmente su Google Scholar. Successivamente vi è stata la formulazione del problema, con la conseguente identificazione delle variabili e delle loro relazioni, per poi andare a definire una strategia di raccolta e pulizia dei dati, e quindi la suddivisione di essi in classi opportune. Dopodiché vi è stata l'individuazione di alcuni modelli algoritmici utili alla causa, la fase di progetto e conseguente stesura dal punto di vista del codice operativo. Per finire vi è stata l'analisi e la verifica dei prototipi sviluppati al fine di valutarne la bontà complessiva attraverso parametri di valutazione.

Il campo di applicazione della mia tesi è la cosiddetta Network Forensics, una disciplina che si occupa di registrare il traffico che passa ad esempio attraverso il perimetro aziendale per poi andare a classificarlo o a studiarlo in modo da comprendere la tipologia di traffico. Tutto ciò può servire per esempio a condurre alcune indagini statistiche oppure dopo un attacco informatico in modo da risalire all'identità dell'attaccante, alla sua posizione geografica o al suo indirizzo IP.

Più in generale questa tesi prende forma perché si vuole andare a cercare una metodologia per poter riuscire a classificare il traffico che transita all'interno dell'azienda. É utile in questo senso un classificatore che ci possa dire che tipo di traffico sta transitando soprattutto per motivi di sicurezza, in modo da rendersi conto se per esempio è passato traffico malevolo che potrebbe minare la sicurezza dei sistemi, ma anche per motivi statistici, ad esempio per capire quali siano i tipo di traffico che vengono più utilizzati in determinati periodi di tempo, o per motivi di validazione, cioè se sto mandando un certo tipo di traffico mi aspetto che

il mio classificatore me lo classifichi in un certo modo. In particolar modo questo tipo di studio vorrebbe prendere forma all'interno del prodotto Aramis, sviluppato all'interno dell'azienda presso cui è stata svolta, appunto per poter raggiungere gli obiettivi menzionati sopra.

1.1.1 Outline

Nel capitolo 2 verranno decritti tutti gli strumenti di cui ho bisogno per poter iniziare e portare avanti il lavoro. Infatti verranno descritti il protocollo SSL/TLS e come funziona, il protocollo SSH (che ci servirà per una classe di traffico in particolare), l'analizzatore di rete Bro e il suo funzionamento, anche all'interno del contesto generale di Aramis, sistema che permette il monitoraggio della propria rete interna.

Nel capitolo 3 verrà descritto lo stato dell'arte con tutto ciò che è stato fatto finora riguardo la classificazione di traffico cifrato, e quindi verrà esposta una sintesi di tutti gli articoli e lavori di tesi che sono stati letti, facendo intendere come è stato preso spunto da questi lavori.

Nel capitolo 4 verrà descritto il punto di partenza della soluzione, con la dichiarazione dei parametri che sono stati valutati essere utili.

Nel capitolo 5 invece risiederà la parte operativa del mio esperimento, in cui si descriverà la cattura dei campioni delle classi e la parte di generazione della tabella contenente per ogni connessione la lista dei valori dei parametri, che sarà poi mandata in input a diversi algoritmi di machine learning e che produrranno dei risultati circa la loro bontà di classificazione.

Nel capitolo 6 verranno quindi presentati e discussi i risultati di questi algoritmi, con l'ultima parte dedicata alle conclusioni.

Nell'appendice A invece saranno presenti i vari algoritmi utilizzati, a partire dagli script in bash e in linguaggio Bro per poi finire con quelli di machine learning. Sono anche presenti le istruzioni per l'installazione di Bro e Stubby.

1.2 Perché utilizzare il traffico cifrato?

Si può constatare come il tema della sicurezza sia antico quanto il mondo, poiché da sempre l'uomo ha cercato il modo di proteggere i suoi segreti e la sua vita privata. Durante la storia si è passati attraverso diverse tecniche crittografiche, mentre attualmente ci sono parecchi algoritmi di crittografia che usano la grande potenza di calcolo dei moderni calcolatori elettronici. Al giorno d'oggi attuare la cifratura è di vitale importanza poiché molte attività, come operazioni bancarie oppure comunicazioni militari, devono garantire la riservatezza delle informazioni scambiate, per impedire ai sempre più numerosi eventuali attaccanti di impossessarsi di informazioni vitali e sensibili.

1.2.1 Cos'è la crittografia?

La crittografia è un metodologia che ci permette di cifrare, di nascondere i contenuti sensibili come ad esempio conversazioni, informazioni o parti di codice. Per poter supportare la crittografia un sistema ha bisogno di una chiave, che deve essere mantenuta segreta nel caso di chiave privata oppure distribuita nel caso di chiave pubblica, e un algoritmo, che non sempre viene mantenuto segreto. Il testo in chiaro da cifrare, tramite le istruzioni dettate dall'algoritmo, viene combinato con la chiave in modo da ottenere il testo cifrato, pronto quindi per essere mandato ad un'eventuale controparte. Spesso il traffico cifrato viene interpretato come un black-box, o scatola nera, poiché se si vuole andare a vedere cosa contiene un pacchetto contenente traffico cifrato si vedranno solo una serie di simboli di ogni tipo che sembrano non volerci dire nulla. Solo conoscendo la chiave e algoritmo di decifratura si può risalire al messaggio originale che avrà quindi un senso. I pacchetti cifrati nel tragitto dal mittente al destinatario passano attraverso un numero molto elevato di nodi intermedi, i quali ad esempio instradano e smistano traffico secondo alcuni criteri, come fanno i router. Ecco che per prevenire il caso di qualche malintenzionato che voglia provare a leggere o intercettare le informazioni che vengono scambiate viene utilizzata la cifratura, che ci permette di avere una sorta di protezione contro alcuni tipi di possibili attacchi e attaccanti.

Capitolo 2

Background

2.1 Il protocollo SSL/TLS

Il protocollo TLS [1] è un protocollo di trasporto sicuro situato circa al livello sessione, che permette l'autenticazione del server (con l'opzione non obbligatoria sul client), la riservatezza della comunicazione, l'autenticazione e l'integrità dei messaggi, la protezione da filtering e dall'attacco replay, ed è quindi applicabile a tutti i protocolli basati sul protocollo di livello trasporto TCP, quindi ad esempio HTTP, SMTP, FTP ecc. La Figura 2.1 e la Figura 2.2 mostrano la fase di handshake del protocollo TLS, in cui le linee tratteggiate indicano messaggi opzionali.

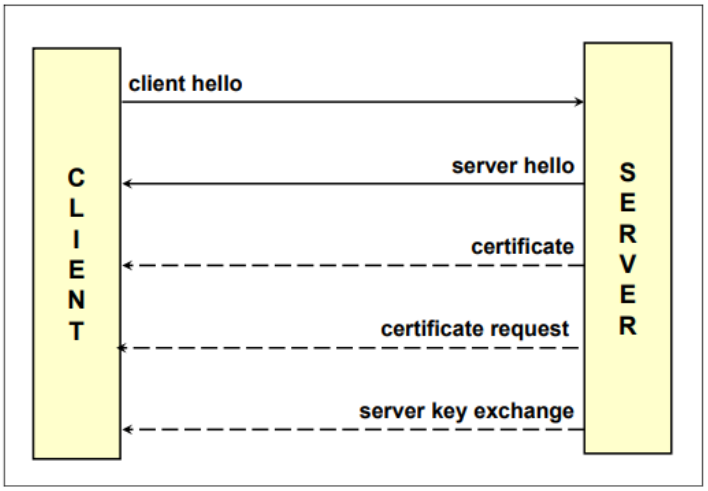

Figura 2.1. Funzionamento fase di handshake - parte 1 (fonte: security.polito.it).

Di seguito vengono elencati i singoli messaggi durante la fase di handshake e le relative caratteristiche:

Client hello porta con sé la versione di TLS preferita dal client, 28 byte pseudo-casuali (Client Random), un session-id (in cui 0 identifica la nuova sessione,

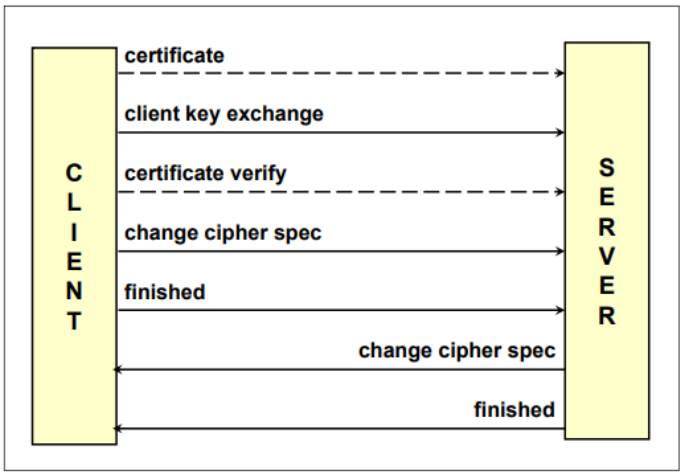

Figura 2.2. Funzionamento fase di handshake - parte 2 (fonte: security.polito.it).

mentre un numero diverso da 0 chiede di riesumare una sessione precedente), la cipher suite (algoritmi di cifratura, scambio chiavi e integrità) ed eventuali metodi di compressione supportati dal client;

Server hello porta con sè la versione di SSL scelta dal server, 28 byte pseudo-casuali (Server Random), il session-id (uno nuovo se quello inviato dal client era 0, oppure rifiuta quello del client, oppure lo accetta se era diverso da 0 e quindi viene riesumata una sessione precedente), la cipher suite scelta (di solito quella più forte in comune col client) e l'eventuale metodo di compressione scelto;

Certificate è il certificato del server, può essere utilizzato solo per firma oppure anche per cifratura. É rappresentato come tratteggiato in figura dal momento che se viene ripresa una sessione precedentemente attivata allora non esiste più il bisogno di rimandare il certificato alla controparte (il client) in quanto era già stato fatto in precedenza;

Certificate request serve per chiedere il certificato per autenticare il client, con una lista delle CA che il server ritiene fidate, quindi il client sa già che se il suo certificato non è stato firmato da una delle CA elencate allora la sua identità non potrà essere validata;

Server key exchange porta con sè la chiave pubblica (utilizzata per lo scambio della chiave) del server, nel caso ad esempio il server abbia fornito un certificato solo per la firma;

(Client) Certificate trasporta il certificato del client;

Client key exchange ci sono varie possibilità, tra cui fargli trasportare il pre-master secret cifrato con la chiave pubblica del server, cosicché solo il server potrà andare a decifrarlo. Per capire bene come funziona il rapporta tra

le chiavi e le sessioni e per capire come vengono generate le chiavi per gli algoritmi vedere la Figura 2.3;

Certificate verify che sarebbe la prova esplicita di firma del client in caso venga richiesta l'autenticazione del client, in quanto contiene l'hash firmato con la chiave privata del client di tutti i messaggi di handshake che precedono questo;

Change cipher spec è valido sia per il client che per il server e permette di passare dai messaggi in chiaro della fase di handshake a quelli cifrati della comunicazione vera e propria;

Finished anche questo valido sia per il client che per il server ed è il primo messaggio che viene protetto dagli algoritmi e dalle chiavi negoziate finora, ed è decisivo per validare tutto lo scambio in quanto contiene un MAC calcolato su tutti i messaggi scambiati fino ad ora.

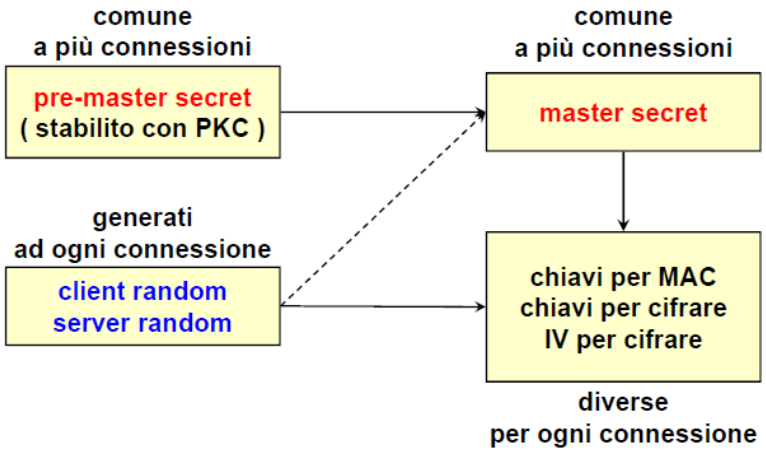

Figura 2.3. Dettaglio di rapporto tra chiavi e sessione tra il server e uno stesso client (fonte: security.polito.it).

Una volta che è stata completata la fase di handshake inizia lo scambio di messaggi e quindi la vera conversazione tra le due controparti, client e server, svolta in maniera protetta con algoritmi e chiavi concordate nella fase di handshake. Come vedremo, nella fase di cattura del traffico tramite il tool Wireshark, tutti i messaggi che vengono scambiati durante la fase di handshake precedente allo scambio di pacchetti cifrati sono catturati e visualizzati, e il primo pacchetto, solitamente il Client hello, è quello che ci permette di andare a rilevare il numero di porta che viene aperta, che ci serve per poi salvare tutta la relativa connessione. Quando invece una delle due controparti decide di chiudere la connessione allora manda una notifica di chiusura (close notify) col protocollo SSL alert, ed attende la medesima risposta dal suo interlocutore.

2.1.1 Sessioni e connessioni

Occorre, prima di iniziare il lavoro di tesi, avere bene in mente la differenza tra connessione e sessione. Infatti una sessione è una associazione logica tra client e

server, viene creata dal protocollo di handshake e può contenere N connessioni. Invece una connessione è un canale sicuro temporaneo tra client e server ed è associata ad una specifica sessione. Procedendo nella tesi si vedrà come il mio compito sia proprio quello di andare ad effettuare l'assegnazione delle etichette, cioè il tagging delle connessioni e non delle sessioni, in quanto è possibile che in una sessione io possa trovare più connessioni contenenti tipo di traffico o dati diversi, mentre in una connessione generalmente no.

2.2 Il protocollo SSH

Il protocollo SSH permette di stabilire una sessione cifrata sicura con un host di una rete informatica, e funziona tramite un'interfaccia a linea di comando. La porta su cui passa questo tipo di servizio è la 22, dove appunto avviene la mutua autenticazione tra client e server, in cui chi inizia la comunicazione agisce da client verso una postazione remota che agisce da server. Questo protocollo viene utilizzato soprattutto per il trasferimento di file in maniera fidata e, come vedremo, verrà sfruttato per poter popolare ad analizzare una classe di traffico, appunto la ssh.

2.3 Introduzione a Bro

Il tool Bro [2], utilizzato all'interno dell'azienda presso cui ho svolto l'esperimento di tesi, cattura il traffico sotto forma di pacchetti per poi mandarlo verso una coda (potrebbe anche filtrare) in attesa di essere analizzata e processata. Questo strumento lavora in maniera simile a quanto fa Wireshark, il quale cattura il traffico sotto forma di pacchetti, ma permette poi un loro filtraggio in base alle esigenze dell'utente che lo sta utilizzando, il quale a sua volta imposterà dei filtri, per poi arrivare allo step finale consistente nel salvataggio dei pacchetti selezionati all'interno di un file, che il più delle volte ha come estensione .pcap o similari. Questo file contiene appunto la lista ordinata per tempo di arrivo dei pacchetti che l'utente ha deciso di isolare. Bro riesce però a fare di più: infatti, dopo aver sniffato i pacchetti in rete, essi passano attraverso un event engine, il quale in base alle peculiarità del traffico o dei pacchetti analizzati può scatenare una serie di eventi correlati appunto al tipo di traffico. Questi eventi, gestiti via software da un serie di event handler sotto forma di script, eseguono del codice per poi registrare opzionalmente dei file di log che descrivono ciò che è accaduto oppure i parametri che sono stati giudicati essere interessanti in un flusso di traffico. Ad esempio, se durante la fase di sniffing viene rilevato del traffico HTTP, allora verrà scatenato l'evento corrispondente e verrà prodotto un file di log di nome http.log contenente tutte le connessioni http che sono state avviate nella suddetta sessione di traffico, e per ogni connessione avrò ad esempio l'identificativo di connessione, l'indirizzo ip sorgente e destinazione, le porte sorgenti e destinazione, il numero di byte scambiati, o insomma tutte le informazioni che all'interno dello script sono state giudicate essere utili e quindi da salvare.

Il linguaggio di scripting di Bro è simile al C, e si può imparare utilizzando un manuale che permetta con una facile consultazione di capire quali sono i tipi, le

strutture e i principali paradigmi di questo linguaggio. Invece per poter fare pratica con il linguaggio di Bro, esiste un sito[1] che ci permette di provare a scrivere codice in Bro, compilarlo e mandarlo in esecuzione, con esercizi divisi per argomento e con difficoltà via via crescente. É stato un passo di vitale importanza iniziare a scrivere alcuni semplici script in Bro per poi iniziare a complicarli, non tanto per capire codice già scritto, ma quanto per provare a scrivere del codice funzionante e utile al mio obiettivo di tesi.

Bro permette anche di potergli passare come input dei file .pcap già fatti, già catturati in diversa sede oppure da qualche altro tool come Wireshark, poiché poi ci fornisce come output alcuni file di log con le caratteristiche di traffico insite all'interno del file in ingresso. Leggendo la documentazione e alcuni documenti online [3] [4] si capisce più nel dettaglio la struttura e il funzionamento di Bro, che vengono esplicate nella Figura 2.4 e più nel dettaglio nella Figura 2.5.

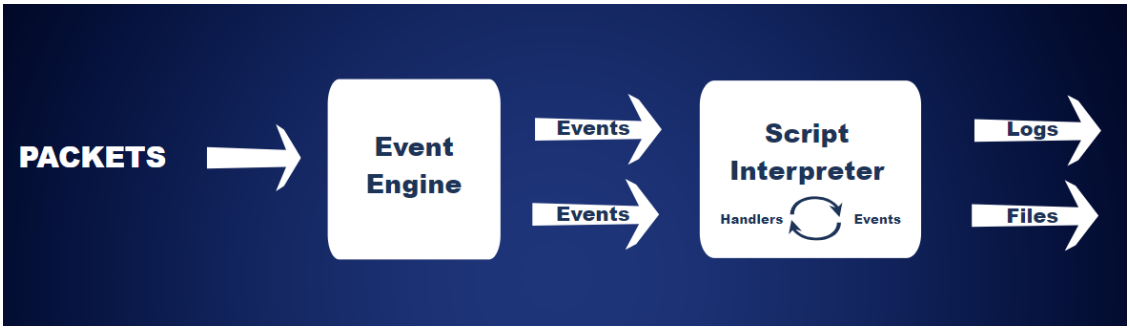

Figura 2.4. Core di Bro (fonte: bro.org).

Per concludere questa introduzione, Bro attua anche dei compiti di analisi e controllo, come la validazione dei certificati e la detection di malware. Bro è comunque modificabile a piacere ed estendibile, anche se non è un classico IDS perché supporta un più ampio spettro di approccio di analisi.

2.4 Vagrant

Durante il lavoro di tesi mi è stato consigliato di usare il software Vagrant, strumento che permette di fare il deploy di una macchina virtuale molto più velocemente di quanto può fare VirtualBox, altro software di virtualizzazione che è stato utilizzato per la parte operativa della tesi. Vagrant utilizza il concetto di virtualizzazione, che ci permette di astrarre l'hardware di un computer affinché questo sia disponibile come risorsa virtuale, e si pone in un livello intermedio tra il sistema ospitante (host) e il sistema virtuale ospitato (guest), che può essere anche più di uno. Vagrant ha il grosso vantaggio, rispetto a VirtualBox, di poter evitare di condividere macchine virtuali da diversi gigabyte (che quindi richiedono diverso tempo) tra diversi gruppo di lavoro.

[1] try.bro.org

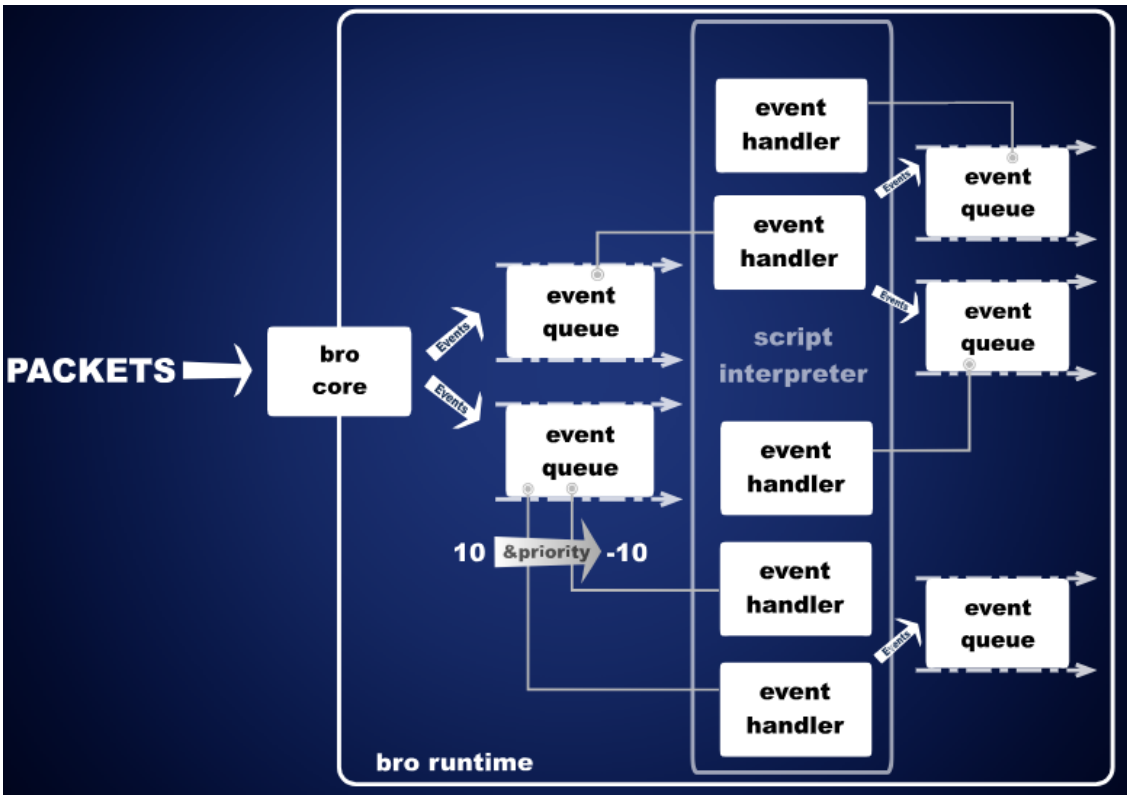

Figura 2.5. Architettura di Bro dettagliata (fonte bro.org).

Installare Vagrant richiede per prima cosa di scaricare un sistema di virtualizzazione come VirtualBox. É giusto aggiungere che è stato anche utilizzato una sorta di clone del comando apt-get di Linux che si chiama Chocolatey e che permette di installare e far partire da linea di comando Windows alcuni comandi che sarebbero solamente disponibili in ambiente Linux. Con i comandi dati da terminale:

```
vagrant init ubuntu/xenial64
vagrant up
vagrant ssh
vagrant halt
```

viene inizializzato un nuovo progetto Vagrant con un sistema operativo Ubuntu con la distribuzione Xenial a 64 bit, e viene creato un Vagrantfile (modificabile secondo le esigenze) il quale racchiude tutte le informazioni della macchina virtuale di cui ci apprestiamo a fare il deploy, come ad esempio la quantità di RAM, le configurazioni di rete e tanto altro ancora. Dopodiché viene fatta partire la macchina virtuale e gli viene fatto accesso con il penultimo comando, mentre con l'ultimo viene fermata. In Vagrant vengono creati diversi contenitori, in cui ognuno contiene una macchina virtuale, un Vagrantfile e un file JSON contenente metadati. A partire da un box originario possono essere fatti partire diversi box clonati, e ogni modifica li renderà diversi dal box originario. La Figura 2.6 contiene un esempio di due macchine virtuali (box) clonate contenenti una distribuzione Ubuntu a partire da un box originario.

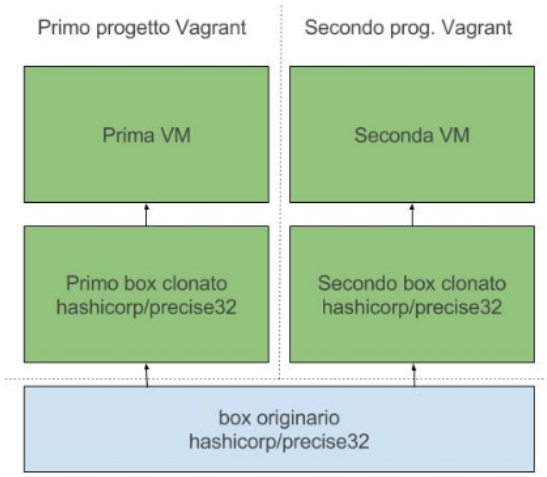

Figura 2.6. Esempio di box originari e box clonati (fonte html.it).

2.5 Installazione ed esecuzione di Bro

Come detto prima per la parte operativa, cioè quella di scrittura ed esecuzione del codice, è stata utilizzata una macchina virtuale Linux con Ubuntu 18.04, a cui sono stati assegnati 30 gigabyte di memoria disco e 4 gigabyte di RAM, anche se tutte le operazioni si potevano svolgere su Vagrant, che è stato più che altro utilizzato per la prima installazione di bro, e per prendere confidenza con le sue caratteristiche e potenzialità.

Per quanto riguarda l'installazione di Bro (che è un software open-source) su Ubuntu sono state eseguite da linea di comando alcune direttive, e che vengono mostrate in appendice A.

2.5.1 Esempio di output di Bro

Tramite il tool che online ci permette di inserire un pcap e studiare l'output che fornisce Bro, proviamo a inviargli come file di input uno contentente traffico generico, e vediamo che in output vengono forniti diversi file: conn.log, files.log, http.log, software.log, stats.log. Molti di essi ci danno poche informazioni o comunque ci dicono cose che non ci interessano molto. Vediamo nella Figura 2.7 l'output del file conn.log che contiene tutte le connessioni e nella Figura 2.8 l'output del file http.log, il quale contiene una riga per ogni connessione http. Come si può vedere tutti i parametri hanno nome, tipo e valore, e il significato è facilmente comprensibile a partire dal loro nome. Come vedremo, i file di output sono modificabili a piacere, nel senso che è possibile aggiungere delle righe con campo, tipo e valore oppure addirittura creare dei nuovi file di log con nuovi tipi al loro interno, ed è proprio quello che faremo per tutti i valori di attributi che sono stati selezionati e che non compaiono già nei file di log di Bro.

Field	Type	Value
ts	time	1258563305.873659
uid	string	CegqyZ2qtCRRiJgOge
id.orig_h	addr	192.168.1.103
id.orig_p	port	1391
id.resp_h	addr	151.207.243.129
id.resp_p	port	80
proto	enum	tcp
service	string	http
duration	interval	22.203564
orig_bytes	count	2575
resp_bytes	count	1694
conn_state	string	S3
local_orig	bool	T
local_resp	bool	T
missed_bytes	count	0
history	string	ShADadf
orig_pkts	count	16
orig_ip_bytes	count	3609
resp_pkts	count	10
resp_ip_bytes	count	2102

Figura 2.7. Screenshot di una connessione nel file conn.log (fonte try bro).

2.6 Aramis

La soluzione che ho proposto verrà attuata all'interno di Aramis, che è un sistema avanzato di intelligenza artificiale che consente un monitoraggio dei rischi della propria rete. Questa piattaforma identifica minacce ancora sconosciute e mette in luce le debolezze. Quattro sono i suoi pilastri fondamentali: il motore di apprendimento automatico per l'identificazione di anomalie; l'identificazione rapida di pattern malevoli; threat intelligence e visualizzazione cognitiva dei dati per fornire immediata evidenza degli allarmi.

Aramis è formato da macro blocchi, in cui ognuna svolge la propria funzionalità. La parte denominata ADS è presente lato sonda, contiene Bro, che è stateless, prende solo dati in real time, mentre i dati non real time vengono mandati all'ALS. Quest'ultimo è lato server, e possiede un database. Vi è poi una parte fuori dalla intranet, l'ACS, che esegue gli aggiornamenti e propaga le informazioni alle altre due parti.

Field	Type	Value
ts	time	1258538143.400927
uid	string	COntIg2Ak9hZfpNG
id.orig_h	addr	192.168.1.103
id.orig_p	port	1208
id.resp_h	addr	212.227.96.110
id.resp_p	port	80
trans_depth	count	1
method	string	POST
host	string	212.227.96.110
uri	string	/rpc.html?e=bl
referrer	string	-
version	string	1.1
user_agent	string	SCSDK-6.0.0
request_body_len	count	984
response_body_len	count	96
status_code	count	200
status_msg	string	OK
info_code	count	-
info_msg	string	-
tags	set[enum]	(empty)

Figura 2.8. Screenshot di una connessione http nel file http.log (fonte try bro).

Capitolo 3

Stato dell'arte

3.1 Articoli e tesi consultate

Non si trovano in rete grosse quantità di studi che prendano in considerazione il traffico cifrato. Infatti, ricevere un pacchetto contenente informazioni cifrate, è come ricevere una scatola che non si può aprire in alcun modo se non si conosce la chiave, e quindi la decisione più facile e comoda da prendere potrebbe essere scartarla oppure lasciare ad altri l'arduo compito di provare a comprenderla a fondo.

L'obiettivo della fase iniziale del lavoro di tesi è stato concentrarmi sulla lettura di articoli e tesi già redatte da altri alla ricerca di parametri e attributi per poter classificare il traffico cifrato, nonché metodi, algoritmi o procedure utili per poter iniziare a impostare un punto di partenza per il successivo lavoro di raccolta e classificazione del traffico.

Una ricerca che fa da ponte tra il traffico non cifrato e cifrato [5] sottolinea come una tecnica che permette la classificazione di traffico non cifrato sia la Deep Packet Inspection (DPI), che agisce a livello 7. Divide il traffico in sensibile (positivo importante), best-effort (positivo non importante) e indesiderato (negativo), per poi andare a definire metodi e parametri di classificazione di traffico non cifrato quali la dimensione del pacchetto, la durata di una connessione, le porte, il contenuto del payload e statistiche sul traffico. Viene concluso che bisognerebbe anche andare a classificare il traffico cifrato perché ogni tipo di traffico ha qualità e caratteristiche diverse. Nonostante questo sia traffico non cifrato ho utilizzato con profitto alcuni parametri di classificazione di questo traffico per andare ad etichettare anche il traffico cifrato.

Per quanto riguarda il machine learning una ricerca [6] ci mostra come utilizzare un algoritmo di machine learning supervised: la Support Vector Machine (SVM). Tuttavia, queste macchine sono influenzate dalla scala dei dati, dalla dimensione delle caratteristiche e dai parametri del classificatore. La classificazione è basata su numeri di porta (non più tanto utilizzato), DPI (ispeziona il payload del pacchetto alla ricerca di firme note, anche se ci sono oggi tante applicazioni non standard tali che la firma sia difficile da trovare, e poi ce un grosso overhead), analisi del protocollo (analizza le caratteristiche e il comportamento del flusso, ma è difficile da usare a causa di protocolli cifrati e applicativi non standard) e tecniche di machine

learning (utilizza algoritmi di machine learning diversi, che costruiscono un modello di classificazione da un training set di istanze etichettate, anche se purtroppo si denota un alto costo computazionale e parametri dei classificatori che cambiano di frequente). Un primo obiettivo è ottimizzare la performance di una SVM, e questo può essere ottenuto andando ad operare un'ottimizzazione dei parametri, il data scaling, in cui i dati vengono scalati in un range più piccolo per aumentare la velocità di classificazione, e infine la feature extraction, in cui viene costruito un nuovo sistema di coordinate per ottenere le caratteristiche essenziali dei campioni originali. Tutto ciò porta a lavorare questa SVM migliorata facendole ottenere prestazioni superiori in termini di accuratezza e tempo di esecuzione rispetto alla SVM normale, al K-Nearest Neighbors e al Naive Bayes. Uno schema di massima del suo funzionamento è rappresentato in Figura 3.1. Come vedremo nell'ultima parte della tesi, la fase di pre-processing caratteristica di alcuni algoritmi di machine learning ci porta ad ottimizzare e a scalare alcuni parametri con apposite funzioni, in quanto diversamente avremmo parametri diversi che possono avere valori anche molto distanti tra loro, e a quel punto il classificatore avrebbe non pochi problemi a costruirsi un proprio modello che sia anche robusto.

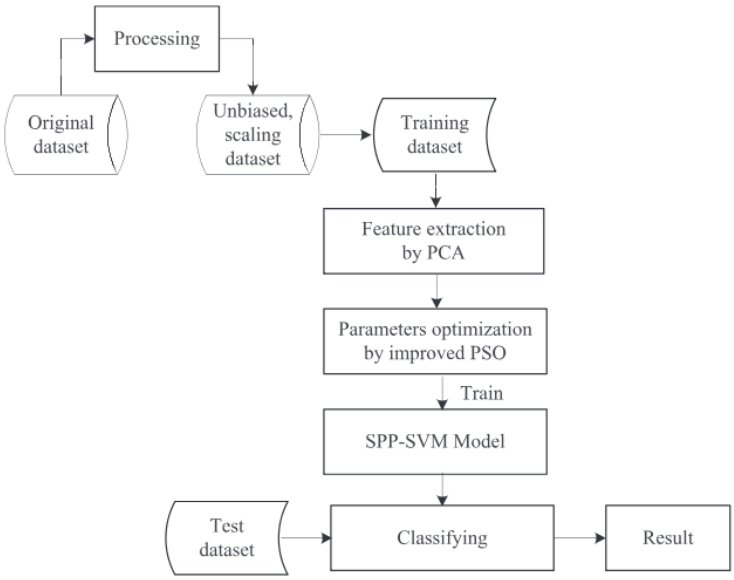

Figura 3.1. Flusso di funzionamento della SVM migliorata (articolo onlinelibrary.wiley.com).

Leggendo invece un articolo [7] sul riconoscimento di traffico cifrato scopriamo che andando a studiare la dimensione dei primi pacchetti di una connessione possiamo rilevare il tipo di traffico sottostante con una precisione dell'85%. Viene assunto che SSL non modifichi significativamente il numero dei pacchetti, la loro dimensione e il loro tempo di arrivo. Le connessioni TCP possono essere classificate in base a durata, numero dei pacchetti e tempo medio di arrivo. Per verificare il fatto che un server utilizzi SSL viene detto semplicemente di analizzare il secondo pacchetto, che è il Server Hello, che mi dice se il server ha accettato di usare SSL e con quali algoritmi. Quindi, dalla dimensione dei primi 3 pacchetti della fase di

handshake riesco a capire se il traffico è SSL oppure no. Se lo è guardo la dimensione dei primi 3 pacchetti cifrati e con un classificatore capisco di quale applicativo si tratta. La dimensione dei pacchetti ci può dare informazioni in questo senso, ma dobbiamo tenere a mente che i differenti algoritmi di cifratura modificano la dimensione del pacchetto in maniera diversa, anche se questo cambio di dimensione è fisso e predicibile. Quindi dall'algoritmo concordato in fase di handshake risaliamo alla dimensione originale del pacchetto. Per velocizzare, se l'algoritmo di cifratura usato è uno tra i primi 5 più utilizzati, possiamo sottrarre 21 alla dimensione per trovare quella del pacchetto originale. Attenzione agli eventuali dati compressi o ai segmenti vuoti perché potremmo generare risultati errati. Come vedremo, la maggior parte (se non tutto) il traffico cifrato catturato viene appunto cifrato utilizzando AES come algoritmo di cifratura simmetrica, e quindi andare a sottrarre oppure fare altre operazioni per ottenere la dimensione originale del pacchetto è del tutto inutile, dal momento che AES appartiene anche alla schiera dei primi 5 algoritmi più utilizzati, e quindi il dato sulla dimensione, e tutto quello che faremo derivare da esso, lo prenderemo così come ci arriva. Per completezza viene riportata la Figura 3.2 [7] che spiega la relazione che intercorre sulla dimensione dei pacchetti cifrati e non, in base alla cipher suite utilizzata.

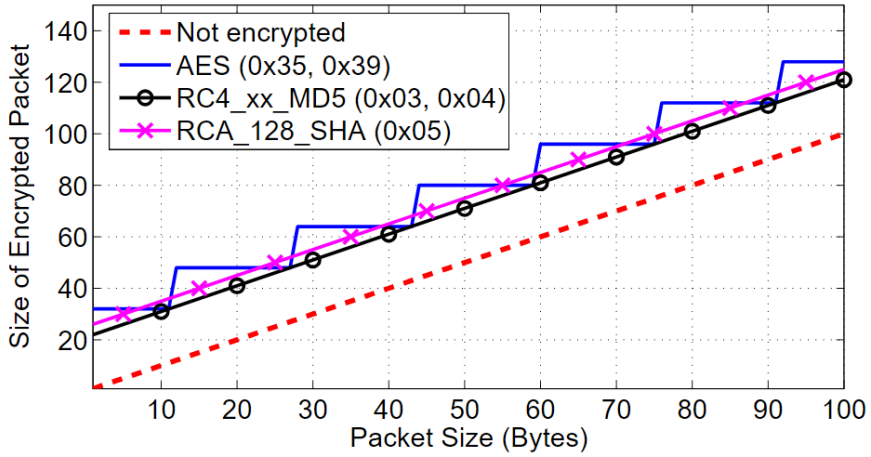

Figura 3.2. Rapporto cifratura - dimensione pacchetto (articolo link.springer.com).

Un articolo che riguarda il WENC [8], che sta per Weighhted ENsemble Classifier, il quale studia il comportamento di alcuni sotto flussi durante la fase di handshake e durante lo scambio di dati vero e proprio, afferma che esso raggiunge un'accuratezza più alta rispetto alla media. Afferma anche che il traffico HTTPS è circa il 30% del traffico totale. Basarsi sulla lunghezza del pacchetto aumenta l'accuratezza, ma anche qui ci sono dei casi in cui non funziona molto bene. WENC cerca di fronteggiare queste limitazioni utilizzando catene di Markov del secondo ordine. Le caratteristiche di tipo di messaggio e lunghezza di pacchetto nella fase di handshake vengono unite per stabilire un modello a catena di Markov di primo e secondo ordine. Tutto ciò serve per classificare il traffico SSL/TLS ma può fallire perché per prima cosa questi modelli considerano solo due stati discreti di transizione che non descrivono in toto le caratteristiche distinguibili di applicazioni HTTPS differenti, e come seconda cosa il modello di stato basato su una singola

variabile fornisce espressività limitata. Per far fronte a questi problemi usiamo un modello di catena di Markov di secondo livello in cui lo stato corrente dipende non solo da quello precedente, e questo serve per risolvere la limitata espressività. Oltre a ciò occorre adottare un Hidden Markov Model (HMM). WENC è però lento perché riunisce i risultati di due classificatori , ma la computazione parallela può essere usata per migliorare la sua efficienza anche se la sua valutazione mostra che WENC ottiene prestazioni di classificazione migliori, superiore a diversi altri approcci alternativi. La Figura 3.3 riassume il modello di funzionamento appena esposto.

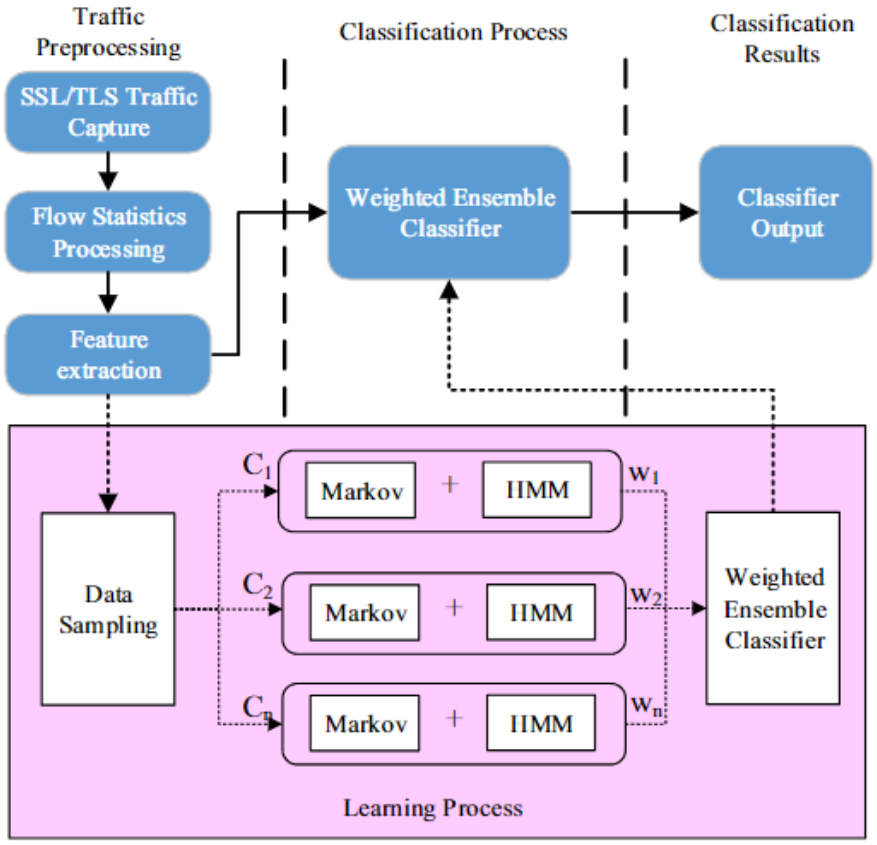

Figura 3.3. Architettura del sistema WENC (articolo ieeexplore.ieee.org).

Un articolo che parla di un algoritmo detto SPID [9] (Statistical Protocol IDentification) spiega che esso identifica il protocollo di livello applicativo usando misure statistiche del flusso dati. Questo algoritmo utilizza la divergenza di Kullback-Leibler (misura non simmetrica della differenza tra due distribuzioni di probabilità P e Q) per confrontare la probabilità dei vettori creati dal traffico di rete osservato con la probabilità dei vettori di protocolli noti, e permette di piazzare il traffico in un certo tipo di protocollo solo se viene anche impostata anche una certa soglia. SPID si appoggia anche a un Network Intrusion Detection System (NIDS), che ha il compito di analizzare i dati del livello applicativo alla ricerca di traffico di rete illecito. Molti NIDS riconoscono il protocollo di livello applicativo utilizzando una tabella di well-known ports assegnate dallo IANA, ma solo il 50-70% del traffico è classificabile in questo modo perché alcuni applicativi utilizzano numeri di porte

non standard. Spesso le backdoor sono mascherate da queste porte non standard. I punti chiave di SPID sono: un piccolo database dei protocolli, minima complessità, il fatto di identificare subito il protocollo in una sessione e l'identificazione affidabile e accurata del protocollo. SPID si può utilizzare nei sistemi real time e embedded, che hanno limitata memoria e capacità di processamento. SPID si basa sui primi 4 pacchetti (senza contare i pacchetti di segnalazione) di una sessione. Viene quindi cercata la miglior corrispondenza tra il modello di un protocollo sconosciuto e quello di protocolli conosciuti. SPID richiede comunque training data già classificati. Gli algoritmi per la classificazione del traffico sono nella maggior parte dei casi basati su Naive Bayes e Hidden Markov Model. SPID può essere usato per identificare i protocolli in qualunque schema di comunicazione bidirezionale determinata da 5 tuple per identificare un flusso (IP sorgente, porta sorgente, IP destinazione, porta destinazione e protocollo di livello trasporto). La complessità di SPID dipende dunque dal numero di protocolli presenti nel database. SPID deve essere regolato, messo a punto in alcune parti come la lunghezza dei vettori e la soglia di divergenza.

Un ultimo algoritmo o modus operandi che ho preso in considerazione è stato BLINC [10]. Questo approccio sposta l'attenzione dai flussi individuali all'associazione degli host con le applicazioni, per poi classificare il loro flusso. Questo nuovo modo di vedere le cose è dovuto al fatto che osservare l'attività di un host può fornire più informazioni e può rivelare la natura dell'applicazione dell'host. Una caratteristica importante di questo algoritmo è che può fornire e ricavare informazioni a vari livelli di dettaglio e i criteri di classificazione sono controllati da soglie che possono essere rilassate o strette in base ai requisiti della nostra ricerca. In seconda battuta questo metodo tenta di catturare il comportamento di un host a tre livelli differenti: sociale, di rete e applicativo. I tre livelli differenti sono così descritti:

Livello sociale in cui viene catturato il comportamento di un host in base a con quali e con quanti host comunica, soprattutto se quelli con cui comunica sono membri di un certo applicativo. Per questi motivi, a questo livello abbiamo solo bisogno di accedere all'indirizzo IP sorgente e destinazione. Nel livello sociale ci sono alcune caratteristiche che ci possono suggerire qualcosa. Ad esempio il fatto che un host interagisca con un largo numero di hosts in un breve periodo di tempo è indice di un possibile attacco malware oppure un traffico peer to peer. Si può anche proporre un modello formato da due insiemi, in cui uno è formato dagli IP sorgente, mentre il secondo dagli IP destinazione, e poi studiarne le interconnessioni tra di essi. Se si formano degli insiemi perfettamente bipartiti è possibile che si tratti di traffico malevolo, mentre se si verifica una sovrapposizione parziale il traffico potrebbe interessare il gaming perché ci sono tanti client che si collegano allo stesso service nello stesso periodo di tempo, oppure si può trattare di traffico peer to peer. Invece se la sovrapposizione è parziale all'interno dello stesso dominio è perché probabilmente ci troviamo di fronte a delle service farm che forniscono servizi ai clienti e quindi ci può essere traffico web, mail, streaming oppure dns ad esempio. Identificare un server all'interno di un dominio, e quindi il servizio applicativo che offre, ci permette di classificare quindi gli altri indirizzi IP client che usufruiscono del servizio e quindi il loro applicativo più utilizzato;

Livello rete o funzionale in cui cerchiamo di capire il comportamento dell'host in base al suo ruolo funzionale nella rete, cioè se funziona come fornitore o consumatore di un servizio, oppure entrambe, in caso di applicazione collaborativa. Per esempio, gli hosts che usano una singola porta per la maggior parte delle loro interazioni con gli altri hosts sono spesso dei fornitori del servizio offerto su quella porta. Per questi motivi a questo livello dobbiamo prendere in considerazione IP sorgente e destinazione e porta sorgente. Al livello di rete oppure funzionale è difficile classificare un host che partecipi a solo uno o pochi flussi. Se un host è un server che fornisce un servizio ed esaminiamo i flussi in cui lui rappresenta la sorgente allora noteremo molto probabilmente che userà sempre la stessa porta sorgente, mentre se questo host si comporta come client verso molti server allora la sua porta sorgente varierà molto per i vari flussi. In un flusso peer to peer non riusciamo a discriminare server e client. Dunque se un host utilizza un numero piccolo di porte sorgente, generalmente minore o uguale a 2 per ogni flusso, allora questo host sta probabilmente fornendo un servizio, anzi se utilizza solo una porta come sorgente allora può essere un mail o un web server ad esempio. In caso contrario agirà da richiedente di un servizio, cioè da client, oppure sta partecipando a traffico peer to peer;

Livello applicativo in cui catturiamo le interazioni a livello trasporto tra hosts particolari su porte specifiche con l'intento di identificare l'applicazione di origine. Viene proposta una classificazione basandosi su 4 caratteristiche: indirizzo sorgente e destinazione, porta sorgente e destinazione. Questo modello viene poi arricchito aggiungendo informazioni quali il protocollo di livello trasporto usato, la dimensione media dei pacchetti scambiati, il numero di pacchetti o di byte trasferiti. A livello applicativo combiniamo la conoscenza dei livelli precedenti con le interazioni a livello trasporto tra gli hosts per identificare l'applicativo di origine. Vengono prodotti dei piccoli grafici chiamati graphlet in base a tutte queste informazioni come classi di riferimento, e poi viene confrontato il dataset con ognuno di questi grafici per trovare quello a cui somiglia di più. In dettaglio, ogni grafico ha 4 colonne corrispondenti agli attributi menzionati prima (IP sorgente e destinazione, porta sorgente e destinazione), mentre in alcuni ce ne sono 5 perché viene aggiunta l'informazione sul protocollo di livello trasporto utilizzato, come viene rappresentato nella Figura 3.4 di esempio. Non bisogna quindi basarsi solo sul numero di porta per identificare il protocollo sottostante ma a un insieme di informazioni. Nel traffico web o gaming solitamente una singola sorgente IP comunica con molti destinatari usando la stessa porta sorgente e diverse porte di destinazione, quindi occorre in questi casi un'analisi più approfondita, come ad esempio la cardinalità delle porte di destinazione contro gli IP di destinazione. In ftp l'host sorgente fornisce il servizio a due specifici numeri di porta (canale di controllo e di dati), mentre lo streaming utilizza specifici numeri di porta sia in sorgente che in destinazione, cosi come fanno i mail server.

Sono state anche raccolte informazioni utili in caso di parità o non completa sicurezza nell'identificazione del protocollo: sappiamo ad esempio che il traffico mail si svolge solo sopra TCP. Il traffico peer to peer invece può usare sia TCP che UDP.

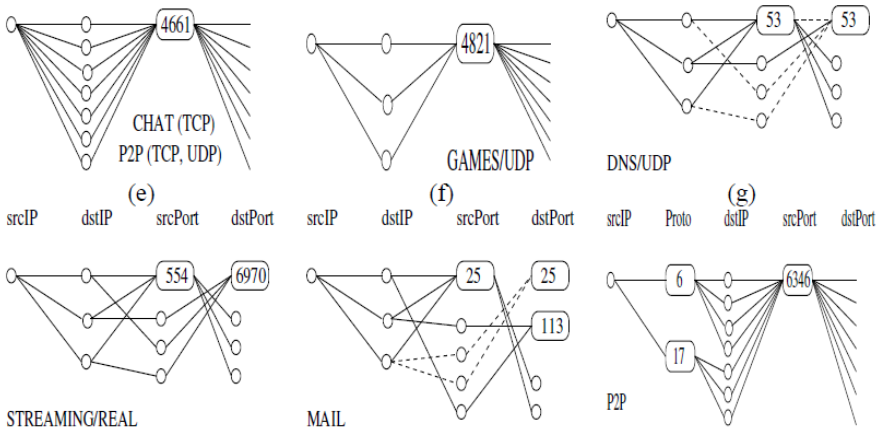

Figura 3.4. Esempi di grafici di alcune classi di traffico (articolo dl.acm.org).

Gaming, malware e mail sono anche caratterizzati da una serie di pacchetti di dimensione costante, calcolato come la media della dimensione di pacchetto per flussi che devono essere costanti, cioè come byte totali fratto numero di pacchetti. Mail e dns server comunicano con altri server dello stesso tipo e utilizzano la stessa porta come sorgente e come destinazione. Flussi falliti, cioè che si basano su un protocollo che non abbiamo analizzato o senza payload, identificano spesso attacchi o reti peer-to-peer. BLINC permette anche di identificare le porte su cui ci può essere un attacco in corso, mostrandoci quelle su cui passa molto traffico diretto verso altri indirizzi IP, cioè cerca di trovare porte di destinazione con un'alta attività di flusso fallito.

Dando un'occhiata ad alcuni altri articoli e ricerche [11] [12] [13] [14] [15] [16] ci si rende conto di alcune altre note che qui riporto:

Well-known ports (intervallo 0-1023) di alcuni protocolli di livello applicativo che utilizzano la cifratura, e insieme ad esse anche quelle di porte registrate (intervallo 1024-49151). In base alla porta su cui mando i pacchetti, supponendo che io stia utilizzando un canale cifrato, posso determinare quale protocollo di livello 7 ci sia sotto. Il problema è che spesso i possibili attaccanti non utilizzano le porte standard, oppure la porta standard di un protocollo può essere cambiata in corsa, dinamicamente, e quindi tutto ciò ci fa pensare a questa soluzione come non universalmente utile e accettata. Il numero di porta destinazione è comunque stato preso in considerazione nel mio esperimento di tesi poiché è stato valutato come parametro significativo;

Fingerprint o impronta digitale. Essa è una sequenza alfanumerica o una stringa di bit di lunghezza prefissata che identifica un certo file con le sue caratteristiche intrinseche, e viene utilizzata per identificare rapidamente i file. Questo tipo di informazione si può ricavare andando a vedere cosa contiene il pacchetto nel campo Message type (o anche Content type). Infatti in tale campo è presente un numero intero durante la connessione SSL che identifica se siamo in fase di handshake, e quindi può essere un pacchetto Client Hello oppure Server Hello, oppure in fase di scambio di dati applicativi, ma non ci dice che tipo di applicativo ci sia sotto, oppure può anche essere un pacchetto di Alert,

quello che permette di terminare la connessione tra le due parti in maniera ordinata. Questo tipo di approccio per classificare il traffico non è sufficiente ed è per questo che viene potenziato utilizzando le catene di Markov del secondo ordine e nascoste per poter classificare soprattutto il traffico HTTPS. In questo modello l'informazione del tipo di pacchetto unito alla sua lunghezza può darci informazioni più accurate sul tipo di traffico trasportato. I valori più utilizzati di Message type sono: 20 (Change cipher spec), 21 (Alert), 22 (Handshake), 23 (Application), 24 (Heartbeat). Questo parametro non è stato preso in considerazione nel mio esperimento di tesi poiché è stato valutato come non significativo;

Server Name Indication è un'estensione al protocollo TLS, tramite cui un client indica quale è il nome dell'host a cui sta cercando di connettersi all'inizio del processo di handshake. Tale informazione non è cifrata, e quindi ha lo svantaggio di rendere noto a chiunque intercetti il traffico di un client di sapere a quale server si vuole collegare. Esso permette di servire molti client da uno stesso indirizzo IP senza richiedere che tutti questi servizi utilizzino lo stesso certificato, cioè permette ai server di utilizzare più di un certificato per lo stesso IP e porta. Occorre quindi trovare il modo per legare il nome del server o del client che viene ricavato da questa ricerca a un tipo di servizio o a un tipo di applicativo. A volte come Common Name o Server Name ci viene dato un indirizzo IP invece di un nome, ma partendo da un indirizzo IP si può sapere l'hostname da terminale Linux tramite un comando, se disponibile. Questo parametro non è stato preso in considerazione nel mio esperimento di tesi poiché il nome di un server non è così discriminante per il tipo di traffico e di servizio erogato, come sarà discusso nei capitoli successivi;

Altri parametri statistici, come ad esempio il massimo, minimo, medio e la deviazione standard del tempo di inter arrivo tra i pacchetti, e molte altre informazioni statistiche che verranno discusse nei prossimi capitoli, in cui saranno anche definite le classi di traffico che sono state prese in considerazione.

Capitolo 4

Progettazione della soluzione

4.1 File di log di interesse

I file di log nativi di Bro che sono interessanti per la nostra classificazione sono conn.log, ssl.log e ssh.log, poiché ci forniscono tutti i parametri di cui abbiamo bisogno per poter operare una classificazione. Tutti i parametri in più che abbiamo aggiunto derivano anch'essi da questi file di log, e quindi tramite una loro combinazione aritmetica si possono ottenere gli attributi che sono stati aggiunti e che sono stati giudicati essere utili alla causa. Vediamo ora nelle seguenti figure tratte da un documento riguardante la documentazione di Bro il contenuto, il tipo e la descrizione dei parametri contenuti nei file di log di maggior interesse. Nella tabella 4.1 è mostrato il contenuto del file conn.log, quello da cui vengono rimediati più parametri, la tabella 4.2 invece mostra il contenuto del file ssl.log, mentre la tabella 4.3 quella del file ssh.log, e questi ultimi vengono generati quando Bro rileva del traffico cifrato. Successivamente saranno elencati i parametri utilizzati per la classificazione, così si potrà capire almeno da dove vengono e che tipo di attributi sono quelli selezionati. Non è stato utilizzato il contenuto del file X509.log, in cui viene aggiunta una entry ogni volta che Bro trova un certificato nell'analisi di rete o di un file .pcap. Infatti è stato notato che non per tutte le connessione è disponibile una entry in questo file di log perché se stiamo catturando una connessione che utilizza un identificativo di sessione già aperta in precedenza allora probabilmente il certificato non viene nuovamente rimandato, come è stato spiegato nel capitolo 2 in cui si spiegava il protocollo SSL/TLS, e quindi non è possibile recuperare tutte le informazioni che ci servirebbero.

4.2 Identificazione parametri di classificazione

Dalla lettura degli articoli che sono già stati prodotti sulla classificazione di traffico cifrato ci sono alcuni parametri, o attributi, che sono stati spesso usati per poter appunto operare una classificazione di traffico non in chiaro. Alcuni di essi sono

Campo	*Tipo*	*Descrizione*
ts	time	timestamp del primo pacchetto
uid	string	id univoco della connessione
id.orig_h	addr	indirizzo ip sorgente
id.orig_p	port	numero porta sorgente
id.resp_h	addr	indirizzo ip destinazione
id.resp_p	port	numero porta destinazione
proto	proto	protocollo di livello trasporto
service	string	protocollo di livello applicativo
duration	interval	durata della connessione
orig_bytes	count	numero bytes inviati dal mittente
resp_bytes	count	numero bytes inviati dal destinatario
orig_pkts	count	numero pacchetti inviati dal mittente
resp_pkts	count	numero pacchetti inviati dal destinatario

Tabella 4.1. Tabella contenente campo, tipo e descrizione di alcuni parametri di conn.log.

Campo	*Tipo*	*Descrizione*
ts	time	timestamp quando SSL rilevato
uid	string	id univoco della connessione
version	string	versione SSL offerta dal server
cipher	string	ciphersuite scelta dal server
server_name	string	nome del server
session_id	string	identificativo della sessione
subject	string	soggetto del certificato offerto dal server
issuer	string	firmatario del certificato del server
client_subject	string	soggetto del certificato emesso dal client
client_issuer	string	firmatario del certificato del client

Tabella 4.2. Tabella contenente campo, tipo e descrizione di alcuni parametri di ssl.log.

stati scartati perché valutati non essere utili in questo senso, mentre alti sono stati mantenuti e potenziati in qualche modo, con altri attributi a loro collegati. Come detto nel precedentemente, ne sono poi stati aggiunti di nuovi perché a differenza di altri sono stati valutati essere utili alla causa, e anche i test poi condotti durante la fase finale dell'esperimento hanno dato i loro frutti fornendo un'approvazione in tal senso, e quindi sono stati giustamente mantenuti.

Di seguito sarà fornito un elenco, che potrà poi essere giustamente esteso nel caso verranno trovati nuovi parametri che possano far crescere le prestazioni degli algoritmi di machine learning applicati alla classificazione del traffico. Tra parentesi si trova il nome con cui sono stati salvati questi parametri nel file di uscita che vedremo nel prossimo capitolo, e che raccoglie tutte le informazioni dopo aver processato tutti i file .pcap raccolti. Le parole entranti e uscenti si riferiscono sempre all'host che ha iniziato o richiesto una connessione:

Campo	*Tipo*	*Descrizione*
ts	time	timestamp quando SSH rilevato
uid	string	id univoco della connessione
version	count	versione SSH utilizzata
client	string	stringa che identifica il client
server_name	string	stringa che identifica il server
cipher_alg	string	algoritmo per la cifratura
mac_alg	string	algoritmo per il calcolo MAC
compression_alg	string	algoritmo per la compressione
kex_alg	string	algoritmo per lo scambio chiavi

Tabella 4.3. Tabella contenente campo, tipo e descrizione di alcuni parametri di ssh.log.

porta destinazione (port_dest), è un parametro che permette di capire verso che porta è stata fatta la richiesta di connessione. Per alcune classi ha un valore standard, come avremo modo di vedere. Questo è un parametro che Bro ci fornisce nel file conn.log sotto il nome di id.resp_p;

durata (duration), intesa come durata della connessione, cioè come intervallo di tempo trascorso tra l'inizio e la fine della connessione, o comunque fino al momento in cui la cattura del traffico appartenente a questa connessione è stata interrotta. In questo lasso di tempo è compresa anche la fase di handshake del protocollo SSL/TLS, e quindi i messaggi di Client Hello, Server Hello, ecc. Questo è un parametro che Bro ci fornisce nel file conn.log sotto il nome di duration;

numero di pacchetti in uscita (forward_pkts_num), contiene l'informazione riguardo il numero totale di pacchetti in uscita in una connessione. Questo è un parametro che Bro ci fornisce nel file conn.log sotto il nome di orig_pkts;

numero di pacchetti in entrata (backward_pkts_num), contiene l'informazione riguardo il numero totale di pacchetti in entrata in una connessione. Questo è un parametro che Bro ci fornisce nel file conn.log sotto il nome di resp_pkts;

percentuale di pacchetti in uscita (forward_pkts_perc), contiene la percentuale di quanti pacchetti escono sul totale. Questa informazione non ce la fornisce direttamente Bro, ma è possibile ricavarla facendo la divisione tra il numero di pacchetti che escono e il numero totale di pacchetti in una connessione, ed entrambi i parametri possono essere trovati nel file conn.log;

percentuale di pacchetti in entrata (backward_pkts_perc), contiene la percentuale di quanti pacchetti entrano sul totale. Questa informazione non ce la fornisce direttamente Bro, ma è possibile ricavarla facendo la divisione tra il numero di pacchetti che entrano e il numero totale di pacchetti in una connessione, ed entrambi i parametri possono essere trovati nel file conn.log;

numero di pacchetti totali (tot_pkts), contiene l'informazione del numero totale di pacchetti entranti e uscenti in una connessione. Bro non ce la fornisce direttamente ma la possiamo ricavare andando a sommare il numero di pacchetti entranti e il numero di pacchetti uscenti, ed entrambe queste informazioni possono essere trovate nel file conn.log;

dimensione media di pacchetto (avg_dim_pkt), contiene l'informazione sulla dimensione media di un pacchetto, e questa informazione non ce la fornisce direttamente Bro, ma ce la dobbiamo calcolare utilizzando uno script che tiene in memoria le dimensioni di tutti i pacchetti ricevuti e mandati, mentre conta anche il numero totale di pacchetti per poi andare a dividere tra di loro queste due grandezze;

dimensione media del pacchetto in uscita (forward_avg_dim_pkt), dove anche questa informazione ce la dobbiamo andare a calcolare utilizzando uno script che fa la stessa cosa della media ma conta solo le dimensioni dei pacchetti uscenti e il numero di pacchetti uscenti, perché Bro purtroppo non ci fornisce questa misura direttamente;

dimensione media del pacchetto in entrata (backward_avg_dim_pkt), dove anche questa informazione ce la dobbiamo andare a calcolare utilizzando uno script che fa la stessa cosa della media ma conta solo le dimensioni dei pacchetti entranti e il numero di pacchetti entranti, perché Bro purtroppo non ci fornisce questa misura direttamente;

massima dimensione del pacchetto (maximum_dim_pkt), che è la misura di quanti byte al massimo contiene un pacchetto di una connessione, e questa informazione, non fornendocela direttamente Bro, ce la dobbiamo ricavare vagliando tutti i pacchetti e cercando quello di dimensione massima. Questa caratteristica sarà ricavata facendo ricorso a uno script creato appositamente, che sarà incluso nel motore di Bro;

minima dimensione del pacchetto (minimum_dim_pkt), che è la misura di quanti byte come minimo contiene un pacchetto di una connessione, e questa informazione, non fornendocela direttamente Bro, ce la dobbiamo ricavare vagliando tutti i pacchetti e cercando quello di dimensione minima. Questa caratteristica sarà ricavata facendo ricorso a uno script creato appositamente, che sarà incluso nel motore di Bro;

deviazione standard della dimensione di un pacchetto (std_dev_dim_pkt), la cui informazione non ce la fornisce direttamente Bro, ma useremo uno script apposta che ce la calcoli sapendo la dimensione di ogni pacchetto, la media e il numero di pacchetti totali, e verrà utilizzata la formula per la deviazione standard:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)}{N}},$$

in cui x_i rappresenta la dimensione di un pacchetto, N il numero totale di pacchetti e μ la media;

numero di byte in uscita (forward_bytes_num), contiene l'informazione riguardo il numero totale di byte in uscita in una connessione. Questo è un parametro che Bro ci fornisce nel file conn.log sotto il nome di orig_bytes;

numero di byte in entrata (backward_bytes_num), contiene l'informazione riguardo il numero totale di byte in entrata in una connessione. Questo è un parametro che Bro ci fornisce nel file conn.log sotto il nome di resp_bytes;

percentuale di byte in uscita (forward_bytes_perc), contiene la percentuale di quanti byte escono sul totale. Questa informazione non ce la fornisce direttamente Bro, ma è possibile ricavarla facendo la divisione tra il numero di byte che escono e il numero totale di byte in una connessione, ed entrambi i parametri possono essere trovati nel file conn.log;

percentuale di byte in entrata (backward_bytes_perc), contiene la percentuale di quanti byte entrano sul totale. Questa informazione non ce la fornisce direttamente Bro, ma è possibile ricavarla facendo la divisione tra il numero di byte che entrano e il numero totale di byte in una connessione, ed entrambi i parametri possono essere trovati nel file conn.log;

numero di bytes totali (tot_bytes), contiene l'informazione del numero totale di byte entranti e uscenti in una connessione. Bro non ce la fornisce direttamente ma la possiamo ricavare andando a sommare il numero di byte entranti e il numero di byte uscenti, ed entrambe queste informazioni possono essere trovate nel file conn.log;

throughput totale di pacchetti (tot_throughput_pkts), contiene l'informazione del numero di pacchetti al secondo che sono transitati tra le due controparti. Questa informazione non ci viene fornita direttamente da Bro ma può essere ricavata sommando pacchetti entranti con pacchetti uscenti e dividendo per la durata della connessione, così da avere come unità di misura il numero di pacchetti al secondo che sono transitati;

throughput in uscita dei pacchetti (forward_throughput_pkts), contiene l'informazione del numero di pacchetti al secondo che sono usciti dall'host che ha iniziato la connessione. Questa informazione non ci viene fornita direttamente da Bro ma può essere ricavata dividendo il numero di pacchetti uscenti per la durata della connessione, così da avere anche qui come unità di misura il numero di pacchetti al secondo;

throughput in entrata dei pacchetti (backward_throughput_pkts), contiene l'informazione del numero di pacchetti al secondo che sono entrati verso l'host che ha iniziato la connessione. Questa informazione non ci viene fornita direttamente da Bro ma può essere ricavata dividendo il numero di pacchetti entranti per la durata della connessione, così da avere anche qui come unità di misura il numero di pacchetti al secondo;

throughput totale di byte (tot_throughput_bytes), contiene l'informazione del numero di byte al secondo che sono transitati tra le due controparti. Questa informazione non ci viene fornita direttamente da Bro ma può essere ricavata

sommando byte entranti con byte uscenti e dividendo per la durata della connessione, così da avere come unità di misura il numero di byte al secondo;

throughput in uscita dei byte (forward_throughput_bytes), contiene l'informazione del numero di byte al secondo che sono usciti dall'host che ha iniziato la connessione. Questa informazione non ci viene fornita direttamente da Bro ma può essere ricavata dividendo il numero di byte uscenti per la durata della connessione, così da avere anche qui come unità di misura il numero di byte al secondo;

throughput in entrata dei byte (backward_throughput_bytes), contiene l'informazione del numero di byte al secondo che sono entrati verso l'host che ha iniziato la connessione. Questa informazione non ci viene fornita direttamente da Bro ma può essere ricavata dividendo il numero di byte entranti per la durata della connessione, così da avere anche qui come unità di misura il numero di byte al secondo;

massimo inter arrival packet time (maximum_inter_pkt), è il massimo tempo intercorso tra l'arrivo di due pacchetti consecutivi in una connessione, e questa informazione non può essere ricavata in alcun modo da Bro se non appoggiandosi a uno script che, come vedremo, tiene conto di tutti i tempi di arrivo dei pacchetti in una connessione, per poi andare a trovarne il massimo;

minimo inter arrival packet time (minimum_inter_pkt), è il minimo tempo intercorso tra l'arrivo di due pacchetti consecutivi in una connessione, e questa informazione non può essere ricavata in alcun modo da Bro se non appoggiandosi a uno script che, come vedremo, tiene conto di tutti i tempi di arrivo dei pacchetti in una connessione, per poi andare a trovarne il minimo;

inter arrival packet time medio (avg_inter_pkt), è la media tra tutti i tempi intercorsi tra l'arrivo di tutti i pacchetti consecutivi in una connessione, e questa informazione non può essere ricavata in alcun modo da Bro se non appoggiandosi a uno script che, come vedremo, tiene conto di tutti i tempi di arrivo dei pacchetti in una connessione, per poi andare a farne la media sommando i tempi e dividendo per il numero totale di campioni considerati;

deviazione standard di inter arrival packet time (std_dev_inter_pkt), è la deviazione standard tra tutti i tempi intercorsi tra l'arrivo di tutti i pacchetti consecutivi in una connessione, e questa informazione non può essere ricavata in alcun modo da Bro se non appoggiandosi a uno script che, come vedremo, tiene conto di tutti i tempi di arrivo dei pacchetti in una connessione, per poi andare a calcolarne la deviazione standard con la formula vista prima:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)}{N}},$$

in cui x_i rappresenta il tempo di inter arrivo tra due pacchetti consecutivi, N il numero totale di tempi di inter arrivo e μ la media;

algoritmo di cifratura asimmetrico (asymmetric), o di scambio chiavi, utilizzato nella connessione cifrata tra le due controparti. Questa informazione può

essere ricavata da Bro per quanto riguarda il traffico cifrato in generale nel file ssl.log sotto il nome di parametro cipher, che contiene tutta la cipher suite, e poi andando ad isolare solamente la parte asimmetrica di interesse. Per quanto riguarda il traffico ssh deve essere ricavata dal parametro host_key_alg di ssh.log;

algoritmo di cifratura simmetrico (symmetric), o di cifratura, utilizzato nella connessione cifrata tra le due controparti. Questa informazione può essere ricavata da Bro per quanto riguarda il traffico cifrato in generale nel file ssl.log sotto il nome di parametro cipher, che contiene tutta la cipher suite, e poi andando ad isolare solamente la parte simmetrica di interesse. Per quanto riguarda il traffico ssh deve essere ricavata dal parametro cipher_alg di ssh.log;

algoritmo a blocchi utilizzato per cifrare (blocks), utilizzato nella connessione cifrata tra le due controparti. Questa informazione può essere ricavata da Bro per quanto riguarda il traffico cifrato in generale nel file ssl.log sotto il nome di parametro cipher, che contiene tutta la cipher suite, e poi andando ad isolare solamente la parte a blocchi di interesse. Per quanto riguarda il traffico ssh invece questa informazione non viene menzionata e quindi viene impostato il valore none;

algoritmo di digest (digest), utilizzato nella connessione cifrata tra le due controparti. Questa informazione può essere ricavata da Bro per quanto riguarda il traffico cifrato in generale nel file ssl.log sotto il nome di parametro cipher, che contiene tutta la cipher suite, e poi andando ad isolare solamente la parte di digest di interesse. Per quanto riguarda il traffico ssh deve essere ricavata dal parametro mac_alg di ssh.log.

4.2.1 Parametri in eccesso

Come si vedrà nel capitolo successivo sono stati utilizzati alcuni parametri in più negli script Bro, rispetto a quelli che sono appena stati menzionati sopra. Alcuni di essi, come la connection id, servivano per fare il merge delle tabelle dei file di log, mentre altri si pensava fossero utili perché erano come un header della connessione, ma sono poi stati eliminati nel merge e nelle valutazioni finali in quanto non influenzavano l'algoritmo di classificazione. Questi parametri sono:

uid, contenuto in conn.bro, che rappresenta l'identificativo di connessione, è un numero random ma quando ci sono più file di log con un parametro come questo esso assume lo stesso valore per la stessa connessione, e questo comportamento ci permette così di unire i dati di più tabelle. Se ad esempio ho una connessione identificata da una certa stringa in conn.log, e in quella riga vengono descritte le sue peculiarità principali (numero di pacchetti, byte...) tra cui il fatto che il servizio che porta è http, allora avrò nel file http.log una stringa con uguale valore che descrive la connessione dal punto di vista del protocollo http (valore dello user agent, l'uri utilizzata nella richiesta...);

id.orig_h, contenuto in conn.bro, che rappresenta l'indirizzo IP sorgente della connessione;

id.resp_h, contenuto in conn.bro, che rappresenta l'indirizzo IP destinazione della connessione;

id.orig_p, contenuto in conn.bro, che rappresenta la porta sorgente della connessione;

proto, contenuto in conn.bro, che rappresenta il protocollo di livello trasporto della connessione. Alcune ricerche indicavano che questa informazione unita all'IP-sorgente, all'IP-destinazione, alla porta-sorgente e alla porta-destinazione ci poteva dare qualche informazione riguardo la classificazione, soprattutto nel caso fosse già passato del traffico con valori uguali per questi 5 parametri. Purtroppo però che nel nostro caso possono insorgere delle complicazioni perché, come vedremo più avanti, il traffico che abbiamo registrato è stato catturato sotto certe condizioni per cui l'indirizzo sorgente, quello destinazione e la porta sorgente variano senza nessun tipo di correlazione rispetto al tipo di traffico, ma solo a causa di eventi di sistema;

server_name, contenuto in ssl.log, contiene l'informazione del nome del server con cui si sta comunicando, e spesso questa informazione non è molto significativa. Un parametro molto simile a questo è il campo subject, contenuto anch'esso in ssl.log, che ci indica il soggetto del certificato X.509 offerto dal server. Entrambe queste informazioni sono menzionate nelle ricerche lette, ma non sono state giudicate essere di interesse poiché ad esempio un server name come google.com, ma non solo, può offrire diversi servizi che possono quindi appartenere a diverse classi di traffico. Quindi se ad esempio addestrassi un classificatore ad associare un certo nome a una certa classe di traffico, e se poi mi arrivasse un traffico di tutt'altro tipo sotto quel nome ecco che il mio classificatore sbaglierebbe di sicuro;

server, contenuto in ssh.log, che identifica la controparte coinvolta nella connessione cifrata, cioè il nome dell'entità a cui la nostra macchina ha fatto richiesta, e per la quale valgono le considerazioni appena fatte nel punto precedente.

Erano stati aggiunti anche altri parametri legati al numero di byte a livello IP uscenti ed entranti, come il totale, quelli entranti sul totale e quelli uscenti sul totale, il numero di byte entranti e uscenti. Però dato che abbiamo già l'informazione dei byte a livello del payload e poiché le due tipologie di dati sono molto vicine quantitativamente parlando è stato deciso di mantenere solo quelli a livello del payload, essendo anche a più alto livello.

4.2.2 Applicabilità in Aramis

Se si volesse applicare il lavoro fatto all'interno dell'azienda Aizoon allora si dovrebbero rivedere alcune cose, poiché non tutti gli script che sono stati aggiunti, come vedremo, possono essere eseguiti su grosse quantità di traffico a causa di vincoli di tempo, poiché ci metterebbero troppo ad essere eseguiti a causa dei calcoli da fare in tempo reale, e poi anche per vincoli di spazio poiché ad esempio bisognerebbe salvare in memoria la dimensione di tutti i pacchetti di tutte le connessioni e i

tempi di inter arrivo di tutti i pacchetti consecutivi di tutte le connessioni! Quindi alcuni parametri sarebbero da sostituire con altri, e diventerebbero solo delle stime (anche se si avvicinano molto ai valori reali, come è stato constatato in fase di esperimento), mentre altri ancora sarebbero addirittura da eliminare. Nel dettaglio:

la dimensione media del pacchetto sarebbe da sostituire con una stima, in cui si divide il numero di byte totali per il numero di pacchetti totali, ed entrambe le grandezze sono disponibili all'interno di conn.log, senza quindi dover contare pacchetto per pacchetto;

la dimensione media dei pacchetti entranti sarebbe da sostituire con il numero totale di byte entranti diviso il numero totale di pacchetti entranti, ed entrambe le grandezze sono disponibili all'interno di conn.log, senza quindi dover contare pacchetto per pacchetto;

la dimensione media dei pacchetti uscenti sarebbe da sostituire con il numero totale di byte uscenti diviso il numero totale di pacchetti uscenti, ed entrambe le grandezze sono disponibili all'interno di conn.log, senza quindi dover contare pacchetto per pacchetto;

la dimensione minima del pacchetto sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene in memoria le dimensioni di tutti i pacchetti;

la dimensione massima del pacchetto sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene in memoria le dimensioni di tutti i pacchetti;

la deviazione standard del pacchetto sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene in memoria le dimensioni di tutti i pacchetti;

il massimo tempo di inter arrivo tra due pacchetti consecutivi sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene i tempi di inter arrivo di tutti i pacchetti consecutivi;

il minimo tempo di inter arrivo tra due pacchetti consecutivi sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene i tempi di inter arrivo di tutti i pacchetti consecutivi;

il tempo di inter arrivo medio tra tutti i pacchetti consecutivi sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene i tempi di inter arrivo di tutti i pacchetti consecutivi;

la deviazione standard del tempo di inter arrivo tra tutti i pacchetti consecutivi sarebbe da eliminare dai parametri di classificazione, in quanto ottenuta solamente tramite script che tiene i tempi di inter arrivo di tutti i pacchetti consecutivi.

4.3 Identificazione classi di traffico

Dopo essere venuto a conoscenza di tutte le classi che venivano menzionate negli articoli letti, ho deciso di implementare la classificazione del traffico solo per alcune di esse. Infatti, per motivi di fattibilità, non sono stati raccolti campioni di traffico per tutte le classi menzionate, soprattutto a causa del fatto che non si riusciva a generare traffico cifrato e allo stesso tempo andare ad intercettarlo per poi salvarlo per tutte le classi che sono state menzionate. Nello specifico, sono state scelte le 8 classi seguenti:

web, che rappresenta il traffico che viene generato navigando sulle pagine web in generale, e quindi su qualunque tipo di sito. Per generare questo tipo di traffico sono stati visitati i siti più comuni, andando a consultare alcune classifiche che si trovano online riguardo i siti più visitati in Italia o nel mondo, come ad esempio amazon.com, intesasanpaolo.com, booking.com e similari[1];

media audio, che rappresenta il traffico che viene generato andando ad ascoltare qualsiasi tipo di audio si possa trovare online. Per generare questo tipo di traffico sono stati utilizzati siti di radio online andando ad isolare solamente il traffico di tipo audio, come ad esempio Rai Radio 1, Radio Capital, RTL 102.5, M2O Radio e Radio Freccia[2];

media video, che rappresenta invece il traffico che viene generato dal alcuni siti che propongono del contenuto multimediale di tipo video, e, come prima, andando successivamente ad isolare solamento il traffico di tipo video. Sono stati sfruttati per questa causa i siti di Mediaset, Rai e altre sorgenti video[3];

file sharing, che rappresenta il traffico che viene generato quando si fa l'upload per la condivisione online di diversi tipi di contenuti sotto forma di file. Sono state sfruttate le possibilità che ci offre Dropbox, Onedrive, Mega, Slack e iCloud[4];

instant messaging, che rappresenta la classe con il traffico generato da qualsiasi dispositivo o applicazione che renda disponibile la funzione di chat verso dispositivi o applicazioni simili. Per fare ciò è stato utilizzato Skype, Telegram e altre sorgenti utilizzate per finalità di messaggistica[5];

ssh, che contiene campioni riguardanti la classe ssh. Questa classe rappresenta il protocollo ftp sicuro, utilizzato per poter trasferire file remoti da linea di comando. Per raccogliere campioni di questa classe è stato usato il tool Filezilla[6], già utilizzato in alcuni corsi presso il Politecnico di Torino per poter

[1] https://www.amazon.com/, https://www.intesasanpaolo.com/, https://www.booking.com/

[2] https://www.raiplayradio.it/radio1/, https://www.capital.it/, https://www.rtl.it/home/, https://www.m2o.it/, https://www.radiofreccia.it/

[3] https://www.mediasetplay.mediaset.it/, https://www.raiplay.it/dirette/?

[4] https://www.dropbox.com/it/, https://onedrive.live.com/about/it-it/, https://mega.nz/, https://slack.com/, https://www.icloud.com/

[5] https://www.skype.com/it/, https://web.telegram.org/#/im

[6] https://filezilla-project.org/

trasferire alcuni file dal proprio PC verso un cluster remoto ad esempio;

peer to peer, che contiene campioni riguardanti traffico Torrent[7], e quindi il miglior generatore di traffico peer to peer, in cui nel dettaglio vengono attivate molte connessioni verso diverse controparti, e non sempre verso la stessa o le stesse come nelle classi precedenti, per poter recuperare le parti mancanti (chunk) in modo da poter scaricare il contenuto desiderato. In particolare, sono state scaricate alcune distribuzioni Linux tramite μtorrent per questo tipo di traffico;

dns over tls, che rappresenta la versione sicura del dns, il quale è stato ricavato installando un software di nome Stubby[8] sulla propria macchina e andando successivamente a fare richieste DNS per risolvere i nomi di alcuni siti;

Sono invece state scartate le seguenti classi di traffico:

- DB;
- mail;
- attacco o traffico malevolo in generale;
- gaming;
- social-network.

Alcune delle classi scartate, oppure solo alcuni campioni appartenenti ad esse, potrebbero comunque essere incluse oppure intersecarsi in quelle inserite nell'esperimento di tesi. Ad esempio il traffico social-network potrebbe in alcune sue sfaccettature essere simile al traffico web.

[7] https://www.utorrent.com/intl/it/utweb-index

[8] https://dnsprivacy.org/wiki/display/DP/Windows+installer+for+Stubby

Capitolo 5

Implementazione della soluzione

5.1 Metodologia di cattura

Inizialmente come strumento per catturare traffico è stato utilizzato Microsoft Network Monitor, che ci permette di operare una cattura per tipo di applicazione. Ad esempio, se si sta generando traffico tramite Skype allora questo tool ci permette di isolare solo quel tipo di traffico. Purtroppo però questo strumento salva i file di uscita con un'estensione propria, .cap, che è diversa da .pcap, e i file con questo tipo di estensione non sono adatti ad essere mandati come input a Bro, il quale non li riconosce e non opera nessun tipo di lavoro su questi file. Nemmeno cambiare il tipo di estensione oppure cercare di convertirli in file .pcap ha avuto successo, poiché si è scoperto che Network Monitor antepone al file un header di produzione propria che a quanto pare non si può togliere in alcun modo.

Abbandonata questa strada si è dunque deciso di utilizzare il già citato Wireshark per la cattura del traffico. La metodologia che è stata utilizzata per la cattura è la seguente: una volta fatto partire Wireshark e dopo aver generato un po' di traffico, si stoppa la cattura e si va alla ricerca delle connessioni aperte durante la generazione del traffico. Dal nome della connessione si può capire se la strada è quella giusta oppure no. Ad esempio se stiamo catturando traffico per conto di amazon.com e abbiamo una connessione verso amazon.com sappiamo che è quella giusta (per quanto riguarda il traffico web), dopodiché andando a vedere il numero di porta sorgente utilizzato per la suddetta connessione si può fare un filtro di questo tipo nella barra in alto di Wireshark:

```
tcp.srcport == 50473 or tcp.dstport == 50473
```

in modo da intercettare tutto il traffico appartenente a quella connessione, compresa la fase di handshake del traffico cifrato. In Figura 5.1 viene riportato un esempio di traffico catturato e salvato in un file .pcap tramite Wireshark. Infatti, una volta fatto il filtro tramite la porta, occorre selezionare tutti i pacchetti per poi andarli a salvare in un file .pcap. Da notare che viene prima svolta la fase di handshake tra le due controparti, che successivamente andranno a scambiarsi i pacchetti cifrati veri e propri. Ogni file .pcap contiene una e una sola connessione per semplicità.

Source	Destination	Protocol	Length	Info
172.25.4.67	192.168.181.231	TCP	66	52324 → 8080 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
192.168.181.231	172.25.4.67	TCP	66	8080 → 52324 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1380 SACK_PERM=1 WS=16
172.25.4.67	192.168.181.231	TCP	54	52324 → 8080 [ACK] Seq=1 Ack=1 Win=17408 Len=0
172.25.4.67	192.168.181.231	HTTP	282	CONNECT www.unicredit.it:443 HTTP/1.1
192.168.181.231	172.25.4.67	TCP	54	8080 → 52324 [ACK] Seq=1 Ack=229 Win=15680 Len=0
192.168.181.231	172.25.4.67	HTTP	123	HTTP/1.1 200 Connection established
172.25.4.67	192.168.181.231	TLSv1.2	571	Client Hello
192.168.181.231	172.25.4.67	TLSv1.2	146	Server Hello, Change Cipher Spec
192.168.181.231	172.25.4.67	TLSv1.2	99	Encrypted Handshake Message
172.25.4.67	192.168.181.231	TCP	54	52324 → 8080 [ACK] Seq=746 Ack=207 Win=17152 Len=0
172.25.4.67	192.168.181.231	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
172.25.4.67	192.168.181.231	TCP	1434	52324 → 8080 [ACK] Seq=797 Ack=207 Win=17152 Len=1380 [TCP segment of a reassembled PDU]
172.25.4.67	192.168.181.231	TCP	1434	52324 → 8080 [ACK] Seq=2177 Ack=207 Win=17152 Len=1380 [TCP segment of a reassembled PDU]
172.25.4.67	192.168.181.231	TLSv1.2	1155	Application Data
192.168.181.231	172.25.4.67	TCP	54	8080 → 52324 [ACK] Seq=207 Ack=2177 Win=19664 Len=0
192.168.181.231	172.25.4.67	TCP	54	8080 → 52324 [ACK] Seq=207 Ack=4658 Win=25344 Len=0
192.168.181.231	172.25.4.67	TLSv1.2	437	Application Data

Figura 5.1. Cattura di esempio per traffico web generato dal sito unicredit.it.

Invece per quanto riguarda alcune classi di traffico, come media audio e media video, non è stato così facile intercettare traffico poiché il nome delle connessioni aperte non riusciva a darci l'idea di quali fossero quelle giuste da isolare. Quindi, la procedura che qui si è seguita è stata cliccare col tasto destro sul flusso multimediale in atto, per poi cliccare su 'Ispeziona', nella finestra che si apriva ci si spostava sul tab 'Network', per poi andare a vedere tra i chunk video/audio scaricati, a quale URL erano dirette le richieste, ed ecco che il nome di questa richiesta era anche quello della connessione che appariva su Wireshark, e dunque anche qui andando a filtrare tramite il numero di porta sorgente si otteneva il traffico voluto.

Per quanto riguarda le classi di traffico instant messaging, ssh, dns over tls e peer to peer non è stato possibile catturare il traffico corrispondente direttamente in azienda a causa di un proxy aziendale che bloccava diverse porte, servizi ludici e download e pagine non sicure. Dunque il suddetto traffico è stato catturato da casa. Da notare infatti che il traffico catturato in azienda ha come porta destinazione la 8080 (probabile segno che siamo dietro a un proxy server), mentre quello catturato da casa ha come porta destinazione la 443 (porta nota utilizzata per traffico HTTPS), mentre i pacchetti della classe ssh sono tutti incanalati sulla porta destinazione 22, e quelli della classe dns over tls utilizzano la 853. Entrambe queste ultime sono infatti porte note per far passare questi tipi di traffico. Un'altra cosa interessante da notare è che per quanto riguarda il traffico catturato in azienda l'indirizzo sorgente è sempre l'IP della mia macchina, mentre quello di destinazione è sempre giustamente rappresentato dall'indirizzo IP del proxy server, essendo interposto tra la mia macchina e la rete. Esso, infatti, ha funzione di filtro e passacarte tra la mia macchina e ciò che si trova all'esterno: è come se fosse un gateway verso il resto della rete esterna. Quando catturo da casa invece come indirizzo IP sorgente ci sarà quello della mia macchina, mentre come destinazione ci sarà quello con cui sto interagendo che sarà diverso da quello del proxy ovviamente, dato che non è più sulla mia rete.

In appendice si possono trovare alcune direttive per avviare e generare traffico

con il già citato software Stubby.

Occorre catturare almeno circa 100 campioni per il training set, che per un algoritmo di machine learning deve essere circa l'80% del totale dei campioni per quella classe, e quindi il restante 20% mi ha permesso di arrivare a 125 campioni per classe. Il tutto per le 8 classi di traffico appena menzionate ci fa arrivare a 1000 campioni in totale, che costituiscono il dataset complessivo.

5.2 Generazione risultati per ogni file .pcap

Ora che abbiamo riempito il nostro dataset con i relativi file .pcap possiamo dedicarci alla parte algoritmica. Questo capitolo è finalizzato a far capire come è stato ottenuto il file finale, che è stato poi dato in input agli algoritmi di machine learning selezionati per valutarne le performance di classificazione. Il file finale è stato prodotto come un file .txt, ma si può anche convertire in un file .csv per poter essere visualizzato meglio. Il suo contenuto è formato da 1001 righe, in cui nella prima, che rappresenta una sorta di header, vi si trova tutto l'elenco di attributi selezionati per la classificazione che sono stati menzionati nel capitolo precedente, i quali sono stati tutti separati da virgole. Invece, nelle successive 1000 righe, vi possiamo trovare un riga per campione, cioè una riga per ogni file .pcap del dataset e quindi per ogni connessione (ricordiamo che ogni file .pcap contiene una ed una sola connessione), in cui per ogni attributo dichiarato nell'header si trova il valore corrispondente.

Ora, a partire dalla variabile d'ambiente $BRO_HOME, che coincide con la directory /usr/local/bro, navighiamo con:

```
cd /share/bro/site
```

e notiamo che è presente un file, local.bro, il quale contiene alcune policy, è modificabile e non viene toccato durante aggiornamenti e re-installazioni. Per poterlo modificare a piacere basta aggiungere al fondo una direttiva @load con il percorso, a partire dalla directory in cui ci troviamo adesso, del file .bro che abbiamo scritto e inserito nel file system. Ecco che quindi all'interno di local.bro a noi conviene aggiungere un file con la direttiva:

```
@load customScript/myStatistics
```

che ci permette, all'interno della directory che abbiamo definito come customScript, di inserire tutti i nostri file contenenti codice Bro operativo e che saranno inseriti all'interno del motore di Bro quando viene mandato in esecuzione.

5.2.1 Aggiunta statistiche

Come detto nel precedente capitolo, Bro fornisce in output a dei file .pcap dei file .log contenenti alcune informazioni sulla connessione e quant'altro, ma noi vogliamo aggiungere altre informazioni che non sono presenti nei file di log originali, ma che possono lo stesso essere ricavate. Per questo viene qui presentato il codice di myStatistics.bro, menzionato sopra, e che abbiamo incluso nel motore di Bro con la

direttiva appena discussa. Questo script, infatti, produrrà come output un file .log con le statistiche mancanti che a noi interessano (che chiameremo statistics.log).

Brevemente, questo algoritmo va a definire la struttura dati che sarà inserita nel file di log di uscita, con in testa la connection ID. A seguire, per quanto riguarda il tempo di inter arrivo tra i pacchetti, ci sarà la misura di massimo, minimo, media e deviazione standard. Per quanto riguarda invece la dimensione del pacchetto ci sarà il massimo, il minimo, la media e la deviazione standard, e anche la dimensione media dei pacchetti uscenti e quelli entranti. La parte centrale dell'intero algoritmo, che potrà essere visionato per intero nell'appendice A, è rappresentato dalla funzione evento 'packet_contents', che viene attivato ogni volta che nel processamento di un file .pcap viene incontrato un pacchetto contenente un payload di livello trasporto non vuoto. In questo evento, che viene diviso in due per separare le parti riguardanti i calcoli dei tempo di inter arrivo e di dimensione, vengono effettuati tutti i calcoli del caso. Nell'evento 'bro_done', scatenato prima della fine dello script, vengono calcolati gli ultimi valori, come quello di media e deviazione standard, e poi viene tutto salvato nel file di log di uscita.

5.2.2 Aggregazione parametri

Una volta prodotto per un certo file .pcap i file conn.log, ssl.log (oppure ssh.log in caso di connessioni contenenti traffico ssh) e statistics.log, occorre, tramite il loro connection ID (uid), aggregarli, filtrare solo le informazione che ci servono, e creare un file results.log (il già citato file finale) in cui per ogni parametro di ogni connessione viene impostato il valore opportuno. Occorre creare altri due file di script, uno che si chiama myScript.bro, e uno che si chiama myScriptSftp.bro. La ragione dell'aggiunta del secondo file è che nel caso di traffico ssh viene prodotto il file ssh.log e non ssl.log, e quindi bisogna gestire il nome di altri parametri rispetto a quelli contenuti in ssl.log. A parte poche righe di codice questi file sono identici. Per poter essere eseguiti bisognerà anche aggiungere nella directory padre di 'customScript' due file local1.bro e local2.bro, in cui nel primo verrà aggiunta la linea di codice che include il primo script:

```
@load customScript/myScript
```

mentre nel secondo verrà aggiunta la linea di codice che include il secondo script:

```
@load customScript/myScriptSftp
```

Anche per quanto riguarda il contenuto del file myScript.bro viene fornita l'implementazione completa del suo algoritmo nell'appendice A. Nel dettaglio si può vedere come sono dichiarati nella struttura principale e in quella del file di log di uscita gli stessi parametri, in quanto la struttura principale viene riempita a poco a poco andando ad aprire e leggere prima il file conn.log, poi il file ssl.log e poi il file statistics.log, e poi una volta completata la struttura principale viene scaricato tutto il quella di log, che farà prendere forma al file results.log. Gli eventi su cui si appoggia questo script sono *bro_init*, che permette di aprire e iniziare a leggere i file di log di interesse, ed *end_of_data*, che scatta quando uno dei file di log finisce di essere letto e tutte le sue entry sono state inserite nella tabella che è stata

dichiarata come globale, e avente come chiave la connection ID e come valori gli attributi di interesse di quella tabella. Da notare la parte di codice che riguarda il file ssl.log, in cui la cipher suite viene spezzata in più parti e ad ogni algoritmo viene assegnata la sua parte corrispondente. Nel file ssl.log, poi, possiamo notare che gli attributi server_name e subject sono entrambi dichiarati optional perché alcune volte possono apparire senza valore dell'uno o dell'altro. In quel caso viene preso il valore di quale dei due è presente, dando la precedenza a server_name.

Invece, per quanto riguarda lo script myScriptSftp, le parti di differenza sono solo quelle che riguardano i parametri del file ssh.log in luogo del file ssl.log, la dichiarazione ed apertura del file ssh.log e il relativo salvataggio dei parametri nella struttura principale, come viene presentato nell'appendice A.

5.2.3 Script principale

Ora concentriamo la nostra attenzione sull'esecuzione dello script principale in linguaggio bash, che utilizza tutti gli algoritmi e i file che sono stati menzionati precedentemente, per produrre il tanto atteso file finale che contiene tutte le statistiche per ogni file .pcap, con tanto di intestazione con la lista dei parametri. Oltre a ciò bisogna far notare che siamo posizionati all'interno della directory customScript quando eseguiamo lo script bash, e che tutti i file .pcap sono posizionati all'interno della sotto-directory chiamata pcap, la quale contiene a sua volta 8 cartelle, ognuna nominata con il nome della classe di cattura e ognuna quindi contenente i file associati a quella classe. I risultati per ogni file .pcap vengono poi messi all'interno di un'altra sotto-directory di customScript, nominata results, e vengono poi processati da due algoritmi in linguaggio C per ottenere il file merged.txt finale. Questi due algoritmi, che vengono nominati myProgram e myMerge, sono eseguiti in cascata per permettere al primo la formattazione dei file results.log secondo uno schema chiave-valore (infatti dopo questa fase il file avrà un formato chiave-valore per ogni riga, e quindi per ogni attributo), mentre il secondo si occupa di eliminare gli attributi in più che sono stati inseriti come header e che non verranno utilizzati nella classificazione perché sono stati valutati come non utili alla causa.

Per capire meglio quanto detto finora viene presentato nell'appendice A lo script corrispondente all'algoritmo myBash.sh.

5.3 Algoritmi di machine learning utilizzati

Dopo un'attenta selezione con il personale posto all'interno dell'ufficio Data Science di Aizoon, si è deciso di utilizzare i seguenti algoritmi di machine learning:

Random forest, è un algoritmo [17] molto utilizzato per la sua semplicità ed è un'evoluzione del Decision Tree, o albero di decisione, che è un altro algoritmo di classificazione. É un algoritmo white-box, nel senso che si può facilmente capire come opera, viene utilizzato sia per obiettivi di classificazione che per regressione. Per dirlo in parole semplici, l'algoritmo di Random Forest costruisce diversi alberi di decisione e li unisce insieme per ottenere una foresta

in un certo senso casuale, e tutto ciò gli permette di ottenere una predizione più stabile ed accurata. É un algoritmo supervisionato, nel senso che ha bisogno di essere allenato da un training set e poi valutato con un validation set, e non può quindi apprendere solo basandosi sul data set e su calcoli relativi ai campioni in esso contenuti per poter decidere come classificare, come accade invece per gli algoritmi non supervisionati;

Support Vector Machine, è un algoritmo [18] che viene utilizzato spesso ma è di tipo black-box, non è molto semplice da comprendere perché utilizza concetti che riguardano gli iper-piani. Infatti, ogni punto viene rappresentato in uno spazio a n dimensioni, dove n è il numero di caratteristiche che sono in gioco, e ogni caratteristica ha un valore che viene rappresentato anch'esso come una coordinata in questo spazio. La classificazione viene dunque effettuata cercando di trovare l'iper-piano ottimo che differenzia due classi. É un algoritmo supervisionato, e, come il Random Forest, può essere utilizzato sia per classificazione che per regressione;

K-Nearest Neighbors, questo è invece un algoritmo [19] utilizzato per classificazione, non è supervisionato e ha come quello precedente bisogno di una fase di pre-processing dei dati per poter essere eseguito. Opera una classificazione andando a vedere la distanza che un certo campione ha dai suo k vicini (con k numero dispari preferibilmente, in modo da evitare casi di parità);

Naive Bayes, in cui possiamo dire che è un algoritmo [20] molto veloce ma anche più impreciso rispetto a quelli menzionati in precedenza. Infatti il Naive Bayes lo abbiamo utilizzato solo come prova per il fatto di farci notare che le sue prestazioni sono inferiori rispetto agli altri algoritmi di classificazione. Come vedremo, queste ipotesi ci verranno confermate. Questo algoritmo si appoggia sul teorema di Bayes, e si basa sulla conoscenze della teoria della probabilità, soprattutto quella a priori che possa verificarsi un certo evento. Viene anche assunta l'indipendenza tra le caratteristiche in gioco, cioè che la presenza di una non implica anche la presenza di un'altra.

Occorre notare che alcuni di questi algoritmi hanno bisogno di una fase di pre-processing per essere eseguiti. Infatti i classificatori non possono avere a che fare con numeri troppo distanti tra di loro. Ad esempio, il minimo tempo di inter arrival packet time è solitamente un numero estremamente piccolo che si avvicina allo zero (ordine dei microsecondi), mentre altri numeri come il numero di byte che vengono scambiati in un secondo sono molto alti (ordine delle centinaia di migliaia). Ecco quindi che si rende indispensabile una fase in cui il dato viene normalizzato secondo regole ben precise [21], ad esempio tramite la normalizzazione minimo-massimo, secondo la formula:

$$\frac{(X - min(X))}{(max(X) - min(X))}$$

, in cui X è il valore del campione e min e max rappresentano rispettivamente il massimo e il minimo valore che ha assunto quel tipo di campione. Come alternativa si può anche utilizzare la standardizzazione Z-score, in quanto la normalizzazione minimo-massimo tende a portare il dato vicino alla media e questo potrebbe

rappresentare uno svantaggio. La normalizzazione verrà effettuata utilizzando determinate funzioni, come vedremo, sia per l'algoritmo Support Vector Machine che per il K-Nearest Neighbors.

Tutti gli algoritmi invece hanno bisogno di una fase di pre-processing in cui gli attribuiti non numerici, come i componenti della cipher suite che sono in forma di stringhe, devono essere convertiti in numeri, andando ad aumentare le colonne della tabella degli attributi poiché per ogni stringa diversa che appare viene aggiunta una colonna, che avrà come valore un 1 oppure uno 0 nel caso l'attributo assuma proprio quel dato valore per quella connessione.

Un altro aspetto di vitale importanza nell'implementazione degli algoritmi che si è voluto implementare è stata la cross validation [22], detta anche convalida o validazione incrociata. In particolare si è voluto sfruttare la k-fold cross validation , tecnica che consiste nel dividere l'intero dataset in k parti uguali, in cui a turno se ne lascia fuori uno (che diventa un validation set) che si cerca di predire utilizzando come training set gli altri k-1. In questo modo si evita il rischio che i campioni che vengono utilizzati come training e come validation set siano troppo sbilanciati, e quindi si potrebbe allenare un classificatore con i campioni di solo una determinata classe, e poi finire nel caso in cui il classificatore cerca di classificare i campioni di una classe che non ha mai visto, ad esempio. Questo fenomeno viene detto campionamento asimmetrico, mentre un altro anche molto conosciuto e che la k-fold cross validation permette di evitare è l'overfitting, che si verifica quando un classificatore è stato allenato per troppo tempo sempre con gli stessi e con troppi campioni, oppure al contrario è stato allenato con pochissimi campioni.

Come misure di classificazione [23], nel senso di misure e quantità che diano una stima o comunque quantifichino la bontà del classificatore, sono state prese in considerazione:

- l'accuratezza;
- la precisione;
- il richiamo;
- l'F1-score.

Questi parametri vengono calcolati utilizzando 4 misure, che vengono riassunte nella tabella 5.1, che rappresenta una matrice di confusione.

	Classe = si	Classe = no
Classe = si	*Vero positivo*	*Falso negativo*
Classe = no	*Falso positivo*	*Vero negativo*

Tabella 5.1. Matrice di confusione contenente le 4 grandezze utilizzate per ottenere le misure di classificazione (fonte blog.exsilio.com).

La nostra speranza è quella di massimizzare le misure che sono sulla diagonale principale, e che sono quindi i veri positivi (TP) e i veri negativi (TN), e contemporaneamente minimizzare i falsi positivi (FP) e i falsi negativi (FN): essendo una matrice di confusione i valori che sono sulla diagonale principale sono considerati buoni, mentre tutti gli altri no. Le 4 misure di bontà della nostra classificazione possono essere quindi calcolate dalle seguenti formule:

$$Accuratezza = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precisione = \frac{TP}{TP + FP}$$

$$Richiamo = \frac{TP}{TP + FN}$$

$$F1 - score = \frac{2 * (Richiamo) * (Precisione)}{Richiamo + Precisione}$$

5.3.1 Implementazione degli algoritmi di machine learning

Per implementare gli algoritmi di machine learning utilizzati è stato utilizzato il linguaggio di scripting Python, che è ad alto livello e orientato agli oggetti, ed è stato scaricato tramite Anaconda, il quale a sua volta contiene l'IDE Spyder, che è stato utilizzato in fase di sviluppo. Una valida alternativa poteva essere rappresentata dall'adozione del linguaggio R. Prima di mandare in input il file contenente tutte le connessioni è stato notato che esse erano in ordine di classe, cioè per prime tutte quelle della classe dns, poi tutte quelle della classe file sharing, e così via. Quindi per evitare, o comunque abbassare la probabilità, di classificatori non allenati a dovere o comunque nella maniera sbagliata è stato aggiunto un programma scritto in Java che mischiasse le carte in tavola, prendendo le righe del file con tutte le connessioni e mischiandole in maniera randomica tramite la funzione shuffle della classe Collections.

Fatte queste precisazioni possiamo dedicarci alla visione degli algoritmi di machine learning utilizzati. Viene presentato il codice commentato del Random Forest in appendice A. Solo nell'algoritmo del Random Forest è possibile stampare la lista delle variabili e della loro rispettiva importanza grazie all'utilizzo di un attributo del classificatore. Negli altri algoritmi non esiste questa possibilità. Quello che cambia negli altri algoritmi è solo la dichiarazione dei diversi classificatori, la fase di preprocessing (per attributi troppo lontani tra loro) per quanto riguarda gli algoritmi di SVM e K-Nearest Neighbors, e appunto la parte mancante sull'importanza degli attributi. Nel dettaglio per quanto riguarda l'algoritmo di Support Vector Machine cambiano i seguenti statements:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
classifier = SVC()
```

mentre per quanto riguarda il K-Nearest Neighbors:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=5)
```

e per il Naive Bayes:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB().
```

Da notare che la k-fold cross validation è stata impostata con un valore di 10 fold (quindi 100 campioni per ogni fold, dato che quelli totali sono 1000) e una ripetizione, in tal modo in totale ci saranno in output 10 stampe di risultati, dove ogni risultato comprende i 4 parametri già menzionati utilizzati per la valutazione. Anche i 4 classificatori utilizzati sono stati inizializzati con parametri abbastanza standard, ma si potevano anche inizializzare con altri valori o con più parametri in input. Ad esempio l'algoritmo di Random Forest può essere utilizzato andando ad inizializzare il classificatore con una lista di parametri diversa. Un esempio [24] a cui ogni parametro viene assegnato un valore potrebbe essere:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=10,
   criterion=gini, max_depth=None, min_samples_split=2,
   min_samples_leaf=1, min_weight_fraction_leaf=0.0,
   max_features=auto, max_leaf_nodes=None,
   min_impurity_decrease=0.0, min_impurity_split=None,
   bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
   verbose=0, warm_start=False, class_weight=None)
```

e quindi si può andare a fare un tuning di parametri e lanciare ogni volta l'algoritmo per valutare se le prestazioni migliorano. É stato provato questo approccio con un sottoinsieme di possibilità ma non sono stati ottenuti risultati tanto diversi da quelli che verranno discussi.

Capitolo 6

Risultati e conclusioni

6.1 Risultati degli algoritmi di machine learning

Dopo aver lanciato gli algoritmi di Random Forest, Support Vector Machine, K-Nearest Neighbors e Naive Bayes, e utilizzando i parametri per impostare i classificatori che sono stati descritti nel precedente capitolo, sono stati raccolti i seguenti risultati che vengono mostrati in tabelle per una migliore comprensione. Ogni algoritmo è stato eseguito due volte: una impostando il k-fold cross validator con 10 folds e una ripetizione, mentre l'altra impostando 5 folds e due ripetizioni. Le prestazioni migliori, seppur di pochissimo, sono state ottenute nel primo caso, quindi verranno presentati quei valori. Sono stati comunque provati diversi valori per quanto riguarda il tuning di questi parametri, ma i risultati sono risultati essere inferiori o equivalenti. Nel dettaglio, in tabella ci saranno per ogni algoritmo 10 iterazioni, in quanto il dataset viene diviso in 10 parti e una parte a giro diventa il validation set, e per ognuna di esse saranno disponibili i valori di accuratezza, precisione, richiamo e F1-score in termini di percentuale. In fondo ci sarà poi per ognuna di queste misure di bontà la media tra le 10 iterazioni, che rappresenta la media totale per quel tipo di algoritmo, anch'essa in termini percentuali.

Nella tabella 6.1 vengono raffigurati i risultati per quanto riguarda l'algoritmo di Random Forest, con le medie sul totale rappresentate in grassetto nell'ultima riga.

Nella tabella 6.2 vengono raffigurati i risultati per quanto riguarda l'algoritmo di Support Vector Machine, con le medie sul totale rappresentate in grassetto nell'ultima riga.

Nella tabella 6.3 vengono raffigurati i risultati per quanto riguarda l'algoritmo di K-Nearest Neighbors, con le medie sul totale rappresentate in grassetto nell'ultima riga.

Nella tabella 6.4 vengono raffigurati i risultati per quanto riguarda l'algoritmo di Naive Bayes, con le medie sul totale rappresentate in grassetto nell'ultima riga. Visionando queste tabelle ci rendiamo subito conto che gli algoritmi che hanno

# Iterazione	*Accuratezza* [%]	*Precisione* [%]	*Richiamo* [%]	*F1-score* [%]
1	96,0	96,0	96,0	96,0
2	91,0	92,0	91,0	91,0
3	91,0	93,0	91,0	91,0
4	91,0	92,0	91,0	91,0
5	94,0	95,0	94,0	94,0
6	89,0	91,0	89,0	89,0
7	92,0	93,0	92,0	92,0
8	95,0	96,0	95,0	95,0
9	96,0	96,0	96,0	96,0
10	89,0	91,0	89,0	89,0
Media/Totale	**92,4**	**93,5**	**92,4**	**92,4**

Tabella 6.1. Risultati algoritmo Random Forest.

# Iterazione	*Accuratezza* [%]	*Precisione* [%]	*Richiamo* [%]	*F1-score* [%]
1	96,0	96,0	96,0	96,0
2	98,0	98,0	98,0	98,0
3	94,0	95,0	94,0	94,0
4	96,0	96,0	96,0	96,0
5	95,0	96,0	95,0	95,0
6	100,0	100,0	100,0	100,0
7	89,0	96,0	89,0	89,0
8	91,0	92,0	91,0	91,0
9	97,0	97,0	97,0	97,0
10	96,0	96,0	96,0	96,0
Media/Totale	**95,2**	**96,2**	**95,2**	**95,2**

Tabella 6.2. Risultati algoritmo Support Vector Machine.

le migliori prestazioni sono la Support Vector Machine e il K-Nearest Neighbors, che superano entrambi il 95% per quanto riguarda tutte le medie sulle misure di bontà della classificazione appena effettuata. Le performance scendono appena con il Random Forest, e scendono ancora un pochino attestandosi a circa il 90% per quanto riguarda il Naive Bayes, del quale abbiamo la prova che sia più veloce come tempi di risposta e quindi di classificazione, ma in buona sostanza è il più rozzo tra i 4 algoritmi scelti. Anche il Random Forest non è di certo il migliore anche per quanto riguarda i tempi di classificazione, in quanto è stato notato in fase di esecuzione di come esso sia il più lento fra tutti. Il K-Nearest Neighbors e la Support Vector Machine invece hanno tempi di risposta abbastanza rapidi. Tra le misure di performance degli algoritmi esposti invece è da notare quella che indica la precisione, che è in tutti i casi appena più alta degli altri valori. I risultati sembrano comunque essere in generale molto buoni, in quanto in almeno 9 casi su 10 mediamente il classificatore riesce a fare bene il suo lavoro, anche grazie ad una fase di allenamento proficua. Infatti per ogni iterazione il classificatore viene

# Iterazione	*Accuratezza* [%]	*Precisione* [%]	*Richiamo* [%]	*F1-score* [%]
1	100,0	100,0	100,0	100,0
2	96,0	96,0	96,0	96,0
3	98,0	98,0	98,0	98,0
4	95,0	95,0	95,0	95,0
5	98,0	98,0	98,0	98,0
6	97,0	97,0	97,0	97,0
7	96,0	97,0	96,0	96,0
8	98,0	98,0	98,0	98,0
9	99,0	99,0	99,0	99,0
10	97,0	97,0	97,0	97,0
Media/Totale	**97,4**	**97,5**	**97,4**	**97,4**

Tabella 6.3. Risultati algoritmo K-Nearest Neighbors.

# Iterazione	*Accuratezza* [%]	*Precisione* [%]	*Richiamo* [%]	*F1-score* [%]
1	86,0	88,0	86,0	85,0
2	94,0	95,0	94,0	94,0
3	90,0	91,0	90,0	90,0
4	89,0	90,0	89,0	89,0
5	92,0	94,0	92,0	92,0
6	86,0	87,0	86,0	86,0
7	90,0	92,0	90,0	90,0
8	93,0	94,0	93,0	93,0
9	93,0	94,0	93,0	93,0
10	87,0	91,0	87,0	88,0
Media/Totale	**90,0**	**91,6**	**90,0**	**90,0**

Tabella 6.4. Risultati algoritmo Naive Bayes.

allenato con 900 campioni in questo caso, per poi andare a classificarne i 100 del fold rimasto fuori.

Per quanto riguarda i tempi di risposta, e quindi di classificazione, degli algoritmi appena visti, si è deciso di impostare un timer via software per poter cronometrare. Il più lento è sicuramente per distacco il Random Forest, che raggiunge risultati abbastanza buoni ma impiegandoci davvero parecchio tempo in quanto mediamente termina in 26,5 secondi. Gli algoritmi di Support Vector Machine e K-Nearest Neighbors invece sono quelli che hanno le performance molto buone e rispondono in breve tempo, rispettivamente 0,230 secondi il primo e 0,105 secondi l'altro, che quindi oltre ai risultati migliori ha anche il tempo dalla sua parte. Invece il Naive Bayes risponde ancora più velocemente (0,054 secondi), ma ha purtroppo le prestazioni peggiori se confrontato con gli altri.

6.1.1 Parametri e relativa importanza

A fianco a questo risultati, si può anche andare a fare uno studio supplementare riguardante l'importanza di ogni parametro. Infatti, come abbiamo visto al termine del precedente capitolo, per quanto riguarda il classificatore istanziato con l'algoritmo del Random Forest, viene anche resa disponibile una funzione che ci permette di far restituire al classificatore la lista degli attributi utilizzati per la classificazione associata alla loro importanza. Negli altri algoritmi non è disponibile questa funzione in quanto il Random Forest è l'unico algoritmo tra quelli trattati ad essere etichettato come white-box, cioè è abbastanza trasparente a chi lo utilizza il suo modo di procedere nella classificazione. Questo non è vero per gli altri algoritmi, che sono etichettati come black-box, e di cui quindi non è così chiaro il modus operandi.

Viene dunque fornita la lista di parametri stampata a video nell'algoritmo di Random Forest, con i valori di importanza che possono variare tra 0 e 1, assumendo tutti i possibili sotto valori con la virgola compresi tra di essi:

- numero di byte in entrata, con un'importanza di 0.31;
- porta destinazione, con un'importanza di 0.16;
- dimensione del pacchetto medio in entrata, con un'importanza di 0.09;
- durata della connessione, con un'importanza di 0.08;
- massima dimensione di un pacchetto, con un'importanza di 0.06;
- percentuale di pacchetti uscenti, entranti, percentuale di byte uscenti ed entranti, tutti con un'importanza di 0.03;
- deviazione standard della dimensione dei pacchetti e algoritmo di cifratura asimmetrico RSA, entrambi con un'importanza di 0.02;
- numero di pacchetti in uscita, in entrata, pacchetti totali, dimensione media di un pacchetto, dimensione del pacchetto medio in uscita, numero di byte uscenti ed ECDHE-RSA come algoritmo di cifratura asimmetrica, tutti con un'importanza di 0.01;
- il resto degli attributi utilizzati per la classificazione, incluse le varie opzioni di cifratura simmetrica, asimmetrica, digest e algoritmi a blocchi che sono stati trovati, tutti con un'importanza di 0.00.

Si può notare come l'attributo che riveste maggiore importanza nella classificazione sia proprio il numero di byte in entrata, con un valore che indica quanto sia discriminante in circa una classificazione su 3. Dopodiché il numero di porta destinazione, che abbiamo già osservato in fase di cattura: infatti, esso assume il valore 22 per il traffico riguardante il protocollo ssh, il valore 853 per il traffico dns over tls, mentre per gli altri tipi di traffico assume valore 8080 (indica che siamo dietro a un proxy server) per le catture svolte in azienda e 443 per quelle svolte altrove. É naturale pensare che se avessimo svolto tutte le catture altrove oppure tutte le catture in

azienda probabilmente questo attributo avrebbe pesato meno perché avremmo solo avuto rispettivamente la 443 oppure la 8080. I parametri che sono stati classificati con peso 0.00 invece possono in teoria essere eliminati, ma potrebbero essere utili agli altri algoritmi di classificazione, ma purtroppo come già detto non possiamo avere un riscontro pratico in questo caso, in quanto non ci viene fornita la loro reale apporto nella classificazione come invece ci viene fornito dal Random Forest. Ci sono anche diversi parametri per quanto riguarda la cipher suite. Infatti ogni pezzo di cipher suite in origine era una stringa ed è stato trasformato, ogni pezzo, in tanti valori quanti ne assumono tutti i campioni. Ad esempio, per quanto riguarda la cifratura simmetrica, si possono avere diversi valori come AES-128, AES-256 e tanti altri, che fanno aumentare la cardinalità delle colonne ma non più di tanto. Ad ogni riga quindi sarà assegnato il valore 1 oppure 0 in corrispondenza del suddetto attributo nel caso il campione assuma proprio quel valore (di cifratura simmetrica ad esempio) oppure no. É stato notato come all'aumentare di questi valori si abbassi anche la loro importanza ai fini della classificazione, quindi con un dataset di file .pcap di grosse dimensioni è quasi trascurabile l'attributo cipher suite, da cui derivano poi successivamente 4 attributi. Tutto ciò è valido a meno che non venga ad esempio scoperto che un certo tipo o classe di traffico cifrato è sempre o quasi sempre ottenuto applicandogli ad esempio un certo algoritmo di cifratura simmetrica, ma è tutto da valutare.

Riprendendo invece il discorso dell'applicabilità in Aramis, come è stato argomentato nel capitolo 4, ci sono alcuni parametri come la dimensione media del pacchetto (e tutte le altre statistiche che sono state aggiunte per una connessione) che non possono essere calcolate in modo corretto come è stato fatto nello script myStatistics.bro, ma può essere solo fornita una stima per i motivi già discussi. Fornire il valore preciso e non una stima potrebbe degradare le prestazioni del classificatore perché quest'ultimo avrà più difficoltà a crearsi dei range di valori che possano identificare una certa classe, essendo le misure più precise e quindi ragionevolmente più diverse rispetto a delle stime che potrebbero più facilmente andare a isolarsi in certi intervalli. Anche questo discorso è da valutare poiché per alcune connessioni è stato notato, come già detto in precedenza, che le stime si sono avvicinate molto ai valori precisi.

6.1.2 Mappe di calore

Tramite le mappe di calore, dette anche heat map, si può andare a far luce su alcuni aspetti importanti della nostra classificazione. Infatti, tramite il codice degli algoritmi utilizzati, si possono stampare le matrici di confusione, che indicano per ogni classe quanti campioni sono stati classificati correttamente sul totale, e tra quelli che non sono invece stati classificati correttamente, si può vedere presso quale classe sono stati erroneamente etichettati. Questo ci può dare indicazione su quali classi il classificatore sbaglia e alcune volte potremmo anche intuirne il motivo. Ad esempio per quanto riguarda la classe media audio se notiamo che i campioni di questa classe vengono a volte erroneamente posizionati all'interno della classe media video, allora possiamo anche capire come il classificatore possa sbagliarsi tra due classi che sono molto simili tra loro, mentre lo possiamo capire meno se si sbagliasse e posizionasse i campioni nella classe ssh, che in questo caso

è più distante in termini di valori da media audio, e quindi l'errore sarebbe più grossolano. Questo è un indice del fatto che il classificatore potrebbe non essere stato allenato a sufficienza o in maniera scorretta.

Le mappe di calore utilizzate si basano su matrici di confusione, in cui i valori più alti dovrebbero stare sulla diagonale principale se gli algoritmi di classificazione hanno operato in maniera corretta. Viceversa, i valori saranno spostati verso altre posizioni se la classificazione ha sbagliato l'etichettamento. I colori vanno dal verde scuro al rosso, rispettivamente per indicare valori alti e bassi, con tutte le gradazioni intermedie corrispondenti agli altri valori. Il caso ideale è quindi rappresentato da numeri alti (verde scuro) tutti sulla diagonale principale, e valori bassi (zero, quindi rosso) nelle altre posizioni.

Nella Figura 6.1 si può vedere la rappresentazione come heat map di un risultato medio ottenuto con un algoritmo di Random Forest.

	dns	sharing	messaging	audio	video	p2p	sftp	web
dns	12	0	0	0	0	0	0	0
sharing	0	11	1	0	0	0	0	0
messaging	0	0	12	1	0	0	0	0
audio	0	0	0	10	1	0	0	0
video	0	0	0	0	11	0	0	0
p2p	0	0	0	0	2	9	0	0
sftp	0	0	0	0	0	1	16	0
web	0	0	0	0	1	1	4	7

Figura 6.1. Mappa di calore per l'algoritmo di Random Forest.

Nella Figura 6.2 si può vedere la rappresentazione come heat map di un risultato medio ottenuto con un algoritmo di Support Vector Machine.

	dns	sharing	messaging	audio	video	p2p	sftp	web
dns	8	0	0	0	0	0	0	0
sharing	0	9	0	0	0	0	0	0
messaging	0	0	8	0	0	0	0	0
audio	0	0	0	15	0	0	0	0
video	0	0	0	4	14	0	0	0
p2p	0	0	0	0	0	11	0	0
sftp	0	0	0	0	0	0	15	0
web	0	0	0	0	0	0	0	16

Figura 6.2. Mappa di calore per l'algoritmo di Support Vector Machine.

Nella Figura 6.3 si può vedere la rappresentazione come heat map di un risultato medio ottenuto con un algoritmo di K-Nearest Neighbors.

Nella Figura 6.4 si può vedere la rappresentazione come heat map di un risultato medio ottenuto con un algoritmo di Naive Bayes.

Per ogni iterazione erano presenti 100 campioni da etichettare, con un numero di campioni per ogni classe che come si può vedere dai risultati oscillava attorno alla decina (vengono presi i campioni dalle classi per il training set in maniera casuale). Anche da queste figure si nota che gli algoritmi più performanti sono la Support Vector Machine e il K-Nearest Neighbors, in quanto hanno il maggior

	dns	sharing	messaging	audio	video	p2p	sftp	web
dns	14	0	0	0	0	0	0	0
sharing	0	12	0	0	0	0	0	0
messaging	0	0	11	0	0	0	0	0
audio	0	0	0	10	0	0	0	0
video	0	0	0	1	15	0	0	0
p2p	0	0	0	0	0	10	0	0
sftp	0	0	0	0	0	0	11	0
web	0	0	0	0	0	0	0	16

Figura 6.3. Mappa di calore per l'algoritmo di K-Nearest Neighbors.

	dns	sharing	messaging	audio	video	p2p	sftp	web
dns	16	0	0	0	0	0	0	0
sharing	0	10	0	0	0	0	0	0
messaging	0	0	15	0	0	2	0	0
audio	0	0	0	14	2	0	0	0
video	0	0	0	1	12	0	0	0
p2p	2	0	0	0	0	4	0	0
sftp	0	0	0	0	0	0	10	0
web	0	0	3	3	0	0	0	6

Figura 6.4. Mappa di calore per l'algoritmo di Naive Bayes.

numero di campioni piazzati correttamente, e ciò vale come prova per quanto detto precedentemente. In dettaglio, per quanto riguarda l'algoritmo di Random Forest si può notare come ci siano alcuni errori per le classi media video, peer to peer , ssh e web, che a quanto pare il classificatore ha notato essere simili. É possibile che questi piccoli errori siano dovuti a range di valori comuni per alcuni attributi di cui noi non ci siamo accorti. Per quanto riguarda invece la Support Vector Machine e il K-Nearest Neighbors gli errori sono ancora più limitati, e in questo caso solo tra le classi media audio e media video, classi molto simili tra loro per caratteristiche. Per finire, per quanto riguarda l'algoritmo di Naive Bayes si possono trovare alcuni campioni sparsi a dimostrazione dell'algoritmo meno preciso tra quelli selezionati. In tal senso, l'errore più grossolano sembra registrarsi riguardo la classe web che ha visto classificare in maniera scorretta la metà dei suoi campioni, che sono finiti con le etichette delle classi instant messaging e media audio.

6.2 Considerazioni finali

Nonostante i risultati sembrano essere buoni per gran parte delle esecuzioni ci possono essere sicuramente dei margini di miglioramento che possono far avvicinare le misure di performance ancora di più verso il 100%. Una miglioria in tal senso potrebbe essere quella di far eseguire gli algoritmi con a turno un parametro in meno per la classificazione per andare a notare quali sono i parametri che, se tolti, non fanno peggiorare le performance medie, in modo che possano essere eliminati. L'obiettivo quindi sarebbe quello di ridurre il numero di parametri al minimo possibile. Oppure si potrebbero tenere i parametri che hanno un peso sopra una certa soglia, ed andare ad eliminare quelli che hanno un peso irrisorio oppure sotto una certa

soglia, e notare se le performance salgono o comunque rimangono invariate. Questo tipo di approccio, configurabile a piacimento, potrebbe portare un giovamento in tal senso, e ridurrebbe la parte computazionale di generazione degli attributi da utilizzare per ogni file .pcap, e quindi il miglioramento sarebbe sia in termini di spazio che di tempo. Tempo che ridurrebbero anche gli algoritmi di classificazione, che ritrovandosi ad avere a che fare con meno attributi ridurrebbero sicuramente i tempi di risposta.

Tralasciando invece i vantaggi di spazio e tempo un lavoro più completo può essere rappresentato dall'aggiungere al dataset le classi mancanti che non sono state prese in considerazione, e aggiungere anche classi che non sono state menzionate nelle ricerche lette. Non è sicuramente un lavoro facile in quanto ad esempio etichettare la classe di attacco è impresa ardua se non impossibile, anche solamente raccogliere una parte dei suoi campioni. Infatti come sappiamo il traffico malevolo sempre più spesso si nasconde vicino a traffico che malevolo non è, e quindi è estremamente difficile raccogliere campioni che abbiano anche dei range di attributi abbastanza stabili. In caso contrario infatti il classificatore avrebbe serie difficoltà ad allenarsi su una classe con così pochi punti di riferimento. Oltre a ciò si può concludere questo discorso dicendo che avere un classificatore che capisca quando sta passando un traffico di tipo malevolo sarebbe quasi un'utopia al giorno d'oggi, poiché ridurrebbe i rischi di attacchi informatici quasi a zero. Ci sono poi altre classi per le quali intercettare il traffico tramite Wireshark è difficile oppure non possibile, e quindi anche in questo senso bisognerebbe trovare altri tools che possano aiutarci da questo punto di vista.

Il tema della classificazione del traffico cifrato è entrato a far parte delle ricerche da qualche anno, e finora sono stati svolti diversi lavori per poter migliorare in un senso o nell'altro questa problematica, che rimane tutt'ora attuale. É tuttavia difficile riuscire ad istruire un classificatore che possa riuscire a classificare ogni tipo di traffico correttamente, e quindi gli studi come questo che si sono svolti finora, si sono concentrati su un loro sottoinsieme o comunque in ambiente limitato. Una forte limitazione di questo lavoro di tesi per esempio è rappresentata dal fatto che sono stati raccolti un numero esiguo (se confrontato con tutto il traffico possibile di una certa categoria) di campioni di traffico. Per poter quindi raggiungere risultati più attendibili si sarebbe dovuto incrementare il dataset, non limitandosi quindi a qualche centinaio di file .pcap, e riuscire anche ad intercettare ad esempio per il traffico media video tutti i tipi di possibili flussi video che si trovano in rete e non solo, per poter rendere un classificatore più robusto. Anche qui per raggiungere questo scopo sarebbero utili anche degli strumenti che possano rendere la raccolta dei campioni più veloce rispetto a quella che è stata fatta a mano durante l'esperimento di tesi, se no si rischierebbe di perdere un sacco di tempo solo per la fase di raccolta e selezione di campioni.

Per quanto riguarda quindi l'applicabilità di questo lavoro all'interno del perimetro aziendale, integrato in un prodotto come Aramis, potrebbe essere provato come detto con alcuni parametri rivisti e sostituiti da alcune stime, mentre altri non sono applicabili date le esigenze spazio-temporali. Ecco che quindi la sua effettiva utilità andrebbe del tutto valutata, e una miglioria da questo punto di vista potrebbe essere rappresentata dall'inserimento di nuovi attributi, nuove caratteristiche che possano aiutare il classificatore, e quindi andare alla ricerca di nuovi

parametri che non sono ancora stati passati al vaglio, oppure riconsiderare i parametri che sono stati scartati perché giudicati non utili, per poterne valutare un loro eventuale reinserimento per rendere il lavoro di classificazione più proficuo.

Appendice A

Appendice

A.1 Istruzioni installazione Bro e cattura traffico

In figura A.1 viene mostrato il codice che permette l'installazione di Bro su di un sistema Linux. Le righe sono state commentate per una maggior chiarezza espositiva.

Listing A.1. Istruzioni lanciate da linea di comando per l'installazione di Bro.

```
# Aggiorna il database dei package.
sudo apt-get update

# Installa tutte le dipendenze in un colpo solo.
sudo apt-get install bison cmake flex g++ gdb make libmagic-dev libpcap-dev
    libgeoip-dev libssl-dev python-dev swig2.0 zlib1g-dev

# Scarica i database per la geo localizzazione degli indirizzi ipv4 e ipv6.
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCityv6-beta/
GeoLiteCityv6.dat.gz

# Decomprime i due file scaricati.
gzip -d GeoLiteCity.dat.gz
gzip -d GeoLiteCityv6.dat.gz

# Li sposta nella directory opportuna, rinominandoli.
sudo mv GeoLiteCity.dat /usr/share/GeoIP/GeoIPCity.dat
sudo mv GeoLiteCityv6.dat /usr/share/GeoIP/GeoIPCityv6.dat

# Clona la repository per l'installazione da GitHub. I file saranno posizionati in
    una directory che si chiama bro.
git clone --recursive git://git.bro.org/bro

# Entra nella dierctory bro.
cd bro
```

```
# Lancia la configurazione di Bro.
./configure

# Opera la build del programma.
make

# Installa Bro.
sudo make install
```

Bro ora sarà installato nella directory Linux /usr/local/bro, percorso che dovrà essere salvato nella variabili d'ambiente $PATH e $BRO_HOME per comodità. Occorre poi modificare il file /usr/local/bro/etc/node.cfg per accogliere i pacchetti in transito sull'interfaccia di rete della nostra postazione e il file /usr/local/bro/etc/networks.cfg per introdurre l'indirizzo IP della nostra rete.

Ora, per catturare traffico tramite l'interfaccia di rete che abbiamo specificato occorre lanciare il comando:

```
sudo /usr/local/bro/bin/bro -i enp0s3 -C
```

se l'interfaccia ha quel determinato nome, e così si possono vedere tutti i file di log che prendono forma e si riempiono in base al tipo di traffico che stiamo catturando. Se si modifica il comando precedente con:

```
sudo /usr/local/bro/bin/bro -r filename.pcap -C
```

allora il comando lancerà Bro, che, invece di leggere il traffico dall'interfaccia di rete, analizzerà il traffico contenuto nel file .pcap e fornirà i relativi file .log di output. Ecco quindi che come differenza rispetto a prima Bro ritornerà appena avrà finito di processare il file, mentre con il primo comando tornerà solo quando glielo diremo noi tramite il comando CTRL-C, segnale di stop dell'esecuzione di un determinato processo. In caso contrario continuerà lo sniffing del traffico. L'ultimo argomento del comando (-C) serve per evitare inutili messaggi di warning su terminale in quanto ci permette di ignorare checksum di pacchetti non validi.

A.2 Comandi avvio Stubby

Vengono aggiunte qui alcune direttive riguardo al software Stubby, utile per catturare traffivo dns over tls, che viene installato tramite un installer Windows scaricabile online. Dopodiché occorre aprire la linea di comando con privilegi di amministratore ed eseguire un comando per poter rendere operativo Stubby:

```
"C:\Program Files\Stubby\stubby.exe" -C "C:\Program
   Files\Stubby\stubby.yml" -l
```

per poi aprire un'altra finestra tramite linea di comando sempre con privilegi di amministratore e testare Stubby con una DNS query di esempio, che verrà resa sicura tramite il tool appena installato, e che farà comparire pacchetti cifrati su Wireshark:

```
"C:\Program Files\Stubby\getdns_query" -s @127.0.0.1
   www.example.com
```

A.3 Algoritmo completo myStatistics.bro

Di seguito nell figura A.2 viene fornito il codice in linguaggio Bro dell'algoritmo myStatistics.bro, menzionato nel corso del capitolo 5 e che ci permette di aggiungere alcune statistiche alle connessioni cifrate utilizzando uno script da inserire nel motore di Bro. Il codice è stato commentato nei punti salienti.

Listing A.2. Codice contenuto in myStatistics.bro.

```
% Appende il valore LOG all'enumerabile Log::ID. Serve per definire il file di log di
    uscita.
redef enum Log::ID += { LOG };

% Struttura che contiene tutte le informazioni aggiuntive che saranno scritte nel file
    di uscita, con l'uid che permette il merge tra files. A fianco possiamo trovare il
    tipo del parametro, ed il fatto che sia utilizzato per essere scritto su un file di
    log.
type Info: record {
        uid : string &log;
        maximum_inter_arrival : double &log;
        minimum_inter_arrival : double &log;
        avg_inter_arrival : double &log;
        std_dev_inter_arrival : double &log;
        avg_pkt_dim : double &log;
        avg_forward_pkt_dim : double &log;
        avg_backward_pkt_dim : double &log;
        maximum_pkt_dim : count &log;
        minimum_pkt_dim : count &log;
        std_dev_pkt_dim : double &log;
};

% Struttura che contiene tutte le differenze di tempo tra i pacchetti.
type Times : record {
        difference : double;
};

% Struttura che contiene tutte le dimensioni dei pacchetti.
type Dimensions: record {
        dim : count;
};

% Dichiarazione variabili globali per le differenze di tempo.
global last_inter : double;
global cont_inter : count = 0;
global tot_inter : double = 0.00;
global max_inter : double;
global min_inter : double;

% Dichiarazione variabili globali per le dimensioni.
global cont_dim : count = 0;
global tot_dim : count = 0;
```

```
global min_dim : count = 1000000;
global max_dim : count = 0;
global last_size_orig : count = 0;
global last_size_resp : count = 0;
global tot_orig : count = 0;
global tot_resp : count = 0;
global cnt_orig : count = 0;
global cnt_resp : count = 0;

% Dichiarazione variabili globali di aiuto.
global flag : bool = T;
global uid : string;

% Dichiarazione strutture.
global times : vector of Times;
global dimensions : vector of Dimensions;

% Evento che viene scatenato prima di qualsiasi altro quando viene sottomesso un
    file .pcap a Bro. Viene usato per dichiarare il file di uscita statistics.log.
event bro_init() {
        Log::create_stream(LOG, [$columns=Info, $path="statistics"]);
}

% Questo evento viene scatenato per ogni pacchetto che abbia un payload di livello
    trasporto che non sia vuoto.
event packet_contents (c: connection, contents: string) {
        % Parte per l'aggiornamento parametri riguardanti la dimensione del
            pacchetto. Una per i pacchetti uscenti e una per quelli entranti.
        local d : count;
        if(c$orig$size != last_size_orig)
        {
                d = c$orig$size - last_size_orig;
                tot_orig = tot_orig + d;
                cnt_orig = cnt_orig + 1;
                last_size_orig = c$orig$size;
                tot_dim = tot_dim + d;
                cont_dim = cont_dim + 1;
                dimensions[|dimensions|] = Dimensions($dim = d);
                if(d < min_dim)
                        min_dim = d;
                if(d > max_dim)
                        max_dim = d;
        }
        if(c$resp$size != last_size_resp)
        {
                d = c$resp$size - last_size_resp;
                tot_resp = tot_resp + d;
                cnt_resp = cnt_resp + 1;
                last_size_resp = c$resp$size;
                tot_dim = tot_dim + d;
                cont_dim = cont_dim + 1;
```

```
                dimensions[|dimensions|] = Dimensions($dim = d);
                if(d < min_dim)
                        min_dim = d;
                if(d > max_dim)
                        max_dim = d;
        }

        % Parte per l'aggiornamento parametri riguardanti i tempi di inter arrivo
            tra i pacchetti.
        local this : double = time_to_double(network_time());
        local diff : double;
        if(flag == T)
        {
                uid = c$uid;
                flag = F;
                last_inter = this;
                max_inter = 0.00;
                min_inter = 100000.00;
        }
        else
        {
                diff = this - last_inter;
                tot_inter = tot_inter + diff;
                cont_inter = cont_inter + 1;
                if(diff > max_inter)
                        max_inter = diff;
                if(diff < min_inter)
                        min_inter = diff;
                last_inter = this;
                times[|times|] = Times($difference = diff);
        }
}

% Evento scatenato per ultimo dopo la sottomissione di un file .pcap. Permette il
    calcolo di media e deviazione standard per tempi di inter arrivo e
    successivamente per la dimensione del pacchetto. Infine salva in statistics.log la
    connection ID e tutte le statistiche calcolate.
event bro_done() {
        local avg : double;
        local std_dev : double;
        local tot_d : double = tot_inter;
        local cont_d : double = cont_inter;
        avg = tot_d / cont_d;
        local sums : double;
        local cnt : double = 0.00;
        for(i in times)
        {
                sums = times[i]$difference - avg;
                sums = sums * sums;
                cnt = cnt + sums;
        }
```

```
        cnt = cnt / cont_d;
        std_dev = sqrt(cnt);

        local tot_dim_d : double = tot_dim;
        local cont_dim_d : double = cont_dim;
        local avg_dim : double;
        avg_dim = tot_dim_d / cont_dim_d;
        local std_dev_dim : double;
        local sums_dim : double;
        local c_dim : double = 0.00;
        local dim_d : double;

        for(i in dimensions)
        {
                dim_d = dimensions[i]$dim;
                sums_dim = dim_d - avg_dim;
                sums_dim = sums_dim * sums_dim;
                c_dim = c_dim + sums_dim;
        }
        c_dim = c_dim / cont_dim_d;
        std_dev_dim = sqrt(c_dim);

        local avg_forward : double;
        local avg_backward : double;
        local tot_orig_d : double = tot_orig;
        local cnt_orig_d : double = cnt_orig;
        local tot_resp_d : double = tot_resp;
        local cnt_resp_d : double = cnt_resp;
        avg_forward = tot_orig_d / cnt_orig_d;
        avg_backward = tot_resp_d / cnt_resp_d;

        Log::write(LOG,[$uid = uid, $maximum_inter_arrival = max_inter,
            $minimum_inter_arrival = min_inter, $avg_inter_arrival = avg,
            $std_dev_inter_arrival = std_dev,
            $avg_pkt_dim = avg_dim, $maximum_pkt_dim = max_dim,
            $minimum_pkt_dim = min_dim, $std_dev_pkt_dim = std_dev_dim,
            $avg_forward_pkt_dim = avg_forward,
            $avg_backward_pkt_dim = avg_backward]);
}
```

A.4 Algoritmo completo myScript.bro

Di seguito nella figura A.3 viene fornito il codice in linguaggio Bro dell'algoritmo myScript.bro, menzionato nel corso del capitolo 5 e che ci permette di creare un file di log con gli attributi che ci interessano per la classificazione, e che saranno poi mandati in input agli algoritmi di classificazione. Il codice è stato commentato nei punti salienti.

Listing A.3. Codice contenuto in myScript.bro.

```
% Appende il valore LOG all'enumerabile Log::ID. Serve per definire il file di log di
    uscita.
redef enum Log::ID += { LOG };

% Struttura che contiene le informazioni che verranno salvate nel file di uscita
    results.log, con il tipo e il fatto che possono essere scritte in un file di log.
type Info: record {
        uid : string &log;
        ip_source : addr &log;
        ip_dest : addr &log;
        port_source : count &log;
        port_dest : count &log;
        transport_prot : string &log;
        duration : double &log;
        forward_pkts_num : count &log;
        backward_pkts_num : count &log;
        forward_pkts_perc : double &log;
        backward_pkts_perc : double &log;
        tot_pkts : count &log;
        avg_dim_pkt : double &log;
        forward_avg_dim_pkt : double &log;
        backward_avg_dim_pkt : double &log;
        maximum_dim_pkt : count &log;
        minimum_dim_pkt : count &log;
        std_dev_dim_pkt : double &log;
        forward_bytes_num : count &log;
        backward_bytes_num : count &log;
        forward_bytes_perc : double &log;
        backward_bytes_perc : double &log;
        tot_bytes : count &log;
        tot_throughput_pkts : double &log;
        forward_throughput_pkts : double &log;
        backward_throughput_pkts : double &log;
        tot_throughput_bytes : double &log;
        forward_throughput_bytes : double &log;
        backward_throughput_bytes : double &log;
        maximum_inter_pkt : double &log;
        minimum_inter_pkt : double &log;
        avg_inter_pkt : double &log;
        std_dev_inter_pkt : double &log;
        asymmetric : string &log;
        symmetric : string &log;
        blocks : string &log;
        digest : string &log;
        name : string &log;
};

% Record per il connection ID.
type Idx: record {
```

```
        uid: string;
};

% Informazioni utili contenute in conn.log, con relativo tipo.
type Val1: record {
        id: conn_id;
        proto: string;
        duration: double;
        orig_bytes: count;
        resp_bytes: count;
        orig_pkts: count;
        resp_pkts: count;
};

% Informazioni utili contenute in ssl.log, con relativo tipo. A volte il campo subject
    oppure server name non sono presenti.
type Val2: record {
        server_name: string &optional;
        subject: string &optional;
        cipher: string;
};

% Informazioni utili contenute in statistics.log, con relativo tipo.
type Val3: record {
        maximum_inter_arrival : double;
        minimum_inter_arrival : double;
        avg_inter_arrival : double;
        std_dev_inter_arrival : double;
        avg_pkt_dim: double;
        avg_forward_pkt_dim : double;
        avg_backward_pkt_dim : double;
        minimum_pkt_dim : count;
        maximum_pkt_dim : count;
        std_dev_pkt_dim : double;
};

% Struttura che viene riempita a poco a poco e che contiene tutti gli attributi che
    verranno inseriti in results.log. Il nome e il numero degli attributi è lo stesso di
    quello che è stato definito sopra, ma sono dichiarati opzionali perché non a tutti
    viene dato un valore subito.
type MyRecord: record {
      uid : string &optional;
      ip_source : addr &optional;
      ip_dest : addr &optional;
      port_source : count &optional;
      port_dest : count &optional;
      transport_prot : string &optional;
      duration : double &optional;
      forward_pkts_num : count &optional;
      backward_pkts_num : count &optional;
      forward_pkts_perc : double &optional;
```

```
        backward_pkts_perc : double &optional;
        tot_pkts : count &optional;
        avg_dim_pkt : double &optional;
        forward_avg_dim_pkt : double &optional;
        backward_avg_dim_pkt : double &optional;
        maximum_dim_pkt : count &optional;
        minimum_dim_pkt : count &optional;
        std_dev_dim_pkt : double &optional;
        forward_bytes_num : count &optional;
        backward_bytes_num : count &optional;
        forward_bytes_perc : double &optional;
        backward_bytes_perc : double &optional;
        tot_bytes : count &optional;
        tot_throughput_pkts : double &optional;
        forward_throughput_pkts : double &optional;
        backward_throughput_pkts : double &optional;
        tot_throughput_bytes : double &optional;
        forward_throughput_bytes : double &optional;
        backward_throughput_bytes : double &optional;
        maximum_inter_pkt : double &optional;
        minimum_inter_pkt : double &optional;
        avg_inter_pkt : double &optional;
        std_dev_inter_pkt : double &optional;
        asymmetric : string &optional;
        symmetric : string &optional;
        blocks : string &optional;
        digest : string &optional;
        name : string &optional;
};

% Dichiarazione variabili globali
global mylist1: table[string] of Val1 = table();
global mylist2: table[string] of Val2 = table();
global mylist3: table[string] of Val3 = table();
global records : vector of MyRecord;

% Evento scatenato prima di ogni altro. Inizializza le tabelle coi file di log che già
    trova. Dichiara il file di log di uscita.
event bro_init() {
        Input::add_table([$source="conn.log", $name="conn",
                $idx=Idx, $val=Val1, $destination=mylist1]);
        Input::remove("conn");

        Input::add_table([$source="ssl.log", $name="ssl",
                $idx=Idx, $val=Val2, $destination=mylist2]);
        Input::remove("ssl");

        Input::add_table([$source="statistics.log", $name="statistics",
                $idx=Idx, $val=Val3, $destination=mylist3]);
        Input::remove("statistics");
```

```
        Log::create_stream(LOG, [$columns=Info, $path="results"]);
}

% Questo evento scatta quando uno dei file di log che viene letto arriva alla fine.
    Dato che è stato inizializzato prima conn.log, poi ssl.log e poi statistics.log,
    questo è anche l'ordine di fine dei files.

event Input::end_of_data(name: string, source: string) {
        % Parte che riguarda il file ssl.log. Esegue gli ultimi calcoli e poi salva in
            struttura principale.
        if(name == "conn")
        {
                local mystring1 : Val1;
                local tot_pkts : count;
                local tot_bytes : count;
                local tot_pkts_d : double;
                local tot_bytes_d : double;
                local throughput_pkts : double;
                local enter_pkts : double;
                local exit_pkts : double;
                local enter_bytes : double;
                local exit_bytes : double;
                local enter_throughput : double;
                local exit_throughput : double;
                local throughput_bytes : double;
                local enter_throughput_bytes : double;
                local exit_throughput_bytes : double;

                for(key in mylist1)
                {
                        mystring1 = mylist1[key];
                        tot_pkts = mystring1$orig_pkts + mystring1$resp_pkts;
                        tot_bytes = mystring1$orig_bytes + mystring1$resp_bytes;
                        tot_pkts_d = tot_pkts;
                        tot_bytes_d = tot_bytes;
                        enter_pkts = mystring1$resp_pkts / tot_pkts_d;
                        exit_pkts = mystring1$orig_pkts / tot_pkts_d;
                        enter_bytes = mystring1$resp_bytes / tot_bytes_d;
                        exit_bytes = mystring1$orig_bytes / tot_bytes_d;
                        throughput_pkts = tot_pkts_d / mystring1$duration;
                        enter_throughput = mystring1$resp_pkts /
                            mystring1$duration;
                        exit_throughput = mystring1$orig_pkts /
                            mystring1$duration;
                        throughput_bytes = tot_bytes_d / mystring1$duration;
                        enter_throughput_bytes = mystring1$resp_bytes /
                            mystring1$duration;
                        exit_throughput_bytes = mystring1$orig_bytes /
                            mystring1$duration;

                        records[|records|] = MyRecord($uid = key,
```

```
                                        $ip_source =
                                            mystring1$id$orig_h,
                                        $ip_dest =
                                            mystring1$id$resp_h,
                                        $port_source = port_to_count
                                        (mystring1$id$orig_p),
                                        $port_dest = port_to_count
                                        (mystring1$id$resp_p),
                                        $transport_prot =
                                            mystring1$proto,
                                        $duration =
                                            mystring1$duration,
                                        $forward_pkts_num =
                                            mystring1$orig_pkts,
                                        $backward_pkts_num =
                                            mystring1$resp_pkts,
                                        $backward_pkts_perc =
                                            enter_pkts,
                                        $forward_pkts_perc =
                                            exit_pkts,
                                        $tot_pkts = tot_pkts,
                                        $forward_bytes_num =
                                            mystring1$orig_bytes,
                                        $backward_bytes_num =
                                            mystring1$resp_bytes,
                                        $backward_bytes_perc =
                                            enter_bytes,
                                        $forward_bytes_perc =
                                            exit_bytes,
                                        $tot_bytes = tot_bytes,
                                        $tot_throughput_pkts =
                                            throughput_pkts,
                                        $backward_throughput_pkts
                                            = enter_throughput,
                                        $forward_throughput_pkts =
                                            exit_throughput,
                                        $tot_throughput_bytes =
                                            throughput_bytes,
                                        $backward_throughput_bytes
                                            =
                                            enter_throughput_bytes,
                                        $forward_throughput_bytes
                                            =
                                            exit_throughput_bytes);

            }
    }
    % Parte riguardante il file ssl.log. Spezza la cipher suite ed assegna ad ogni
        pezzo il suo valore. In base a quale informazione tra subject e server
        name è presente viene assegnata a name, dando precedenza a server
        name. Dopodiché continua a riempire la struttura principale.
```

```
if(name == "ssl")
{
        local mystring2 : Val2;
        local array : string_vec;
        local asymmetric : string;
        local symmetric : string;
        local blocks : string;
        local digest : string;
        local array2 : string_vec;
        for(key in mylist2)
        {
                mystring2 = mylist2[key];
                for(i in records)
                {
                        if(strcmp(records[i]$uid, key) == 0)
                        {
                                array = split_string(mystring2$cipher,/_/);
                                records[i]$asymmetric = "";
                                records[i]$symmetric = array[|array|-4] +
                                    "-" + array[|array|-3];
                                records[i]$blocks = array[|array|-2];
                                records[i]$digest = array[|array|-1];
                                for(j in array) {
                                        if(array[j] == "WITH")
                                                break;
                                        if(j > 0)
                                                records[i]$asymmetric =
                                                    records[i]$asymmetric
                                                    + "-" + array[j];
                                }
                                records[i]$asymmetric = sub_bytes
                                (records[i]$asymmetric,2,
                                |records[i]$asymmetric|);
                                if(!mystring2?$subject)
                                        records[i]$name =
                                            mystring2$server_name;
                                else {
                                        array2 = split_string
                                        (mystring2$subject,/,/);
                                        if(array[0][0] == "C" &&
                                            array[0][1] == "N")
                                                records[i]$name = sub
                                                (array2[0],/CN=/,"");
                                        else
                                                records[i]$name =
                                                    mystring2$server_name;
                                }
                        }
                }
        }
}
```

```
% Parte dedicata a statistics.log, file che abbiamo generato precedentemente
    con le statistiche mancanti. Dopodiché tutto viene salvato nella struttura
    principale.
    if(name == "statistics")
    {
            local mystring3 : Val3;

            for(key in mylist3)
            {
                    mystring3 = mylist3[key];
                    for(i in records)
                    {
                            if(strcmp(records[i]$uid, key) == 0)
                            {
                                    records[i]$maximum_inter_pkt =
                                        mystring3$maximum_inter;
                                    records[i]$minimum_inter_pkt =
                                        mystring3$minimum_inter;
                                    records[i]$avg_inter_pkt =
                                        mystring3$avg_inter;
                                    records[i]$std_dev_inter_pkt =
                                        mystring3$std_dev_inter;
                                    records[i]$avg_dim_pkt =
                                        mystring3$avg_pkt;
                                    records[i]$maximum_dim_pkt =
                                        mystring3$maximum_pkt;
                                    records[i]$minimum_dim_pkt =
                                        mystring3$minimum_pkt;
                                    records[i]$std_dev_dim_pkt =
                                        mystring3$std_dev_pkt;
                                    records[i]$forward_avg_dim_pkt =
                                        mystring3$avg_forward_pkt;
                                    records[i]$backward_avg_dim_pkt =
                                        mystring3$avg_backward_pkt;
                            }
                    }
            }

            % Se è entrato nella clausola if di statistics.log allora significa che è
                anche terminato il riempimento della struttura principale.
                Quindi ora posso salvare tutto nel file di log finale. Notare che
                non è presente l'evento bro_done() poiché verrebbe eseguito
                prima che la lettura dei files di log venisse portata a termine.

            for(i in records)
            {
                    Log::write(LOG,[$uid = records[i]$uid,
                                    $ip_source = records[i]$ip_source,
                                    $ip_dest = records[i]$ip_dest,
                                    $port_source = records[i]$port_source,
                                    $port_dest = records[i]$port_dest,
```

```
$transport_prot = records[i]$transport_prot,
$duration = records[i]$duration,
$forward_pkts_num =
    records[i]$forward_pkts_num,
$backward_pkts_num =
    records[i]$backward_pkts_num,
$backward_pkts_perc =
    records[i]$backward_pkts_perc,
$forward_pkts_perc =
    records[i]$forward_pkts_perc,
$tot_pkts = records[i]$tot_pkts,
$avg_dim_pkt = records[i]$avg_dim_pkt,
$forward_avg_dim_pkt =
    records[i]$forward_avg_dim_pkt,
$backward_avg_dim_pkt =
    records[i]$backward_avg_dim_pkt,
$maximum_dim_pkt =
    records[i]$maximum_dim_pkt,
$minimum_dim_pkt =
    records[i]$minimum_dim_pkt,
$std_dev_dim_pkt =
    records[i]$std_dev_dim_pkt,
$forward_bytes_num =
    records[i]$forward_bytes_num,
$backward_bytes_num =
    records[i]$backward_bytes_num,
$backward_bytes_perc =
    records[i]$backward_bytes_perc,
$forward_bytes_perc =
    records[i]$forward_bytes_perc,
$tot_bytes = records[i]$tot_bytes,
$tot_throughput_pkts =
    records[i]$tot_throughput_pkts,
$backward_throughput_pkts =
    records[i]$backward_throughput_pkts,
$forward_throughput_pkts =
    records[i]$forward_throughput_pkts,
$tot_throughput_bytes =
    records[i]$tot_throughput_bytes,
$backward_throughput_bytes =
    records[i]$backward_throughput_bytes,
$forward_throughput_bytes =
    records[i]$forward_throughput_bytes,
$maximum_inter_pkt =
    records[i]$maximum_inter_pkt,
$minimum_inter_pkt =
    records[i]$minimum_inter_pkt,
$avg_inter_pkt = records[i]$avg_inter_pkt,
$std_dev_inter_pkt =
    records[i]$std_dev_inter_pkt,
$asymmetric = records[i]$asymmetric,
```

```
                                        $symmetric = records[i]$symmetric,
                                        $blocks = records[i]$blocks,
                                        $digest = records[i]$digest,
                                        $name = records[i]$name]);
                }
        }
}
```

A.5 Algoritmo parziale myScriptSftp.bro

Di seguito nella figura A.4 viene fornito il codice in linguaggio Bro (con i relativi commenti) che è stato utilizzato al posto delle parti riguardanti i parametri del file ssl.log in myScript.bro. Si vede che è stata operata una sostituzione con i parametri del file ssh.log.

Listing A.4. Codice di differenza contenuto in myScriptSftp.bro.

```
% Informazioni utili contenute in ssh.log, e relativo tipo.
type Val2: record {
        server: string;
        cipher_alg: string;
        mac_alg: string;
        host_key_alg: string;
};

% Apertura del file ssh.log.
Input::add_table([$source="ssh.log", $name="ssh",
                $idx=Idx, $val=Val2, $destination=mylist2]);
        Input::remove("ssh");

% Parte dedicata a ssh.log. Dopodiché tutto viene salvato nella struttura principale.

if(name == "ssh")
{
        local mystring2 : Val2;
        local array1 : string_vec;
        local array2 : string_vec;
        local array3 : string_vec;
        local array4 : string_vec;
        local array5 : string_vec;
        local asymmetric : string;
        local symmetric : string;
        local blocks : string;
        local digest : string;
        for(key in mylist2)
        {
                mystring2 = mylist2[key];
                for(i in records)
                {
```

```
                        if(strcmp(records[i]$uid, key) == 0)
                        {
                                array1 = split_string(mystring2$host_key_alg,/-/);
                                records[i]$asymmetric = to_upper(array1[1]);
                                array2 = split_string(mystring2$cipher_alg,/@/);
                                array3 = split_string(array2[0],/-/);
                                records[i]$symmetric = to_upper(array3[0]);
                                records[i]$blocks = "none";
                                array4 = split_string(mystring2$mac_alg,/@/);
                                array5 = split_string(array4[0],/-/);
                                array5[1] = sub(array5[1],/2/,"");
                                records[i]$digest = to_upper(array5[1] + "-" +
                                    array5[2]);
                                records[i]$name = mystring2$server;
                        }
                }
        }
}
```

A.6 Algoritmo script myBash.sh

Nella figura A.5 il file myBash.sh, che indica il modus operandi della generazione dei risultati, opportunamente commentato:

Listing A.5. Codice contenuto nello script myBash.sh.

```
#!/bin/bash

# Compila i due programmi scritti in C.
gcc -o myProgram myProgram.c
gcc -o myMerge myMerge.c

# Elimina gli eventuali vecchi risultati, crea la directory per ospitarne di nuovi e si
    sposta nella directory contenente tutti i files .pcap.
sudo rm -rf ./results
mkdir results
cd ./pcaps
j=1

# Per tutti i direttori, cioè per ogni classe di traffico.
for directory in */
do

        # Prende solo il nome del direttorio e stampa un messaggio di debug.
        cd ./$directory
        directory=${directory:0:${#directory}-1}
        echo ""
        echo "Processing_$directory_directory..."
        echo ""
```

```
        i=0

        # Per ogni file presente nel direttorio corrente.
        for filename in ./*.pcap
        do

                # Calcola la percentuale di lavoro effettuato e stampa un
                    messaggio di debug.
                n_files=$(ls -1 *.pcap 2>/dev/null | wc -l)
                percent=$(awk "BEGIN {printf \"%.0f\",${i}/${n_files}*100}")
                new=${filename:2:${#filename}-7}
                echo "[$percent%] Processing $new file..."

                # Processa il file .pcap aggiungendo lo script myStatistics.bro,
                    chiamato in local.bro.
                sudo /usr/local/bro/bin/bro -r $filename -C
                    /usr/local/bro/share/bro/site/local.bro

                # Nel caso io sia nella directory sftp significa che devo includere il
                    file myScriptSftp.bro chiamato nel file local2.bro, in caso
                    contrario includo myScript.bro chiamato in local1.bro, quindi
                    eseguo Bro con questi nuovi files.
                if [ "$directory" == "sftp" ]
                then
                        sudo /usr/local/bro/bin/bro
                            /usr/local/bro/share/bro/site/local2.bro
                else
                        sudo /usr/local/bro/bin/bro
                            /usr/local/bro/share/bro/site/local1.bro
                fi

                # Eseguo il codice del primo algoritmo in C con il file results.log
                    come parametro. Lo rinomino e lo sposto nella directory dei
                    risultati.
                ./../../myProgram results.log $j
                newres="results_$new.txt"
                mv results.txt $newres
                mv $newres ./../../results

                # Rimuovo i file di log appena creati per far posto a quelli nuovi e
                    non creare sovrapposizioni col prossimo file .pcap.
                sudo rm *.log
                sudo rm -rf .state/
                ((i++))
        done
        ((j++))
        cd ./..
done
cd ./..
```

```
# Eseguo il codice del secondo algoritmo in C dopo aver fatto un merge di tutti i
    risultati.
cp myMerge ./results
cd ./results
cat results* > merged-file.txt
./myMerge merged-file.txt

# Rimuovo tutto e lascio solo il file generato e nominato merged.txt.
rm myMerge
rm merged-file.txt
rm results*
```

Una volta ottenuto il file finale posso ora sottometterlo in input agli algoritmi di classificazione. Da notare il fatto che nell'algoritmo bash è stata utilizzata la variabile j come identificativo della classe: infatti, dato che come vedremo gli algoritmi di classificazione usati accettano come etichetta un numero intero, è stato deciso di identificare le classi con numeri interi da 0 a 7 per le 8 classi coinvolte. La variabile i, invece, è solo utilizzata per questioni di debug, e stampare a video l'avanzamento dell'algoritmo in percentuale di completamento.

A.7 Algoritmo Random Forest

Viene presentato in figura A.6 il codice commentato del Random Forest in linguaggio Python.

Listing A.6. Codice contenuto nello script random_forest.py.

```
# Importa la libreria numpy per convertire in array.
import numpy as np

# Importa il modello che stiamo utilizzando.
from sklearn.ensemble import RandomForestClassifier

# Importa la libreria per la matrice di confusione.
from sklearn.metrics import confusion_matrix

# Importa la libreria per la k-fold cross validation.
from sklearn.model_selection import RepeatedKFold

# Importa il report di classificazione.
from sklearn.metrics import classification_report

# Importa pandas per la manipolazione dei dati.
import pandas as pd

# Legge i dati in input
features = pd.read_csv('merged.txt',sep=',')
```

```
# Converte le features non numeriche in features numeriche, aggiungendo colonne e
    impostando i rispettivi zeri e uni.
features = pd.get_dummies(features)

# L'attributo con nome label è quello che vogliamo predire.
labels = np.array(features['label'])

# Rimuove l'etichetta dalle features, dove axis 1 si riferisce alle colonne.
features= features.drop('label', axis = 1)

# Salva la lista di features per un successivo utilizzo.
feature_list = list(features.columns)

# Converte le features in array.
features = np.array(features)

# Viene istanziato il modello con 1000 decision trees e viene definito il k-fold cross
    validator e i suoi valori.
classifier = RandomForestClassifier(n_estimators = 1000, random_state = 42)
kf = RepeatedKFold(n_splits=10, n_repeats=1, random_state=None)

# Viene definita una funzione che calcola l'accuratezza tramite la matrice di
    confusione.
def accuracy(confusion_matrix):
    diagonal_sum = confusion_matrix.trace()
    sum_of_all_elements = confusion_matrix.sum()
    return diagonal_sum / sum_of_all_elements

# Implementazione dell'algoritmo per ogni split definito nel cross validator.
for train_index, test_index in kf.split(features):
      X_train, X_test = features[train_index], features[test_index]
      y_train, y_test = labels[train_index], labels[test_index]

      # Allenamento del classificatore sul training set.
      classifier.fit(X_train, y_train)

      # Viene usato il classificatore per predire le etichette sul validation set.
      predictions = classifier.predict(X_test)

      # Arrotondamento delle etichette.
      for i, val in enumerate(predictions):
          predictions[i] = int(round(predictions[i]))

      # Costruzione della matrice di confusione e del report.
      cm_test = confusion_matrix(y_test, predictions)
      report = classification_report(y_test, predictions)

      # Stampa dei risultati. Prima l'accuratezza e poi tutti gli altri parametri di
          valutazione.
      print ('Accuracy:', 100*accuracy(cm_test), '%.')
      print (report)
```

```
# Ricezione della tabella con l'importanza delle features e inserimento in una lista.
importances = list(classifier.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, importance in
    zip(feature_list, importances)]

# Ordinamento delle features per importanza.
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse
    = True)

# Stampa del risultato feature-importanza con numeri compresi tra 0 e 1.
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in
    feature_importances];
```

Bibliografia

[1] A. Lioy, "Sicurezza delle applicazioni di rete." http://security.polito.it/~lioy/03gsd/remote_2x.pdf

[2] The Bro Network Security Monitor, https://www.bro.org/

[3] A. Pumphrey, "A bro primer." https://www.bro.org/brocon2017/slides/bro_primer.pdf

[4] R. Sommer, "Following the packets: A walk through bro's internal processing pipeline." https://www.bro.org/brocon2017/slides/bro_internals.pdf

[5] P. Mehta and R. Shah, "A survey of network based traffic classification methods." http://www.dbjournal.ro/archive/26/26_3.pdf

[6] J. Cao, Z. Fang, G. Qu, and H. Sun, "An accurate traffic classification model based on support vector machines", International Journal of Network Management, vol. 27, 1 2017, p. 1962, DOI 10.1002/nem.1962

[7] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications", Lecture Notes in Computer Science, pp. 165–175, DOI 10.1007/978-3-540-71617-4_17

[8] W. Pan, G. Cheng, and Y. Tang, "Wenc: Https encrypted traffic classification usingweighted ensemble learning and markov chain", 2017 IEEE Trustcom/-BigDataSE/ICESS, 8 2017, DOI 10.1109/trustcom/bigdatase/icess.2017.219

[9] E. Hjelmvik, "The spid algorithm statistical protocol identification." https://www.iis.se/docs/The_SPID_Algorithm_-_Statistical_Protocol_IDentification.pdf

[10] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark", Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '05, 2005, DOI 10.1145/1080091.1080119

[11] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification:identifying ssh and skype", 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 7 2009, DOI 10.1109/cisda.2009.5356534

[12] Y. Zion, J. Muehlstein, M. Bahumi, and I. Kirshenboim, "Analyzing https traffic for a robust identification of operating system, browser and application", 2017 14th IEEE Annual Consumer Communications and Networking Conference (CCNC), 1 2017, DOI 10.1109/ccnc.2017.8013420

[13] J. Zhang, X. Chen, Y. Xiang, and W. Zhou, "Robust network traffic classification", IEEE/ACM Transactions on Networking, vol. 23, 8 2015, pp. 2653–2655, DOI 10.1109/tnet.2014.2320577

[14] A. Rodolico, "Classificazione deltraffico cifrato (https) ed in particolare del traffico delle app per dispositivi mobili." https://docplayer.it/

[15] C. Richter, M. Finsterbusch, K. Hanbgen, and J. Muller, "Classification of tls applications." https://www.thinkmind.org/download.php?articleid=icimp_2014_1_10_30006

[16] W. Shbair, T. Cholez, J. Francois, and I. Chrisment, "Https traffic classification", 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 5 2015, DOI 10.1109/inm.2015.7140423

[17] N. Donges, "The random forest algorithm." https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

[18] S. Ray, "Understanding support vector machine." https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[19] A. Bronshtein, "A quick introduction to k-nearest neighbors algorithm." https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

[20] S. Ray, "6 easy steps to learn naive bayes algorithm." https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[21] A. Kumar, "Data science - how to scale or normalize numeric data." https://vitalflux.com/data-science-scale-normalize-numeric-data-using-r/

[22] S. Ray, "Improve your model performance using cross validation." https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/

[23] Accuracy, Precision, Recall and F1 Score: Interpretation of Performance Measures, http://blog.exsilio.com/all/

[24] Random Forest Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html