Università di Pisa

Dipartimento di Informatica

Laurea triennale in Informatica

Rilevazione di anomalie nel traffico di rete senza l'utilizzo di tecniche di ML

Relatore
Luca Deri

Candidato
Simone Ricci

Anno Accademico
2018/2019

Sommario

Negli ultimi anni si è diffuso, nell'ambito delle tecnologie informatiche e non solo, l'utilizzo di numerose e diverse tecniche di machine learning per riconoscere, classificare e, in alcuni casi, riprodurre fenomeni esterni. Uno degli ambiti informatici che più ha fatto uso di questa tecnologia sin dai suoi albori è quello della sicurezza informatica e dell'analisi del traffico di rete. In entrambi i casi si è cercato di sviluppare software che fosse in grado di riconoscere un comportamento anomalo di una macchina, o un'anomalia nel traffico di rete, e di segnalarlo, ottenendo, senza dubbio, ottimi risultati, ma a discapito di efficienza e aumentando notevolmente la complessità dell'architettura e i tempi necessari a sviluppare, allenare e testare il software.

Con questo lavoro si vuole dimostrare che le tecniche di ML non sono l'unica soluzione per rilevare anomalie nel traffico di rete, ma si possono ottenere risultati simili anche senza di esse e che, seppur senza la stessa accuratezza, ci portano a rilevare l'host che è causa di tali anomalie, con un minor dispendio di risorse.

Indice

1. Introduzione

Oramai le comunicazioni via internet stanno andando verso la direzione di nascondere completamente il contenuto dei messaggi in modo che sia visibile soltanto al mittente e al destinatario. Se da un lato, questa strada intrapresa è un punto a favore per la privacy di un servizio e di chi ne usufruisce, dall'altro rappresenta un ostacolo per chi ha l'intento di leggere il contenuto di uno o più pacchetti non per scopi malevoli, ma per assicurarsi che quei pacchetti non possano causare danni al destinatario.

Per aggirare tale ostacolo sono stati sviluppati strumenti che si occupano dell'analisi del traffico di rete, senza andare a leggere il contenuto del pacchetto, in grado di raccogliere statistiche su di esso (ad esempio la banda occupata da un determinato host, il numero di connessioni aperte per host, la durata di una connessione ecc…) e possono limitarsi a mostrare tali statistiche lasciando poi all'utente il compito di analizzarle e trarne le dovute conclusioni.

Dopo la rapida diffusione avuta dalle tecniche di machine learning nell'ultimo decennio, si è iniziato ad utilizzare le nuove tecnologie per dar vita a strumenti in grado di saper distinguere il traffico malevolo dal traffico normale in modo da poterlo segnalare (si parla dunque di "Intrusion Detection System", IDS) o anche bloccare tale tipo di traffico (e allora si parla di "Intrusion Prevenction System", IPS). Tale evoluzione ha portato sì ottimi risultati, ma l'utilizzo di queste tecniche richiede grandi quantità di tempo e risorse hardware all'altezza del compito da svolgere.

In questo lavoro di tirocinio viene proposta una metodologia per la rilevazione di anomalie nel traffico di rete, che non fa uso di alcuna tecnica di machine learning e che può essere utilizzata anche su sistemi poco performanti.

1.1 Motivazioni e obiettivi

La motivazione principale alla base di questo lavoro è stata il voler dimostrare che il machine learning non è l'unica strada per raggiungere certi risultati, ma che, come si vedrà nel corso di questo elaborato, scendendo a compromesso con una, seppur poco, minore efficacia si possono avere prestazioni molto più elevate. E con il termine prestazioni non ci si riferisce soltanto all'efficienza con cui l'anomalia viene individuata, ma anche ai tempi necessari per lo sviluppo, per la fase di training (che senza il machine learning viene completamente abolito) e alla minor necessità di dover disporre hardware di prima qualità, elemento assai importante quando si parla di machine learning. L'obiettivo principale di tale lavoro, e più in generale di un Intrusion Detection System, non è quello di individuare esattamente, uno per uno, tutti i flussi sui quali è in corso un attacco, ma quello di individuare una metodologia che ci permetta, analizzando il traffico di rete, di evidenziare un host malevolo o che comunque ha un comportamento anomalo.

1.2 Struttura della relazione

- Nella sezione 2 verrà descritto il dataset utilizzato per portare a termine il lavoro di tirocinio
- Nella sezione 3 verrà trattato lo stato dell'arte, descrivendo alcuni strumenti esistenti che utilizzano tecniche differenti per la rilevazione di anomalie nel traffico di rete.
- Nel capitolo 4 verranno illustrate le soluzione proposte per il raggiungimento dell'obiettivo, descrivendo gli strumenti utilizzati e analizzando sotto ogni aspetto la metodologia proposta. I risultati ottenuti verranno poi discussi nel capitolo successivo e paragonati con altri software.
- Nei capitoli 6 e 7 verranno discussi le conclusioni raggiunte dopo tale lavoro di tirocinio e i lavori con cui sarebbe possibili estendere la metodologia proposta

2. Dataset

Il dataset su cui è stato condotto il lavoro di tirocinio è il *CICIDS2017[1]*, costruito dal CIC (Candian Instituite of Cybersecurity), che fornisce cinque file PCAPs annotati, ovvero accompagnati da una descrizione di cosa è successo in ogni intervallo temporale, oltre alla descrizione della topologia della rete[2].

2.1 Topologia di rete

Il dataset del CIC fornisce una configurazione di rete completa in cui è possibile trovare switches, modem, router, firewall e diversi host interni alla rete con differenti OS, quali Ubuntu, Windows e Mac OS X. Inoltre sono presenti anche tre host esterni alla rete che costituiscono gli attacanti[3].

	Machine	OS	IPs
Victim-Network	Servers	Win Server 2016 (DC and DNS) Ubuntu 16 (Web Server) Ubuntu 12	192.168.10.3 192.168.10.50-205.174.165.68 192.168.10.51-205.174.165.66
	PCs	Ubuntu 14.4 (32, 64) Ubuntu 16.4 (32-64) Win 7Pro Win 8.1-64 Win Vista Win 10 (Pro 32-64) Mac	192.168.10.19-192168.10.17 192.168.10.16-192.168.10.12 192.168.10.9 192.168.10.5 192.168.10.8 192.168.10.14-192.168.10.15 192.168.10.25
	Firewall	Fortinet	
Attackers	PCs	Kali win 8.1 Win 8.1 Win 8.1	205.174.165.73 205.174.165.69 205.174.165.70 205.174.165.71

Immagine 1: Mappa della rete su cui è costruito il dataset CICIDS2017[3]

Come si osserva nella Tabella 1, la rete è costituita da tre macchine server (Win server 2016, Ubuntu 16 e Ubuntu 12) con i relativi indirizzi IP interno ed esterno.

Oltre ai servers sopra descritti, troviamo undici hosts con diversi OS e i relativi indirizzi IP.

Il CIC fornisce anche una descrizione degli hosts attaccanti.

2.2 Tipo di traffico

Il dataset CICIDS2017 include una vasta varietà di traffico includendo i protocolli più comuni come FTP, SSH, HTTP e i protocolli e-mail. Inoltre è possibile rilevare una completa interazione tra tutti gli hosts della rete[2].
Piccola pecca del dataset è quella di non includere traffico cifrato: è rilevante l'assenza quasi totale del protocollo HTTPS, senza dubbio il protocollo più usato sul web. Tale mancanza allontana notevolmente il traffico presente nel CICIDS2017 dal traffico reale. Nonostante ciò, l'assenza di traffico cifrato non ha presentato alcuna agevolazione, né tanto meno un ostacolo, nello svolgere il lavoro. Nell'analizzare il traffico, non si è mai andati a guardare il payload del pacchetto, fermandomi al livello trasporto, salvo alcune eccezioni che verranno motivate in seguito.
L'obiettivo di questo dataset, al momento della sua costruzione, era quello di emulare nel miglior modo possibile il traffico benigno: questo obiettivo è stato raggiunto mediante un agent sviluppato in Java che simula il comportamento di 25 utenti, non malevoli, della rete[3].
Oltre al traffico benigno sono stati inseriti degli attacchi di vario tipo (attacchi Dos, brute force, port scan e injection), utilizzando degli appositi tool o scrivendo codice in Python[3].
Il risultato ottenuto è un insieme di cinque PCAPs il cui periodo di cattura inizia il giorno Lunedì 03/07/2017 e termina il giorno Venerdì 07/07/2017, dalle ore 09:00 alle ore 17:00 (GMT-4) di ogni giorno[2].

2.3 Descrizione degli attacchi

Il dataset CICIDS2017 comprende sia traffico benigno che traffico malevolo. Ogni file PCAP che compone il dataset è annotato: viene specificato cosa accade e che tipo di

traffico è rilevabile in ogni intervallo di tempo. In questa sezione viene elencato e descritto ogni attacco presente nel dataset.

La prima tipologia di attacchi presenti nel dataset è quella deglli attacchi DoS (Denial of Service). L'obiettivo di questo tipo di attacchi è quello di, come si evince dal nome, impedire ad un server di svolgere il suo servizio[4]. La strategia tipica di un attacco DoS è quella di aprire numerose connessioni verso il server designato come vittima e mantenerle aperte finché l'host remoto non sarà più in grado di aprirne nuove, negando così il servizio agli utenti benigni. Di attacchi DoS ne esistono diverse tipologie. Qui sotto vengono brevemente descritte quelle presenti nel dataset.

- **DoS Goldeneye**: è uno dei più diffusi attacchi DoS e prende il nome dall'omonimo tool *goldeneye*, il quale cerca di riempire il server aprendo numerose connessioni HTTP, verso un host vittima, specificando l'opzione "*keep alive*" facendo s' che vangano lasciate aperte per lunghi periodi. Quando il server avrà troppe connessione aperte, sarà impossibilitato ad aprirne nuove, negando così il servizio agli utenti benigni.[5]
- **DoS Hulk**: anche in questo caso si tratta di un tipico (anche se un po' datato) attacco dos che prende il nome dal tool *hulk*. Anche in questo caso lo scopo dell'attacco è quello di fare in modo che il server non possa rispondere alle richieste di altri utenti, ma stavolta il mezzo per raggiungere l'obiettivo non si basa sul numero di connessioni aperte, bensì sul tipo di richieste. Il tool hulk infatti invia molte richieste sulla stessa connessione (che può essere anche più di una, ma pur sempre in numero limitato), modificando ogni volta i parametri richiesti in modo che il server non possa fare affidamento sui meccanismi di caching e sia costretto a processare ogni richiesta.[5]
- **DoS slow-HTTP** e **slowloris**: sono attacchi facenti parte della categoria slow DoSs, in quanto agiscono in modo lento e difficile da individuare, consumando pochissima banda e risorse hardware, rendendoli eseguibili anche su dispositivi non molto performanti. Il primo apre numerose connessioni verso il server vittima e punta a

mantenerle aperte inviando su di esse richieste HTTP incomplete facendo sì che il server attenda che la richiesta venga completata prima di chiudere la connessione. Lo slowloris è un tipo DoS slow-HTTP la cui particolarità è quella di effettuare richieste HTTP che hanno un header HTTP incompleto. Come i precedenti, anch'esso prende il nome dall'omonimo tool. Data la loro somiglianza, d'ora in poi, verrà utilizzata il nome **DoS slow** per riferirsi ad entrambi indifferentemente.[5]

- **DDoS**: non è un vero e proprio tipo di attacco DoS ma può essere effettuato una qualunque delle tipologie di attacco DoS sopra descritte. La differenza è che, come dice il nome (*Distributed Denial of Service*), viene eseguito in modo distribuito, ovvero controllando più macchine che contribuiscono a bloccare il servizio offerto da una singola macchina remota.[5]

La seconda tipologia di attacchi presenti nel dataset CICIDS2017 è quella degli attacchi forza bruta. Essi vengono eseguiti mediante il tool *patator[6]*, in grado di forzare username e password per 30 protocolli tra i quali DNS, SQL queries, POP, Telnet, SMTP, FTP e SSH. Il tool viene utilizzato nel dataset soltanto sui protocolli FTP e SSH. In entrambi i casi il suo funzionamento consiste nel iterare su una lista di usernames e passwords (o, in alternativa fissare uno dei due e iterare soltanto sull'altro), costruire l'header adeguato per il protocollo che si desidera forzare ed effettuare la richiesta.[7]

Un particolare tipo di attacco, molto diffuso e pericoloso fino a pochi anni fa, presente nel dataset è l'attacco detto **heartbleed**. Si tratta di un attacco che sfrutta un bug presente nella libreria OpenSSL fino al 2014, anno in cui il bug venne rimosso. Per capire come tale malfunzionamento veniva sfruttato, è necessario prima comprendere come funziona l'*heartbeet* nella libreria OpenSSL. Quando due host sono connessi tra loro e, durante la comunicazione, i dati scambiati vengono cifrati mediante TLS utilizzando la suddetta libreria possono testare la connessione inviandosi dei messaggi di *Heartbeet Request*: l'host che decide di testare la connessione invia un messaggio contenente una stringa e la sua lunghezza. Il destinatario deve leggere tale messaggio e rispedirlo al mittente senza

modificarlo. Il bug, che ha dato vita all'attacco hearbleed, è un caso tipo di *buffer overflow[8]*: il primo host invia una stringa, di lunghezza *l*, specificando una lunghezza fasulla *k*>n. Il destinatario dell'Heartbeet Request, non controlla la reale lunghezza della stringa e spedisce al mittente un buffer di dimensione k, i cui primi n bytes contengono la stringa ricevuta e i successivi k-n bytes contengono i valori presenti in memoria al momento dell'allocazione del buffer, che possono riguardare informazioni sensibili relative ad altri client o al server stesso.[9]

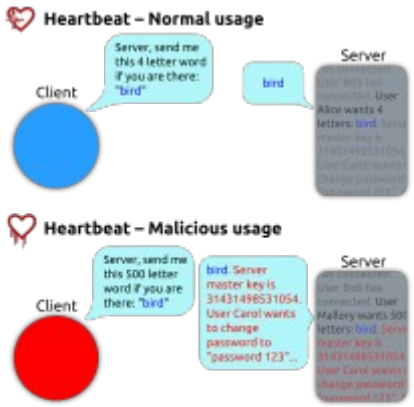

Immagine 2: Il funzionamento normale (sopra) e maligno (sotto) dell'Heartbeet Request

Come si osserva nell'immagine accanto, nel caso di una Heartbeet Request malevola, l'utente invia al server la stringa "*bird*" di lunghezza 4, dichiarando una lunghezza di 500. Il server, che non controlla la reale lunghezza della stringa ricevuta, spedisce al client un buffer di dimensione 500, i cui primi 4 bytes contengono la stringa "bird" e i successivi 496 bytes contengono i valori presenti in memoria al momento dell'allocazione del buffer.

Un'altra tipologia di attacco che possiamo trovare in questo dataset è quella dei WebAttack. Precisamente ne possiamo trovare tre tipi.

- **Web Attack – Brute Force**: Non molto diverso da quanto fatto dal tool patator, ovvero provare insistentemente combinazioni username-password fino a trovarne una corretta. La differenza risiede nel campo di azione. Mentre patator è stato sviluppato per forzare diversi protocolli, quest'attacco mira ad indovinare la coppia username-password per un form di login di una qualunque pagina web.[10]
- **Web Attack – SQL Injection**: si tratta di uno degli attcchi più diffusi, più facili da effettuare e più difficili da contrastare. Consiste nell'iniettare tra i parametri di una

query SQL, eseguita verso un server remoto, una stringa SQL in modo tale che, una volta che il server avrà terminato di processare i veri parametri della query, inizierà a processare la stringa malevola eseguendo la query passata come argomento. È dunque sufficiente conoscere la struttura del database per poter formulare una query corretta ed ottenere il risultato desiderato che può essere semplicemente quello di ottenere dei dati relativi ad altri utenti o anche quello di cancellare il contenuto dell'intero DB.[11]

- **Web Attack – XSS**: il *cross-site scripting* non differisce molto dall'SQL injection. Consiste nell'inserire in una URL passata al server dei frammenti di codice javascript (o di qualunque altro linguaggio di programmazione, purché sia lo stesso in cui è stato scritto il sito web), che verranno eseguiti sul server non appena verrà effettuato il parsing della URL. È facile capire come, mediante questo attacco, si possa far eseguire ad un server web un qualunque comando javascript con qualunque tipo di effetto.[12]

Le ultimi tipologie di attacchi presenti nel dataset sono attacchi più conosciuti anche se la varietà di modi in cui possono essere effettuati li rende assai ostici da individuare.

- **Infiltration**: consiste nell'infiltrarsi, appunto, in un host remoto, spesso mediante un malware in grado di prenderne il controllo.
- **Botnet**: molto simile, concettualmente, all'attacco infiltration. La differenza sta nel numero: mentre il primo consiste nel controllare un solo host, quest'ultimo mira a prendere il controllo di un'intera rete (o di un suo sottoinsieme sufficientemente vasto) in modo da poter eseguire un altro tipo di attacco aumentando la disponibilità di risorse in dote all'attaccante e rendendo intracciabile la (vera) sorgente dell'attacco.[13]
- **PortScan**: in realtà non è un vero e proprio attacco, ma si tratta di un'azione che mira a conoscere le porte, su cui è aperto un servizio, su un host remoto. Per effettuare questo attacco si possono utilizzare dei tool (il più diffuso è *nmap[14]*), il

cui unico compito è quello di tentare di aprire connessioni verso l'host specificato iterando su tutte le porte. Se la connessione viene aperta con successo, allora su quella determinata porta è aperto un servizio. [15]

2.4 Traffico benigno

Oltre agli attacchi descritti nel paragrafo precedente, nel dataset è stato inserito anche un PCAP, corrispondente ad un'intera giornata di lavoro, di traffico benigno, che non contiene alcun tipo di attacco e che si è rilevato utile per confrontare situazioni di traffico malevolo con situazioni in cui il traffico è del tutto "pulito".

3. Stato dell’arte

Data l’estrema diffusione che stanno avendo questi attacchi (e non solo) nel mondo del web, negli ultimi anni sono nati diversi IDS o IPS che utilizzano le più svariate tecniche per individuare (e, nel caso degli IPS, fermare) diversi tipi di attacchi. Non solo tool, ideati per essere venduti o distribuiti gratuitamente, ma anche numerose ricerche sono state effettuate nel campo. In entrambi i casi si è vista una rapida diffusione dell’utilizzo di tecniche di machine learning. Lo stesso detaset descritto nel capitolo 2 è stato più volte usato come training-set o validation-set per progetti che avevano lo scopo di individuare anomalie nel traffico di rete mediante algoritmi di machine learning.
In questa sezione verranno analizzati dei lavori basati sul dataset CICIDS2017, oltre ntopng che si è rivelato utile in questo lavoro per l’individuazione di alcuni attacchi.

3.1 CICFlowMeter

Il primo strumento che verrà presentato è il CICFlowMeter. Non è un vero e proprio IDS, ma un semplice tool che si limita a sniffare il traffico di rete e a raccogliere i flussi, calcolando numerose metriche per ognuno di essi. Il motivo per cui verrà comunque analizzato, nonostante la sua natura non-IDS, sta nel fatto che molti progetti basati sul CICIDS2017, compreso quello protagonista di questo lavoro di tirocinio, si rifanno a CICFlowMeter sotto alcuni aspetti, in particolare nella decisione di dividere il traffico per flussi e nelle metriche calcolate.

3.1.1 Cos’è CICFlowMeter?

CICFlowMeter è un tool di analisi di rete, scritto in Java, sviluppato presso il Canadian Institute of Cybersecurity. Il suo scopo non è quello di individuare anomalie nel traffico di rete, né tanto meno allertare l’utente o provare a fermarle. Le sue funzionalità si limitano a sniffare il traffico di rete, raccogliere statistiche e, al termine dell’esecuzione, produrre un

file in formato CSV, che può essere facilmente letto da occhi umani e allo stesso tempo può diventare input di un secondo programma che lo analizza e ne estrae informazioni di qualunque tipo.[16]

3.1.2 Divisione del traffico in flussi

La caratteristica principale introdotta dal CIC e, successivamente, adottata dalla maggior parte dei progetti ispirati ad esso è la divisione del traffico in flussi. Un flusso rappresenta una comunicazione tra due applicazioni, eseguite su host diversi e che comunicano tramite protocollo di rete IP. Un flusso è identificato da una quintupla, composta da indirizzo IP sorgente, numero di porta sorgente, indirizzo IP destinatario, numero di porta destinatario e protocollo di trasporto utilizzato. Questa quintupla sarebbe sufficiente ad identificare in modo univoco (basti pensare che un host non può usare la stessa porta per due comunicazioni distinte), ma nel caso del CICFlowMeter (e non solo) si è scelto di aggiungere un ulteriore attributo univoco, il flowid: un intero che viene assegnato, in ordine crescente, al flusso nel momento in cui se ne crea l'istanza, in modo da avere un singolo attributo che permetta di identificare il flusso senza ricorrere alla quintupla sopra descritta.

Inoltre, nel momento in cui il flusso viene istanziato, si stabilisce anche la direzione di *forwarding* (fwd) e di *backwarding* (bwd). Supponiamo che l'host A inizi una comunicazione con l'host B, inviando ad esso un pacchetto. In questo caso si parlerà di "*direzione di forwarding*" per tutti i pacchetti successivi inviati da A a B, mentre si parlerà di "*direzione di backwarding*" per tutti i pacchetti che compiono il percorso inverso, ovvero da B ad A. In altri progetti ci si riferisce a queste due direzioni con altri termini. Ad esempio il termine "forwarding" può essere sostituito da "client to server", supponendo che l'host che inizia una comunicazione sia un cliente che si connette ad un server, e, allo stesso modo, "backwarding" viene rimpiazzato con "server to client". Indipendentemente dal nome usato, la distinzione tra le due direzione permette, come vedremo nel paragrafo successivo, di differenziare il calcolo di alcune metriche per le due direzioni, oltre che

mantenere un calcolo per l'intero flusso. Un flusso è considerato terminato in due distinti casi:

1) se si tratta di un flusso TCP allora sarà terminato nel momento in cui avviene in two-way-handshake finale o, semplicemente, uno dei due host invia un pacchetto con flag RST settato
2) se si tratta di un flusso UDP allora viene terminato da un *flow-timeout*, impostato di default a 600 secondi, ma che può essere settato tramite un opzione a riga di comando.

Oltre che nello stato di "terminato" e nello stato "*attivo*" un flusso può andare anche nello stato di "*idle*", ovvero quando su un flusso non transitano pacchetti da un tempo maggiore del *flow-timeout* e il flusso viene, temporaneamente rimosso dalla memoria per liberare spazio ad altri flussi più attivi.

3.1.3 Metriche calcolate

Come spiegato nel paragrafo 3.1.1, CICFlowMeter produce in output un file CSV. Ogni riga di questo file riporta le metriche che sono state calcolate per ogni flusso individuato durante l'esecuzione. I primi sei valori di ogni riga contengono la quintupla che identifica il flusso e il flowid. Successivamente a questi troviamo più di 80 features. Eccone alcune:

- duration, in microseconds, ovvero tempo che intercorre dall'istante in cui il flusso viene creato al momento in cui termina
- total packet per tutto il flusso e nelle due direzioni fwd e bwd
- pakcet len min, max, totale, avg e deviazione standard per tutto il flusso e nelle due direzioni fwd e bwd
- bytes/s, bytes inviati in ogni secondo per tutto il flusso e nelle due direzioni fwd e bwd
- packets/s, bytes inviati in ogni secondo per tutto il flusso e nelle due direzioni fwd e bwd
- Inter Arrival Time (iat), ovvero il tempo che intercorre tra ogni pacchetto, min, max, tot, avg e deviazione standard per tutto il flusso e nelle due direzioni fwd e bwd

- ECE, CWR, PSH, RST, SYN, FIN, ACK, URG flags count, per tutto il flusso e nelle due direzioni fwd e bwd
- down/up ratio, il rapport tra i bytes inviati in fwd direction e bwd direction
- init win size, la dimensione della finestra tcp, sia per l'host client che per l'host server
- active time, min, max, tot, avg e deviazione standard, ovvero il tempo per cui il flusso si è trovato nello stato di attivo
- idle time, min, max, tot, avg e deviazione standard, ovvero il tempo per cui il flusso si è trovato nello stato di idle

Quelle elencate sono le più significative metriche calcolate da CICFlowMeter, nonché quelle utilizzate nei lavori presentati di seguito. [16]

3.1.4 Features-extraction con CICFlowMeter

A prima vista, per come descritto, il tool del CIC potrebbe sembrare uno strumento di utilità marginale ai fini del lavoro. In realtà, risulta interessante il lavoro svolto dallo stesso istituto grazie a CICFlowMeter. Si tratta di un lavoro di ricerca che ha l'obiettivo di estrarre dall'insieme di metriche calcolate il "più piccolo insieme di features per individuare ogni famiglia di attacchi"[3]. Il lavoro è stato portato a termine utilizzando un algoritmo di machine learning denominato *Random Forest Regression* e i risultati ottenuti sono stati testati valutando l'accuratezza, nel rilevare i vari attacchi, ottenuta mediante altri sette diversi algoritmi di machine learning.

Tale ricerca del CIC è stata molto utile, non solo in ambito di questo lavoro, ma anche di altre ricerche basate sul dataset CICIDS2017, per avere un punto di partenza nella detection. Come si vedrà nel capitolo 4, anche nel nostro caso, ad ogni attacco viene associato un sotto insieme di features che servirà ad identificare l'attacco stesso. Per creare i sottoinsiemi necessari al nostro scopo, il punto di partenza è stato, nella maggior parte dei casi, proprio il lavoro svolto dal CIC, i cui risultati sono mostrati nella tabella a pagina seguente.

Label	Features
Benign	Bwd Packet Len Subflow Fwd Bytes Total Len Fwd Packets Fwd Packet Len Mean
DoS Goldeneye	Bwd Packet Len Std Flow IAT Min Fwd IAT Min Flow IAT Mean
Heartbleed	Bwd Packet Len Std Subflow Fwd Bytes Flow Duration Total Len F.Packets
DoS Hulk	Bwd Packet Len Std Bwd Packet Len Std Flow Duration Flow IAT Std
DoS slowhttp	Flow Duration Active Min Active Mean Flow IAT Std
DoS slow loris	Flow Duration Fwd IAT Min Bwd IAT Mean Fwd IAT Mean
SSH-patator	Init Win Fwd Bytes Subflow Fwd Bytes Total Len Fwd Packets ACK Flag Count
FTP-patator	Init Win F.Bytes Fwd PSH Flags SYN Flag Count Fwd Packets/s
Web Attack	Init Win Fwd Bytes Subflow Fwd Bytes Init Win Bwd Bytes Total Len Fwd Packets
Infiltration	Subflow Fwd Bytes Total Len Fwd Packets Flow Duration Active Mean
BotNet	Subflow Fwd Bytes Total Len Fwd Packets Fwd Packet Len Mean B.Packets/s
Port scan	Init Win Fwd Bytes Bwd Packets/s PSH Flag Count
DDoS	Bwd Packet Len Std Avg Packet Size Flow Duration Flow IAT Std

Tabella 1: features-set estratti dal CIC utilizzando l'algoritmo Random Forest

3.2 Zeek

Il secondo progetto analizzato è un lavoro di tesi, svolto presso il *Royal Institute of Technology* (Stockholm, Svezia), basato su *Zeek* un IDS nato ormai più di venti anni fa ma ancora in uso grazie alla sua estensibilità.[17]

3.2.1 Cos'è Zeek?

Zeek è un monitor di rete che analizza tutto il traffico di rete su un collegamento alla ricerca di attività sospette.

Nonostante sia stato ideato nei primi anni '90, con il nome di Bro, ancora oggi rimane un protagonista nel mercato dei monitor di rete, ma anche nel campo della ricerca a riguardo. Il motivo di ciò risiede nei principi che sono alla base di Zeek: nasce come un software estensibile, scalabile e destinato a durare nel tempo. La sua particolarità consiste nel fatto che può essere esteso mediante script, permettendo così all'utente di ampliare le funzionalità del software, aggiungendo, in qualunque momento, pattern con cui effettuare il matching dell'attività di rete.

Un altro elemento che ha permesso a Zeek di essere ancora oggi così popolare è il formato degli output. Zeek non produce dei veri e propri output: non è dotato di una particolare GUI, né di output su linea di comando, ma produce dei file di log differenziati in base all'attività che essi descrivono. Viene prodotto un file quasi per ogni protocollo di rete, di trasporto e applicazione, ad esempio dopo un esecuzione di Zeek si potrà analizzare il file *http.log* dove vengono descritte tutte le richieste http con le rispettive risposte, un file *dns.log*, dove viene riassunta tutta l'attività DNS monitorata, e così via per ognuno dei protocolli conosciuti, compreso un file *dps.log* che contiene informazioni riguardo al traffico riscontrato sulle porte non standard. Questi log-files sono scritti in formato human-readable, in modo che lo stesso utente possa decidere di analizzarli direttamente oppure parsarli mediante uno Zeek-script nel livello PSI. [18]

3.2.2 Architettura di Zeek

Come scritto nel paragrafo precedente, Zeek si propone di avere come attributo caratterizzante l'estensibilità. Riesce a raggiungere tale obiettivo grazie ad un'architettura multi livello.

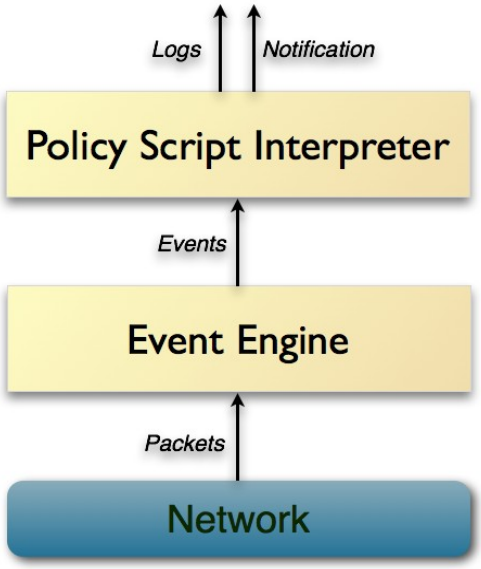

Immagine 3: Architettura di Zeek[19]

Come si osserva nell'Immagine 2, Zeek è composto da 2 livelli principali, l'*Event Engine* (EE) e il *Policy Script Interpreter* (PSI), oltre al livello di *Network*, il cui unico compito è quello di monitorare il traffico di rete, trasferendo ogni pacchetto al livello di EE, che è il vero e proprio nucleo dell'applicativo.

- livello *Event Engine*: costituisce il core di Zeek ed è il livello che permette agli utenti di manipolare, estendere ed ampliare le funzionalità dell'intero software. Ogni pacchetto che transita sulla rete giunge a questo livello e viene trasformato in un evento ad alto livello. Gli eventi sono la struttura dati alla base del funzionamento di Zeek. Riflettono le attività di rete, esprimendo solamente *cosa* è stato visto, senza specificarne il *perché* o se si tratta di un evento significante. Ad esempio ogni richiesta HTTP viene trasformata in un evento `http_request`, che contiene le informazioni principali della richiesta, come indirizzi IP e numeri di porta, la URI richiesta e la versione HTTP in uso.

- livello *Policy Script Interpreter*: è il livello caratterizzante di zeek, che si occupa di attribuire una semantica agli eventi ricevuti dal livello EE, eseguendo un insieme di "*event handlers*" scritti nel linguaggio di scripting di Zeek. Tali script analizzano gli eventi ricevuti, controllando, ad esempio, che non violino le regole di un azienda o che non rappresentino anomalie dannose per la rete. Possono, inoltre, lanciare degli alert in real time oppure eseguire programmi esterni per notificare, mitigare o anche fermare l'anomalia.

3.2.3 Machine Learning con Zeek

Dopo aver brevemente descritto il funzionamento dell'applicativo Zeek, verrà illustrato come è stato usato per effettuare una detection degli attacchi mediante algoritmi di machine learning, illustrando un lavoro di tesi svolto presso il Royal Institute of Technology, Università di Stoccolma.

Per prima cosa è stato prodotto uno script che permettesse a Zeek di calcolare metriche aggiuntive, senza le quali sarebbe risultato più difficile effettuare la detection.

A questo punto sono stati creati due script che avessero lo scopo di effettuare *features extraction*, ovvero di calcolare le metriche più significative per effettuare la detection. La motivazione per cui sono stati prodotti due script che effettuano lo stesso lavoro risiede nel fatto che il primo script, più complesso, estrae 50 features, mentre il secondo, più semplice, ne estrae soltanto 10[17]. Se a prima vista può sembrare inutile utilizzare entrambi gli scritps piuttosto che affidarsi soltanto al primo, in realtà bisogna farne un discorso di performance: il primo script utilizza una funzione a basso livello che diminuisce notevolmente le prestazioni di Zeek, aumentando, di conseguenza, il numero di pacchetti persi. Nel caso del secondo script le performance sono molto più elevate e il numero di pacchetti persi è prossimo allo zero.

Oltre a quello di estrarre le features con peso maggiore, l'altro compito degli scripts è quello di tradurre i log di output prodotti in un formato interpretabile da un terzo script, questa volta scritto in python che, grazie alla libreria *Pandas* fosse in grado di applicare

sui dati in ingresso diversi algoritmi di classificazione ML. Gli algoritmi usati sono i seguenti:

- Decision Tree, DT
- Random Forest, RF
- K-Nearest Neighbours, KNN
- Quadratic Discriminant Analyzis, QDA
- Naive Bayes, NB
- Support Vector Machine, SVM

Per utilizzare i suddetti algoritmi, il dataset è stato diviso in due porzioni: il 75% è stato utilizzato come *training-set*, mentre il restante 25% è stato utilizzato nella fase di *test*.

Dopo aver eseguito lo script, testando tutti gli algoritmi sopra elencati, viene riportata la seguente tabella, che mostra il tempo di esecuzione e l'accuratezza nell'individuare gli attacchi di ogni algoritmo. L'autore specifica che i dati relativi all'algoritmo SVM non sono stati riportati in tabella in quanto l'esecuzione è andata avanti per più di tre ore senza produrre alcun risultato.

	DT	RF	KNN	QDA	NB	SVM
Total time (s)	**0.36**	3.56	182.85	3.95	2.46	-
Accuracy (%)	98.87	98.95	**99.16**	19.18	10.33	-

Tabella 1: Velocità e accuratezza dei differenti algoritmi di ML[17]. I valori sono stati ottenuti testando Zeek, esteso con il modello descritto, su una macchina con OS Ubuntu, CPU 4×2.7 Ghz e memoria di 16 GB

Come si osserva dalla tabella, soltanto tre (DT, RF e KNN) dei sei algoritmi hanno prodotto valori di accuratezza degni di nota, seppur con tempi di esecuzione sopra la media (vedi KNN), e sono stati utilizzati per ulteriori esperimenti, mettendo da parte gli algoritmi meno efficaci.

Inoltre, utilizzando utilizzando l'algoritmo Random Forest, è stata prodotta la matrice di confusione rappresentata in Immagine 4. Sulle righe di tale matrice troviamo l'attacco che è stato identificato dall'algoritmo, mentre sulle colonne l'attacco che ci si aspetta che venga identificato. Dunque la cella (*i,j*) della matrice (riga i, colonna j), contiene il numero di flussi identificati come attacco i-esimo quando ci si aspetta l'attacco j-esimo. Ne segue che sulla diagonale troviamo i *True Positive*, ovvero i flussi classificati correttamente.

Actual attack class \ Predicted attack class	Benign	DoS GoldenEye	DoS Hulk	DoS Slowhttp	DoS Slowloris	Heartbleed	Infiltration	Botnet	Web Bruteforce	DDoS	FTP Bruteforce	Port Scan	SQL Injection	SSH Bruteforce	XSS
Benign	$9 \cdot 10^5$	939	108	465	53	1	2	2	212	785	2	1,641	4	59	135
DoS GoldenEye	811	3,830	4	19	2	0	0	0	0	0	0	1	0	0	0
DoS Hulk	155	14	41,314	341	12	0	0	0	4	0	0	0	0	0	1
DoS Slowhttp	540	17	271	3,806	278	0	0	0	0	0	0	1	0	0	0
DoS Slowloris	41	2	15	278	3,793	0	0	0	0	0	0	20	0	0	0
Heartbleed	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0
Infiltration	3	0	0	0	0	0	13	0	0	0	0	0	0	0	0
Botnet	0	0	0	0	0	0	0	182	0	0	0	0	0	0	0
Web Bruteforce	168	0	2	0	0	0	0	0	324	0	0	9	0	0	123
DDoS	1,078	0	0	0	0	0	0	0	0	36,970	0	0	0	0	0
FTP Bruteforce	4	0	0	0	0	0	0	0	0	0	986	0	0	0	0
Port Scan	1,922	0	0	0	12	0	0	0	8	0	0	38,132	0	0	1
SQL Injection	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SSH Bruteforce	73	0	0	0	0	0	0	0	0	0	0	0	0	569	0
XSS	119	0	3	1	0	0	0	0	110	0	0	8	0	0	69

Immagine 4: Confusion matrix per l'algoritmo Random Forest[17]

Come si osserva nella matrice di confusione, i risultati sono più che notevoli: ogni cella sulla diagonale troviamo un valore molto più alto rispetto quelli che troviamo sulla stessa riga e sulla stessa colonna. Non mancano però gli aspetti negativi: osservando la prima colonna (eccetto la cella [0,0]), notiamo che molti flussi per cui ci si aspetta che non venisse identificato alcun attacco sono stati classificati dall'algoritmo come malevoli. Possiamo dunque affermare che l'algoritmo usato non è esente dai *False Positive*, ovvero quei casi in cui un flusso viene classificato come malevolo anche se non lo è. Più preoccupante, immedesimandoci in una situazione reale, è la prima riga (sempre considerando l'eccezione della cella [0,0]): notiamo molti casi in cui ci si aspettava che l'algoritmo identificasse un attacco, ma ha classificato il flusso come benigno.

Per visualizzare meglio questi dati l'autore la seguente tabella in cui vengono riportati i valori del TPR e FPR per i tre algoritmi usati.

	DT	RF	KNN
TPR (%)	96.47	96.50	**97.76**
FPR (%)	0.57	0.48	**0.21**

Tabella 2: TPR e FPR del lavoro svolto su Zeek [17]

Con ciò non si vuole assolutamente criticare il lavoro svolto, ma soltanto far notare che nemmeno le tecniche di machine learning ci portano ad avere modelli perfetti, di cui possiamo fidarci al 100%.

3.3 ntopng

In questa sezione viene descritto *ntopng*, un software in grado di monitorare il traffico di rete raccogliendo metriche sui flussi, sugli hosts che compaiono e sulle interfacce di rete utilizzate. Inoltre, grazie alla sua interfaccia grafica accessibile tramite browser, è in grado di notificare errori, anomalie e malfunzionamenti nella rete.

3.3.1 Come funziona ntopng?

Ntopng è un software scritto in C/C++ che sniffa il traffico da un interfaccia di rete e analizza i pacchetti, suddividendo il traffico in flussi e in hosts. Inoltre utilizza la libreria nDPI (vedi paragrafo 4.1.1) per identificare il protocollo di livello applicazione usato in una determinata comunicazione. Tutti i dati raccolti vengono inviati sull'interfaccia localhost e possono essere analizzati collegandosi, da un qualsiasi browser, all'indirizzo *localhost:3000*. A questo punto verrà fornita una GUI che permette di analizzare le statistiche di ogni flusso e di ogni host individuato.
Durante la sua esecuzione, all'avvenire di determinati eventi (periodicità di cinque minuti, individuazione da parte di nDPI del protocollo per un flusso, …), vengono invocate delle callback scritte in Lua che verificano se il flusso (o l'host) contengano delle anomalie e, eventualmente, aggiornare la GUI.[20]

3.3.2 ntopng come IDS

Come accennato nel paragrafo precedente, ntopng è in grado di rilevare anomalie nel traffico di rete, il tutto senza utilizzare alcuna tecnica di machine learning. La differenza tra ntopng e l'obiettivo di questo elaborato risiede nel fatto che le quanto identificato da ntopng non riguarda gli attacchi descritti nel capitolo 2 (non tutti almeno), ma si tratta di anomalie interne alla singola comunicazione, come un numero troppo elevato di ritrasmissioni in un flusso TCP, un cambiamento nel fingerprint SSL (JA3) o un host

troppo lento a rispondere. Quando ntopng rileva una di queste situazioni allora notifica l’utente, tramite GUI, che può prendere i provvedimenti del caso. Questo software è anche in grado di rilevare alcuni attacchi più particolari, sempre rimanendo distante da qualunque forma di intelligenza artificiale. Alcuni esempi di questi attacchi sono l’attacco port scan e l’attacco botnet. Tale capacità di ntopng si è rivelata molto utile nello svolgimento del lavoro, permettendo di identificare anche gli attacchi su cui la metodologia descritta nel capitolo seguente non ha avuto effetto.[20]

4. Soluzione proposta

In questo capitolo verrà presentata la soluzione proposta per svolgere il lavoro di tirocinio, ovvero quello di individuare una strategia che permettesse di rilevare gli attacchi presenti nel dataset CICIDS2017 senza l'utilizzo di tecniche di machine learning.

4.1 Strumenti utilizzati

Per portare a termine il lavoro sono stati utili diversi struementi che permettessero di visualizzare il traffico di rete raccolto nel dataset CICIDS2017, e analizzarlo facilmente anche ad "occhio nudo".

4.1.1 nDPI e ndpiReader

Lo strumento utilizzato principalmente per svolgere tale lavoro, è stato senza dubbio nDPI e la sua demo-app ndpiReader. nDPI è una libreria che si occupa di attribuire il giusto protocollo di livello applicazione ad un flusso. Ad esempio sappiamo che, tipicamente, nel protocollo HTTP il client si connette al server sulla porta 80. Ciò non vieta che il server possa essere in ascolto delle richieste su una porta diversa dalla 80. Anche in questo caso nDPI riesce ad indovinare il protocollo utilizzato nella comunicazione.
Inoltre, alcuni protocolli possono essere utilizzati in svariati modi: ad esempio il protocollo HTTP, originariamente designato per il trasferimento di documenti HTML, oggi viene utilizzato per lo streaming audio/video o in protocolli peer-to-peer[20].
I protocolli supportati sono numerosi: dai più classici HTTP, TLS, SSH fino a quelli meno comuni come i protocolli utilizzati da specifiche applicazioni, si veda il caso del protocollo Facebook, WhatsApp o Telegram anch'essi supportati dalla libreria. Nello svolgere tale lavoro, lo strumento più utilizzato è stato *ndpiReader*, una demo-app che mostra le principali funzionalità della suddetta libreria. La scelta dell'uso di tale strumento è dovuta alla sua somiglianza con CICFlowMeter: si tratta, infatti, di un semplice tool che sniffa il

traffico di rete, suddividendolo in flussi, esattamente come descritti nel paragrafo 3.1.2, raccogliendo diverse metriche per ogni flusso. Questa somiglianza con CICFlowMeter ha permesso di poter analizzare il traffico di rete flusso per flusso e basarsi sulle stesse metriche calcolate dal tool del CIC.

Inoltre, utilizzando le dovute opzioni a riga di comando, è possibile far produrre a ndpiReader un file in formato CSV, simile a quello prodotto da CICFlowMeter, in modo che renda più facile l'analisi del traffico osservato sia da occhi umani che da tool esterni.

Per prima cosa è stato esteso l'insieme delle metriche calcolate in modo da avvicinarlo il più possibile a quello fornito dal CICFlowMeter. Ecco l'elenco delle features che sono state aggiunte:

- min/max packet len, sia nella direzione di bwd che nella direzione di fwd
- std dev packet len, sia nella direzione di bwd che nella direzione di fwd
- std dev iat, sia nella direzione di bwd che nella direzione di fwd
- min/max iat, del flusso
- std dev iat, del flusso
- conteggio dei flag TCP, sia nelle di rezioni di bwd e fwd che per l'intero flusso
- dimensione iniziale della finestra TCP per entrambi gli hosts
- goodput, in entrambe le direzioni, ovvero il rapporto tra la dimensione totale dei payloads scambiati e la lunghezza totale dei pacchetti

Oltre all'aggiunta di queste metriche, ndpiReader è stato anche esteso per poter eseguire la strategia di detection che verrà spiegata in seguito.

4.1.2 qtextasdata

Dopo aver prodotto, tramite ndpiReader, il file in formato CSV, è stato necessario poter analizzarlo. In questa fase del lavoro si è rivelato molto utile poter usufruire del tool *qtextasdata,* comunemente conosciuto col nome di "*q*", che permette di effettuare delle queries in linguaggio SQL, su file in formato CSV (e non solo).

Si è, dunque, rivelato utile per cogliere le principali differenze tra i flussi e benigni e quelli malevoli, per poi sfruttarle, come si vedrà nei paragrafi successivi, nella strategia di detection.[21]

4.1.3 Wireshark

Quando si parla di analisi di rete, Wireshark si dimostra sempre uno strumento affidabile ed efficace. La sua funzionalità principale è semplicemente quella di sniffare il traffico di rete (sia da interfacce live che da interfacce PCAP) e mostrare sulla GUI la lista dei pacchetti raccolti, fornendo tutte le informazioni relative a ciascun pacchetto, quali indirizzi ip e numeri di porta (sorgente e destinatario), l'istante in cui è stato catturato il pacchetto e la dimensione e contenuto del payload. La lista di pacchetti mostrata a schermo, può, inoltre, essere filtrata mediante un filtro di visualizzazione, scritto in un apposito linguaggio.

Oltre questa semplice funzionalità, tramite wireshark, è possibile trasportare il traffico di rete (completo o filtrato) su dei grafici che permettono di cogliere eventi particolari (disconnessioni, ritrasmissioni…) e periodicità nelle dinamiche di rete. [22]

Questo strumento si è rivelato utile per analizzare alcuni attacchi per cui le metriche calcolate non sono state direttamente d'aiuto.

4.1.4 editcap e tcpdump

Come si vedrà nel paragrafo 4.2.2 il dataset è stato "frammentato" in pcap più piccoli. Per farlo si è ricorso a due tool:

- editcap, che permette di estrarre da un file .pcap tutti i pacchetti compresi in una determinata fascia oraria passata come argomento al programma [23]
- tcpdum, che prende come argomento un file .pcap e un filtro eBPF, creando un secondo PCAP contenente solo i pacchetti che soddisfano i requisiti del filtro.[24]

In realtà, entrambi i software forniscono all'utente molte più funzionalità, a ci siamo limitati a descrivere soltanto quelle risultati utili ai fini del lavoro.

4.2 Strategia di detection

Entriamo ora nel vivo della strategia utilizzata per effettuare la detection degli attacchi elencati nel paragrafo 2.3.

Una buona strategia di detection deve essere tale da avere un elevato *True Positive Rate* (TPR) e un basso *False Positive Rate* (FPR). Nel nostro caso si parla di “true positive” quando la strategia classifica un flusso come maligno e quel flusso, come specificato nel dataset, è effettivamente un flusso maligno. Al contrario si parla di “false positive” quando la strategia compie un errore, classificando un flusso che, secondo le specifiche del dataset, dovrebbe essere benigno come un flusso malevolo, lanciando quindi un falso allarme. Quello che si desidera da una qualunque strategia di detection, indipendentemente dalle tecniche che essa adotta, è che sia perfetta, ovvero che abbia un TPR pari a 100% (in cui tutti i flussi malevoli vengono classificati come tali) e un FPR pari a 0% (nessun flusso benigno viene classificato come malevolo). Risulta, però, assai complicato far sì che entrambe le condizioni si verifichino contemporaneamente, pertanto l’obiettivo è quello di avere un TPR elevato, che si avvicini il più possibile al 100%, e un FPR che tenda a 0%.

4.2.1 Un approccio errato

Basandoci su quanto appena detto vediamo subito un approccio semplice ma errato, che porta ad avere, forse, un elevato TPR ma discapito di un anch’esso elevato FPR.

L’ approccio consiste nello stabilire, per ogni metrica, un intervallo di valori tale per cui se un flusso presenta valori all’interno degli intervalli relativi ad ogni metrica, allora il flusso è considerato benigno; se, invece, il valore di una sola metrica esce dall’intervallo allora il flusso viene considerato malevolo. Senza entrare troppo nel dettaglio, già si possono percepire diversi problemi: “come stabilire le soglie?”, “cosa garantisce che una volta determinato l’intervallo tutti (o quasi) i flussi benigni vi stiano all’interno e tutti (o quasi) i flussi malevoli vi stiano all’esterno?”, “quando il valore di una metrica esce dall’intervallo, come determinare quale attacco è in corso?”. Questa serie di domande, a cui è difficile trovare una risposta, ci fa capire che la strategia descritta poco fa non è adatta al nostro scopo, ma possiamo utilizzarla per capire la strada da intraprendere:

1) se un flusso è malevolo o meno, non lo possiamo decidere affidandoci al valore di una sola soglia
2) quando più metriche hanno valori al di fuori dell'intervallo, allora saranno queste stesse metriche ad indicarci quale attacco è in corso.

4.2.2 Suddivisione del dataset

Per prima cosa è stato necessario suddividere il dataset in files .pcap di dimensioni minori, in modo che potessero essere più facilmente gestibili. Per prima cosa ogni file .pcap è stato diviso in fasce orarie, tramite editcap, ognuna contente un attacco, seguendo le specifiche fornite dal CIC. Successivamente, i PCAPs così ottenuti sono stati filtrati nuovamente, usando tcpdump, per creare il PCAP finale contenente solamente i pacchetti scambiati tra i due host coinvolti nell'attacco. Parallelamente il file PCAP contente solo traffico benigno è stato frammentato soltanto basandosi sulle fasce orarie in cui, in un altro PCAP, avvengono degli attacchi. Ecco un esempio di procedimento di frammentazione del dataset.

Supponiamo di voler estrarre i dati relativi all'attacco DDoS, che secondo il CIC è avvenuto il giorno Venerdì 07/07/2017 dalle ore 15:56 alle ore 16:16 (GMT-4). Per prima cosa estraiamo l'attacco dal PCAP relativo al giorno venerdì, denominato Friday.pcap:

```
editcap -A "2017-07-07 20:56:00" -B "2017-07-07 21:16:00" Friday.pcap Friday_tmp.pcap
```

In questo modo abbiamo catturato nel file "Friday_tmp.pcap" tutto il traffico avvenuto il giorno Venerdì 07/07/2017 dalle 15:56 alle 16:16 (attenzione al fuso orario). Ora procediamo con l'estrazione dell'attacco vero e proprio:

```
tcpdump -r Friday_tmp.pcap -w DDdoS.pcap "host 172.16.0.1 and host 192.168.10.50"
```

Dopo aver eseguito questo comando da terminale, troveremo nel file DDoS.pcap tutti i pacchetti scambiati tra l'indirizzo IP 172.16.0.1 e 192.168.10.50 che sono rispettivamente

l’indirizzo IP dell’attaccante, processato dal firewall, e l’indirizzo IP del server locale attaccato.

A questo punto possiamo estrarre del file contenente solo traffico benigno, ovvero relativo al giorno di Lunedì 03/07/2017, i pacchetti transitati nella rete nella stessa fascia oraria in cui il venerdì è avvenuto l’attacco. Questa operazione è necessaria per poter confrontare i flussi di un attacco con flussi benigni relativi alla stessa fascia oraria. Si noti che, all’interno di una rete, il traffico può essere molto diverso a seconda della fascia oraria in cui viene catturato, ma che, allo stesso tempo, specialmente in aziende o uffici in cui gli orari di lavoro sono fissi, il traffico catturato, in giorni diversi ma nella stessa fascia oraria, presenta molte somiglianze. Sarà proprio l’assenza di queste somiglianze la chiave per la detection della maggior parte degli attacchi. Supponendo che il file relativo al giorno di Lunedì sia denominato Monday.pcap, allora il comando da eseguire è il seguente:

```
editcap -A “2017-07-07 20:56:00” -B “2017-07-07 21:16:00” Monday.pcap DDoS_benign.pcap
```

A questo punto disponiamo di due file “DDoS.pcap” e “DDoS_benign.pcap” che possono essere confrontati per ottenere le metriche da utilizzare per la detection.

Sui file così ottenuti è stata effettuata un ulteriore scomposizione in modo da poter utilizzare una parte, più grande, per determinare gli elementi necessari per effettuare la detection (come soglie, pesi e features-set) e la seconda porzione, più piccola, per poter testare i valori ottenuti. I file sono stati, dunque, divisi in una porzione dell’80% e una del 20% del file originale. Dai file “DDoS.pcap” e “DDoS_benign.pcap” abbiamo ottenuto dunque i file “DDoS_80.pcap” e “DDoS_benign_80.pcap”, utilizzati per stabilire soglie, features-set e pesi, e i file “DDoS_20.pcap” e “DDoS_benign_20.pcap”, utilizzati per la fase di test.

4.2.3 Metodologia

Verrà spiegata ora la procedura utilizzata per effettuare la detection, senza addentrarci in dettagli quali la costruzione dei features-set e l’assegnamento delle soglie e dei pesi, che verranno discussi nei paragrafi successivi. La strategia è basata su un punteggio attribuito

al flusso, il *flow_score*. Si tratta di un valore maggiore o uguale di zero che ci indica se il flusso è benevolo o meno. Se $0 \leq$ flow_score ≤ 1 allora il flusso è benevolo, se invece risulta che flow_score > 1 (il motivo di questa soglia, piuttosto che un'altra, si comprenderà in seguito) allora si tratta di un flusso malevolo. In questo modo però abbiamo fatto una semplice distinzione tra flussi benigni e flussi malevoli, che non ci permette di capire, nel caso in cui il flusso sia malevolo, quale attacco è in corso. Per tale motivo ad ogni flusso non viene assegnato un singolo punteggio, ma ne viene assegnato uno per ogni attacco che tale strategia è in grado di rilevare. Ogni flusso pertanto avrà un *DDoS score*, un DoS *Goldeneye score*, un *FTP-patator score* e così via per tutti gli attacchi elencati nel paragrafo 2.3. I valori degli score di un flusso vengono calcolati al termine dell'esecuzione di ndpiReader e vengono scritti nel file CSV, al quale è stata aggiunta un'opportuna colonna per ogni score. Si noti che i valori degli score non devono essere calcolati necessariamente alla fine, ma possono anche essere aggiornati periodicamente. Va da sé che se lo score viene calcolato su un flusso ancora "piccolo", ovvero su cui è stato scambiato un numero esiguo di pacchetti, anche nel caso il flusso sia maligno, lo score potrebbe risultare basso. Comunque sia, quando verrà calcolato le volte successive allora risulterà via via più alto. L'esatto numero di pacchetti scambiati, necessario perché la strategia abbia l'effetto desiderato, non è ben determinato in quanto dipende dall'attacco che si sta cercando. Per questo motivo, come si vedrà in seguito, tale metodologia non ha effetto su attacchi che, per loro struttura, generano sempre un numero di pacchetti inferiore a dieci, senza considerare eventuali three-way-handshake iniziale e two-way-handshake finale.

Prendendo spunto dalle considerazioni al termine del paragrafo 4.2.1, per calcolare gli score è stato assegnato, ad ogni attacco, un sottoinsieme metriche, i cui valori contribuiscono a determinare lo score. Il problema che sorge ora consiste nel "mescolare" i valori di tali metriche. C'è da notare che, specialmente nel caso di metriche molto diverse tra loro (si prenda come esempio l'inter arrival time e la lunghezza dei pacchetti), i valori di esse esprimono grandezze diverse. Per tanto si è deciso di *normalizzare* i valori delle metriche nell'intervallo [0, 1] prima di calcolare lo score. Per effettuare questo passaggio è

necessario dunque stabilire un valore minimo, *m*, e un valore massimo, *M*, per ogni metrica nel features-set dell'attacco per cui si sta calcolando lo score. Se il valore della metrica, *v*, è tale che $m \leq v \leq M$ allora il suo valore normalizzato, *v1*, sarà tale che $v1 \in [0, 1]$, in caso contrario risulterà $v1 \notin [0, 1]$. Queste due situazioni indicano che la metrica ha un valore accettabile, nel primo caso, e non accettabile nel secondo caso.

Vediamo come viene ottenuto il valore v1. Sia v il valore della metrica che si vuole normalizzare e m, M rispettivamente valore minimo e massimo determinati per la metrica, allora:

$$v1 = \begin{cases} \frac{v-m}{M-m} & \text{sse } v \geq m \\ 1-\frac{v-m}{M-m} & \text{else} \end{cases} \qquad (1)$$

In realtà per ottenere una semplice normalizzazione sarebbe sufficiente considerare il caso $v \geq m$, ma per il nostro scopo è necessario effettuare questa distinzione (vedremo poi il motivo). Inoltre grazie a questa distinzione possiamo assumere che nel caso in cui risulti $v \notin [m, M]$, allora necessariamente avremo $v1 > 1$.[24]

Dopo aver normalizzato i valori di tutte le metriche che compongono il features-set di un determinato attacco avremo un insieme *V* di valori $\{v1_1, v1_2, \dots, v1_k\}$, dove k corrisponde alla dimensione del features-set considerato. Lo score verrà calcolato mediante una somma pesata[26] dei valori normalizzati. Ciò vuol dire che ad ogni metrica di un features-set relativo ad un attacco, viene assegnato un peso, $0 < w < 1$, che indica quanto quella determinata metrica è caratterizzante per individuare un determinato attacco. A questo punto ci troviamo ad avere due array:

- un array di valori normalizzati *V*, $\{v1_1, v1_2, \dots, v1_k\}$

- un array di pesi *W*, $\{w_1, w_2, \dots, w_k\}$, dove w_i è il peso della metrica il cui valore normalizzato è $v1_i$ e tale che $(\sum_{i=0}^{k} W[i]) = 1$

Possiamo dunque calcolare lo score:

$$\text{score} = \sum_{i=0}^{k} V[i] * W[i] \qquad (2)$$

Se, al termine del calcolo risulta score > 1 allora significa che è in corso un attacco. Qual'è l'attacco in corso è determinato dal features-set che si sta esaminando. Osservando la formula (2) si comprende il motivo per cui nel normalizzare i valori è stata fatta la suddetta distinzione nel calcolo dei valori normalizzati. Nel caso in cui infatti il valore v fosse minore della soglia minima si avrebbe un valore normalizzato v1 < 0, comunque al di fuori del range considerato benigno. Tale valore, però, avrebbe avvelenato il calcolo dello score, rendendolo pressoché inutile. Per capire meglio consideriamo il seguente esempio, in cui si ha:

- l'insieme V dei valori normalizzati, {-1.5, 1.5, 0.73, -0.73}, in cui la normalizzazione è stata calcolata senza effettuare la distinzione tra il caso $v \geq m$ e $v < m$
- l'insieme W dei pesi, {0.3, 0.3, 0.25, 0.15}

Lo score, in questo caso, assumerà un valore pari a 0.073 che indica che non è in corso l'attacco considerato, nonostante il fatto che tre metriche su quattro considerate, tra cui le due con peso maggiore, abbiano un valore al di fuori del range. Se invece avessimo calcolato i valori normalizzati con la funzione descritta sopra allora l'insieme V sarebbe il seguente: {2.5, 1.5, 0.73, 1.73}. Lasciando invariato l'insieme dei pesi allora si otterrebbe uno score pari a 1.642.

Il perché sia necessario considerare più metriche deriva dalle considerazioni al termine del paragrafo 4.2.1. Il motivo per cui si è scelto di normalizzare i valori delle metriche risiede nel fatto che metriche diverse esprimono grandezze diverse. La scelta di assegnare un peso ad ogni metrica è dovuta, invece, ad un preconcetto teorico che riguarda la struttura e la dinamica dell'attacco che si sta considerando. Se prendiamo, per esempio, un attacco di

tipo slow, allora il suo features-set sarà composto prevalentemente da metriche riguardanti il tempo, come lo IAT, il numero di pacchetti al secondo o la durata. Ciò non vuol dire che tutte queste metriche hanno la stessa importanza nell'individuare gli attacchi slow, ma ce ne saranno alcune più caratterizzanti e che, di conseguenza, avranno un peso maggiore delle altre.

4.2.4 Eccezioni

Nonostante questa tecnica abbia funzionato per la maggior parte degli attacchi, in alcuni casi si è dovuto ricorrere a strategie diverse. Nel caso dell'*SQL injection*, del *XSS*, del *Port Scan* e dell'attacco BotNet la metodologia descritta sopra non ha portato ai risultati desiderati, in quanto, perché essa funzioni, è necessario che un flusso abbia dimensioni notevoli e nel caso degli attacchi di cui sopra questo non avviene mai. Ecco il perché:

- nel caso di SQL-injection e XSS, il client malevolo apre una connessione, invia una richiesta "avvelenata" e, una volta ottenuta la risposta chiude la connessione. Viene dunque rilevato un flusso da soli 2 pacchetti, oltre i necessari per aprire e chiudere la connessione
- il port scan, invece, è basato soltanto sull'apertura di una connessione per porta, senza che sia scambiato alcun ulteriore pacchetto tra i due host. Per tanto, sui flussi generati da quest'attacco, vengono scambiati soltanto il numero di pacchetti necessari ad aprire la connessione.
- l'attacco BotNet, come descritto nel paragrafo 2.3, è un attacco che non ha una sua struttura propria in quanto un BotNet viene poi utilizzata dall'attaccante per eseguire un altro attacco, avendo a disposizione un numero maggiore di hosts. Nel caso dell'attacco BotNet presente nel dataset, i flussi generati non sono sufficientemente grandi per poter essere identificati come malevoli dall'algoritmo descritto nel paragrafo precedente.

Inoltre gli attacchi sopra elencati generano dei flussi che, oltre che avere esigue dimensioni, a prima vista sembrano del tutto normali. Questo accade perché questi attacchi, per loro struttura, non sono basati sull'invio di un elevato numero di pacchetti

(come nel caso dei DoS/DDoS) o sull'idea di compiere ripetutamente la stessa azione (vedi l'esempio di patator e brute force), ma si limitano ad inviare delle richieste "avvelenate" ad un altro host e, per tale motivo, le metriche dei flussi generati non vengono contaminate.

Per identificare questi attacchi si è, dunque, dovuto ricorrere a strategie diverse. Nel paragrafo 3.3.2 si è detto che esistono già delle strategie, che non prevedono alcuna forma di intelligenza artificiale, per identificare attacchi di tipo BotNet[27] e Port Scan[28], pertanto non si è andati più a fondo per identificare questi tipi di attacchi.

Mentre, finora, non ci si è mai approfittati del fatto che il dataset non contenesse traffico cifrato, per non legare la strategia al dataset su cui è stata ideata, nel caso dell'SQL-injection e dell'XSS, invece, ci si è allontanati notevolmente dalla metodologia utilizzata per tutti gli altri attacchi, andando ad analizzare proprio il payload HTTP del pacchetto, controllando che la query effettuata non contenesse tracce di codice SQL, nel primo caso, o javascript, nel secondo caso. Tale controllo è stato effettuato contando le keywords dei due linguaggi presenti nella query.

La scelta di andare ad analizzare il payload è stata guidata dal fatto che i due attacchi generano flussi di dimensioni ridotte e dalle cui statistiche è difficile ricavare informazioni utili. Inoltre l'algoritmo utilizzato per decidere se una certa richiesta è "avvelenata" da codice SQL/javascript, non è strettamente legato al dataset CICIDS2017, ma può essere utilizzato anche da un server, prima di effettuare il parsing (e di esaudire) la richiesta del client.

4.2.5 Assegnazione delle soglie

Nel paragrafo precedente è stato spiegato che la strategia di detection è basata sul fatto che ad ogni attacco è associato un insieme di metriche, ad ognuna delle quali sono stati assegnate una soglia minima e una soglia massima, tali che se il valore di una metrica esce fuori dalle soglie allora tale metrica contribuirà ad aumentare il valore dello score.

Stabilire i valori di tali soglie è stata una procedura delicata: ricordiamo che il fatto che se il valore di una metrica è all'interno di un intervallo prestabilito non significa,

necessariamente, che il flusso sia benigno, come, allo stesso modo, se il valore è all'esterno del range non significa che il flusso sia maligno.

Le soglie sono state stabilite analizzando il traffico benigno. Dopo aver selezionato un attacco e costruito, con il procedimento che vedremo nei paragrafi successivi, il suo features-set adeguato, si è eseguito ndpiReader sul PCAP di traffico benigno corrispondente all'attacco scelto, specificando l'opzione che ci permette di avere, al termine dell'esecuzione, il file CSV, che è stato analizzato per estrarre le soglie per le metriche necessarie. La scelta che è stata fatta è quella di utilizzare il *percentile[29]*. Calcolare il k-esimo percentile di un insieme di valori significa trovare quel valore per cui il k% degli elementi dell'insieme è minore del k-esimo percentile, mentre il (100-k)% dei valori si trova al di sopra di esso. Nel nostro caso è stato scelto il terzo percentile per la soglia minima di una metrica e il novantasettesimo percentile per la soglia massima. In tal modo, per ognuna delle metriche selezionate, il 10% dei flussi benigni analizzati ha un valore al di fuori del range [3° percentile, 97° percentile]. La scelta di questi numeri non è stata casuale ma frutto di una serie di test. Se il range scelto fosse stato più stretto allora sarebbe sicuramente aumentato il TPR, ma insieme adesso sarebbe salito anche il FPR. Al contrario, scegliendo un range più ampio sia il TPR che il FPR sarebbero scesi.

4.2.6 Assegnamento dei pesi

Nel momento di dover assegnare dei pesi alle metriche che compongono il features-set di un attacco, la procedura seguita è stata molto simile a quella descritta, nel paragrafo precedente, per stabilire il range di valori, all'interno del quale, il valore di una metrica potesse essere considerato accettabile. Si ricordi, innanzi tutto, che il peso determina quanto una metrica, appartenente ad un features-set di un determinato attacco, sia caratterizzante per quell'attacco. Inoltre c'è da considerare che alcune sono meno determinanti di altre nella distinzione tra flussi benigni e flussi malevoli. Prendiamo come esempio la durata di un flusso e la dimensione media dei pacchetti. Il fatto che, nel caso di un flusso, i valori siano all'esterno del range accettabile stabilito per la durata è molto

meno indicativo di quanto lo sia un valore all'esterno dell'intervallo stabilito per la dimensione media dei pacchetti. Questo accade perché una volta che si conosce il protocollo di livello applicazione su cui avviene la comunicazione tra gli hosts del flusso, la dimensione dei pacchetti scambiati risulta essere quasi costante o, comunque, oscilla all'interno di un intervallo limitato. Per la durata della comunicazione, invece, non è possibile fare questo tipo di assunzioni: anche conoscendo il protocollo utilizzato e aver stabilito il range, come descritto nel paragrafo precedente, è molto più probabile che il valore della durata esca da tale range piuttosto che accada con il valore della dimensione media dei pacchetti.

Seguendo questa premessa, il primo passo per l'assegnazione dei pesi è stato quello di basarsi sulla dinamica dell'attacco considerato. Ricorrendo, ancora una volta, all'esempio degli attacchi di tipo slow, metriche come gli inter arrival time avranno un peso maggiore della durata del flusso. Quando ciò non è stato possibile o non ha portato ai risultati desiderati allora si è seguita la seguente tecnica. Fissato un attacco e determinato il suo features-set, si va a considerare il PCAP che contiene l'attacco e il file .pcap che contiene traffico benigno catturato nella stessa fascia oraria. Dopo averli analizzati con ndpiReader e aver prodotto i relativi file .csv si sono andati a contare il numero di flussi del PCAP maligno i cui valori, per le metriche selezionate, sono al di fuori dell'intervallo assegnato ad esse. Facendo il rapporto tra il numero di flussi con un valore al di fuori del range e il numero totale di flussi si ottiene la percentuale di flussi che hanno un valore anomalo per quella metrica. Una volta effettuato tale calcolo per tutte le metriche che compongono il features-set si ottiene un insieme di percentuali $P = \{p_1, p_2, \ldots, p_k\}$. Ricordiamo, però, che l'array di pesi che vogliamo ottenere deve essere tale che la sommatoria degli elementi dell'array sia 1. Pertanto, le percentuali ottenute vanno rapportate con la loro somma. Dunque l'array dei pesi finale $W = \{w_1, w_2, \ldots, w_k\}$ sarà tale che

$$\forall\ 0 \leq i < k,\ w_i = \frac{P[i]}{\sum_{j=0}^{k} P[j]}$$

4.2.7 Costruzione dei features-set

Nei paragrafi precedenti abbiamo detto che per ogni attacco presente nel dataset, è stato costruito un insieme di metriche utili ad individuare l'attacco, senza però soffermarci sul *come* questi insiemi sono stati ricavati.

Innanzitutto, si consideri che la dimensione dei features-sets non è fissata a priori, ma possono avere dimensioni variabili, a patto che l'insieme sia sufficientemente grande da allontanarsi dal caso in cui si affidi il compito di classificare il flusso ad una sola metrica. Allo stesso modo, scegliere un features-set di dimensione troppo grande contribuirebbe ad abbassare i pesi delle metriche (si ricordi che la somma dei pesi deve essere sempre pari a 1), comprese quelle più determinanti.

Si sono andate a guardare, prima di tutto, le caratteristiche dell'attacco che si stava considerando, ad esempio nel momento in cui si cercano le metriche per individuare un attacco slow, logicamente si terrà conto delle metriche riguardanti il tempo, tralasciando le metriche riguardanti la dimensione o il numero dei pacchetti, se invece si sta considerando un attacco volumetrico allora diventano significative metriche come la dimensione e il numero di pacchetti, lasciando da parte metriche come lo IAT o la durata del flusso. Non sempre però la dinamica di un attacco ci viene d'aiuto, specialmente nel caso in cui non è ben definita. Se si parla di un attacco DoS o DDoS, spesso questi hanno delle caratteristiche ben precise, ed è facile individuare quali sono le metriche caratterizzanti. Al contrario, per attacchi brute force risulta più complicato trovare delle caratteristiche che li accomunano: possono essere eseguiti a velocità diverse e la dimensione del pacchetto non è determinata a priori in quanto dipende dal protocollo o dall'applicazione che si sta cercando di forzare. In questi casi le strade seguite sono state principalmente due:

1) affidarsi ai features-set costruiti dal CIC, di cui si è discusso nel 3.1.4
2) estrarre i features-set dal file CSV prodotto da ndpiReader

Mentre il caso 1) risulta di immediata comprensione, ora verrà approfondito il caso 2). Una volta ottenuto i files .csv relativi al PCAP dell'attacco e al rispettivo PCAP benigno, si estraggono, per tutte le metriche, il 3° e il 97° percentile dal CSV benigno e, per ogni metrica, si contano il numero di flussi nel CSV malevolo che non ricadono nell'intervallo

ricavato. Le *n* metriche, con n a piacere, per le quali il numero di flussi così ricavato è più alto andranno a comporre il features-set.

5 Valutazione delle performance

In questo capitolo verranno valutate le performance, sia in termini di efficienza che di efficacia, della metodologia descritta nel paragrafo precedente. Spesso verrà utilizzato come termine di paragone il lavoro di tesi descritto nel paragrafo 3.2, non con l'obiettivo di svalutare il suddetto lavoro, ma cercando di evidenziare pregi e difetti dell'utilizzo di tecniche di machine learning nella rilevazione di anomalie nel traffico di rete. Il lavoro svolto presso il Royal Institute of Technology di Stoccolma, utilizzando Zeek è stato scelto come metro di paragone anche per il fatto che il lavoro si è basato sul CICIDS2017, lo stesso dataset utilizzato per questo lavoro di tirocinio.
Come accennato al termine del paragrafo 4.2.2, mentre per stabilire soglie, pesi e determinare i features-set, sono state utilizzate le porzioni maggiori estratte da ogni PCAP, i risultati descritti in questo capitolo sono stati ricavati testando i valori ottenuti sulla porzione più piccola.

5.1 Valutazione dell'efficacia

Per efficacia di un software, di un algoritmo o, come nel nostro caso, di una strategia, si intende la precisione del software (o dell'algoritmo o della strategia) nello svolgere il lavoro per cui è stato ideato. Come è stato descritto nel paragrafo 4.2, i valori che ci consentono di valutare un IDS sono il *True Positive Rate* e il *False Positive Rate[30]*. Nel nostro caso, il TPR è stato calcolato eseguendo ndpiReader, integrato con la metodologia descritta, sul .pcap maligno contenente l'attacco, mentre per calcolare il FPR abbiamo eseguito ndpiReader su un file .pcap contente un'ora di traffico estratta dal giorno di Lunedì. Per visualizzare i risultati e per poterli confrontare con quelli di cui si è parlato nel paragrafo 3.1, è stata costruita, anche in questo caso una matrice di confusione (a pagina seguente), imitando la semantica di quella visualizzata nella *Immagine 4* in modo che siano più facilmente confrontabili, nonostante le diverse partizioni del dataset.

	benign	ddos	dos_goldeneye	dos_hulk	dos_slow	ftp_patator	hearthbleed	infiltration	ssh_patator
benign	4305	20	279	0	26	35	0	0	6
ddos	0	9068	3	2828	0	0	1	2	0
dos_goldeneye	27	8950	1152	2828	11	0	1	0	0
dos_hulk	0	0	0	2828	0	0	1	2	0
dos_slow	225	6403	103	2828	1647	0	1	1	0
ftp_patator	155	5434	1	40	0	803	1	0	3
hearthbleed	120	0	0	2815	0	803	1	1	3
infiltration	0	118	0	31	0	0	1	2	0
ssh_patator	0	0	0	115	0	0	1	2	535

Actual Attack

Predicted Attack

Legenda

TP

FP

FN

Tabella 3: confusion matrix generata mediante la strategia descritta nel capitolo 4

Notiamo che le celle colorate di verde, costituiscono i True Positive, ovvero quei casi in cui ci è stato rilavato proprio l'attacco che ci si aspettava. In giallo sono colorati i False Positive, ovvero quei casi in cui un flusso è stato identificato come malevolo, ma in realtà non è avvenuto nessun attacco. In rosso, invece, abbiamo i False Negative, quei flussi malevoli che però sono stati classificati come benigni, ovvero dove la strategia fallisce.
Un altro elemento che salta all'occhio nella tabella è quando un flusso malevolo viene, sì identificato come tale, ma il tipo di attacco restituito da ndpiReader non è esattamente quello che ci si aspettava. In realtà non è proprio così: si ricordi che ndpiReader calcola diversi score, uno per ogni attacco, e in alcune situazioni capita che due score abbiano, contemporaneamente, dei valori malevoli (>1). Questo accade specialmente quando si parla di attacchi molto simili tra loro, ad esempio la famiglia degli attacchi DoS. In realtà quello che ci interessa è classificare un flusso in modo binario: malevolo o benigno. Per questo è stata costruita una seconda matrice di confusione, la cui semantica è identica alle precedenti, su cui però troveremo solamente 2 righe: una per i flussi classificati come benigni e una per i flussi classificati come maligni.

Tabella 4: Matrice di confusione binaria, i flussi vengono classificati solo come benigni o malevoli

	benign	ddos	dos_goldeneye	dos_hulk	dos_slow	ftp_patator	hearthbleed	infiltration	ssh_patator
benign	4305	20	279	0	26	35	0	0	6
malicious	282	9068	1295	2828	1647	803	1	2	538

In questa seconda tabella si può osservare una miglior distinzione tra i true positive, false positive e true negative, escludendo i casi in cui un flusso viene identificato con più di un attacco. In questo modo, inoltre, risulta più immediato il calcolo del TPR, del FPR, come mostrati nella seguente tabella:

FPR	TPR
6,15%	96,93%

A questo punto possiamo effettuare delle considerazioni su questi valori, confrontandoli con quelli ottenuti effettuando machine learning sulla piattaforma Zeek. Si nota che il True Positive Rate ottenuto con ndpiReader è leggermente più basso, ma identifica comunque la quasi totalità dei flussi malevoli. Per quanto riguarda il False Positive Rate, questo risulta essere più elevato.

Sia per il traffico benigno, che per gli attacchi, sono stati condotti dei test su dei file .pcap, trovati in rete, contenenti attacchi reali, ottenendo, anche in questi casi, un TPR molto elevato, talvolta anche del 100%, e un FPR inferiore a quello rilevato sul dataset.

In generale possiamo affermare che i valori di FPR e TPR ottenuti mediante machine learning sono più precisi di quelli ottenuti utilizzando la metodologia descritta. Si ricordi però che l'obiettivo di un IDS è quello individuare gli host con un comportamento anomalo. Anche nel nostro caso, questo obiettivo è stato raggiunto: basti osservare che con gli attacchi inclusi nel dataset, vengono rilevati diversi flussi tra l'host attaccante e l'host vittima. Seppur questi non vengono individuati tutti, quando oltre il 90% dei flussi tra due host sono classificati come malevoli, allora certamente uno di quei due host sta commettendo l'attacco riportato.

5.2 Valutazione dell’efficienza

Valutare l’efficienza di questa metodologia, e confrontarla con quella applicata su Zeek, risulta più complesso di quanto possa sembrare. Infatti non si tratta di un semplice paragone tra due applicativi che svolgono lo stesso lavoro, ma, mentre Zeek si occupa di raccogliere i pacchetti e a creare eventi per il livello di script, che in questo caso si limita solamente ad allenare il modello di machine learning, nel caso di ndpiReader vengono raccolte numerosi dati, prima di arrivare al calcolo degli score. Possiamo, però, calcolare il tempo impiegato da ndpiReader per il calcolo degli score. Per farlo è stato modificato ndpiReader in modo che calcoli il tempo impiegato per calcolare gli score di ogni flusso e, al termine dell’esecuzione, calcoli la media del tempo impiegato. È stato eseguito per cinque volte sul file .pcap contenente l’attacco DDoS ed ha restituito un tempo medio di calcolo degli score di 4 microsecondi eseguito su una macchina con OS Ubuntu 18.04 64-bit e CPU 2×1.10 Ghz e memoria di 4 GB. Se andiamo a confrontare tali valori con quelli riportati nella tabella 1 notiamo che il tempo impiegato da ndpiReader è circa 4 ordini di grandezza inferiore a quello impiegato da Zeek per effettuare la detection, considerando che i test effettuati con Zeek sono stati condotti su un hardware molto più performante.

Purtroppo l’autore non riporta dettagli su l’algoritmo utilizzato per effettuare la detection nello script di Zeek. Sappiamo però che si tratta di algoritmi di machine learning che, molto probabilmente, fanno uso di grandi quantità di memoria e di risorse hardware in generale. Se, invece, si va a fare un analisi computazionale dell’algoritmo descritto nel paragrafo 4.2.3, si nota che ndpiReader carica in memoria i dati relativi a un flusso e su di essi esegue una piccola quantità di somme e moltiplicazioni per poi liberare la memoria e ripetere il procedimento sul flusso successivo.

Si ricordi che questo è l’unico tempo necessario per effettuare la detection utilizzando questa strategia, mentre nel caso dei tempi riportati nel lavoro svolto su Zeek si parla dei tempi necessari per effettuare soltanto la fase di training, che sono comunque molto bassi, ma non di quelli necessari per effettuare la detection vera e propria.

6 Lavori futuri

In questo lavoro di tirocinio si è voluta proporre un'alternativa agli ormai numerosi IDS che fanno uso di machine learning per svolgere il loro lavoro. Come si è visto, la strategia presentata non è perfetta, ma necessita di essere affinata, principalmente cercando di migliorare il FPR, per poter risultare competitiva.

Uno dei lavori futuri che potrebbe risultare interessante sarebbe quello di rivedere il metodo utilizzato per assegnare i pesi e per stabilire le soglie di ogni metrica in modo da poter lasciare (quasi) invariato il TPR e abbassare il FPR.

Un'altra strada che si può intraprendere è quella di non preoccuparsi del FPR e TPR, in quanto, anche mantenendo questi valori è facile individuare sia un host che sta conducendo un attacco, sia un flusso che è stato annotato come malevolo a causa di un fallimento della strategia di detection. In merito a questo percorso sono già stati effettuati dei test che sono risultati promettenti: si è pensato di assegnare oltre che uno score ad ogni flusso, anche uno score ad ogni host, il cui valore è determinato dai valori degli scores dei flussi in cui è presente l'host stesso. Nei test che sono stati effettuati finora, gli scores degli hosts sono stati calcolati moltiplicando tra loro gli scores dei flussi tra l'host considerato un secondo host. In tal modo se l'host è presente in tanti flussi annotati come malevoli il suo *host_score*, sarà determinato da una produttoria di tutti, o quasi, fattori maggiori di uno. Se invece, tra tutti i flussi in cui troviamo l'host considerato, solo in una piccola parte vengono classificati come malevoli allore l'*host_score* sarà il prodotto di pochi fattori maggiori di uno e tanti fattori compresi tra zero ed uno, che contribuiranno a fare scendere lo score dell'host. L'applicazione di questi metodi ha portato a risultati promettenti, ma comunque da affinare, ad esempio per quanto riguarda il caso in cui un host è presente in pochi flussi e anche uno solo di questi viene rilevato come maligno. In questo caso non ci sono sufficienti flow_scores inferiori di ad uno che contribuirebbero a far scendere l'host_score.

Purtroppo non è stato possibile implementare anche questa strategia su ndpiReader in quanto il tool non raccoglie informazioni sugli hosts della rete e, pertanto, sarebbe stato necessario molto lavoro aggiuntivo soltanto per estendere ndpiReader in modo che sia in grado di raccogliere informazioni e statistiche, non solo per flusso, ma anche per host.
Ovviamente la strategia è stata ideata e testata sugli attacchi presente nel dataset CICIDS2017, ma si può applicare anche ad altri tipi di attacchi, non inclusi nel dataset, purché si disponga di una documentazione e di un traffico benigno adeguato per poter effettuare il paragone.

7 Conclusioni

Al termine di questo lavoro affermare che è possibile individuare anomalie nel traffico di rete, presenti nel dataset CICIDS2017, senza utilizzare alcuna forma di intelligenza artificiale.

Si ricordi che, come si è detto nel corso di questa relazione, il lavoro di un IDS, piuttosto che identificare uno ad uno i flussi, i pacchetti o, in generale, gli eventi malevoli, è quello di individuare un host che è causa di anomalie nel traffico di rete[31]. La metodologia riportata si limita ad annotare un flusso come malevolo o benigno, ma come si è visto nei capitoli precedenti, l'accuratezza di questa strategia non è perfetta. Se, però, si vanno ad analizzare i flussi nel file CSV prodotto da ndpiReader si possono fare due osservazioni:

1) se è in corso un attacco, allora si osserva un elevato numero di flussi tra l'host attaccante e l'host vittima
2) se non è in corso alcun attacco e, per errore della strategia, alcuni flussi vengono classificati come malevoli, allora si può vedere che sono pochi i flussi stabiliti tra le stesse coppie di hosts.

Questa differenza di volume tra le due situazioni ci porta a pensare che nel primo caso, nonostante non tutti i flussi maligni sono stati individuati, uno dei due hosts che compaiono in tutti i flussi malevoli stia conducendo un attacco verso l'altro host, mentre nel secondo caso, osservando un esiguo numero di flussi, annotati come malevoli, tra la stessa coppia di hosts si può tranquillamente affermare che l'allarme lanciato da ndpiReader sia dovuto semplicemente ad un errore nella connessione che ha causato rallentamenti nel flusso specifico, tralasciando l'idea che sia in corso un attacco.

Perciò, nonostante i valori di FPR e TPR non sono paragonabili con quelli ottenuti mediante machine learning, è comunque possibile notare un host che sta tenendo un comportamento anomalo e è possibile farlo con un minor dispendio di risorse (tempo, denaro e hardware). Si ricordi però che la metodologia descritta si limita soltanto a classificare i flussi, senza permetterci di effettuare un analisi più approfondita o di

osservare qual è stato il comportamento di un determinato flusso nel corso del tempo e come è cambiato. Inutile dire che tali obiettivi possono essere raggiunti soltanto mediante la costruzione di un modello di machine learning.

8 Bibliografia

[1] unb.ca/CICDataset/CIC-IDS-2017/Dataset/PCAPs/

[2] unb.ca/cic/datasets/ids-2017.html

[3] Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." *ICISSP*. 2018.

[4] Mirkovic, Jelena, and Peter Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms." *ACM SIGCOMM Computer Communication Review* 34.2 (2004): 39-53.

[5] Catillo, Marta, Massimiliano Rak, and Umberto Villano. "Discovery of DoS attacks by the ZED-IDS anomaly detector." *Journal of High Speed Networks* Preprint (2019): 1-17.

[6] github.com/lanjelot/patator

[7] Hofstede, Rick, et al. "Flow-Based Web Application Brute-Force Attack and Compromise Detection." *Journal of network and systems management* 25.4 (2017): 735-758.

[8] Wang, Jun, et al. "Risk assessment of buffer" Heartbleed" over-read vulnerabilities." *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015.

[9] Durumeric, Zakir, et al. "The matter of heartbleed." *Proceedings of the 2014 conference on internet measurement conference*. ACM, 2014.

[10] Florêncio, Dinei, Cormac Herley, and Baris Coskun. "Do strong web passwords accomplish anything?." *HotSec* 7.6 (2007).

[11] Anley, Chris. "Advanced SQL injection in SQL server applications." (2002).

[12] Bates, Daniel, Adam Barth, and Collin Jackson. "Regular expressions considered harmful in client-side XSS filters." *Proceedings of the 19th international conference on World wide web*. ACM, 2010.

[13] Stone-Gross, Brett, et al. "Your botnet is my botnet: analysis of a botnet takeover." *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009.

[14] Wolfgang, Mark. "Host Discovery with nmap." *Exploring nmap's default behavior* 1 (2002): 16.

[15] Lee, Cynthia Bailey, Chris Roedel, and Elena Silenok. "Detection and characterization of port scan attacks." *Univeristy of California, Department of Computer Science and Engineering* (2003).

[16] netflowmeter.ca/netflowmeter.html

[17] Gustavsson, Vilhelm. "Machine Learning for a Network-based Intrusion Detection System: An application using Zeek and the CICIDS2017 dataset." (2019).

[18] www.zeek.org

[19] docs.zeek.org

[20] Deri, Luca, Maurizio Martinelli, and Alfredo Cardigliano. "Realtime high-speed network traffic monitoring using ntopng." *28th Large Installation System Administration Conference (LISA14)*. 2014.

[21] github.com/harelba/q

[22] Lamping, Ulf, and Ed Warnicke. "Wireshark user's guide." *Interface* 4.6 (2004).

[23] Analyzer, Network Protocol. "Ethereal." (2003).

[24] Jacobson, Van, Craig Leres, and Steve McCanne. "The Tcpdump Manual Page. Lawrence Berkeley Laboratory." (1989).

[24] Patro, S., and Kishore Kumar Sahu. "Normalization: A preprocessing stage." *arXiv preprint arXiv:1503.06462* (2015).

[26] Smith, T. F., Z. F. Shen, and J. N. Friedman. "Evaluation of coefficients for the weighted sum of gray gases model." (1982): 602-608.

[27] Bailey, Michael, et al. "A survey of botnet technology and defenses." *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE, 2009.

[28] Sridharan, Avinash, Tao Ye, and Supratik Bhattacharyya. "Connectionless port scan detection on the backbone." *2006 IEEE International Performance Computing and Communications Conference*. IEEE, 2006.

[29] Newson, Roger. "Confidence intervals for rank statistics: Percentile slopes, differences, and ratios." *The Stata Journal* 6.4 (2006): 497-520.

[30] Hodo, Elike, et al. "Shallow and deep networks intrusion detection system: A taxonomy and survey." *arXiv preprint arXiv:1701.02145* (2017).

[31] Bace, Rebecca, and Peter Mell. *NIST special publication on intrusion detection systems*. BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, 2001.