POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Identificazione di scansioni di vulnerabilità in traffico di rete

Relatori
prof. Antonio Lioy
dott. Daniele Canavese
dott. Leonardo Regano

Candidato
Alberto Solaro

Anno accademico 2019-2020

A Maria Teresa, Emilia,
Luciano ed Eusebio

Ringraziamenti

Con questa tesi colgo l'occasione per ringraziare tutte le persone che mi hanno accompagnato in questo lungo e soddisfacente viaggio. Quando mi sono iscritto al Politecnico mai avrei immaginato una simile conclusione del percorso, soprattutto perchè allora non mi sarei aspettato tutto l'aiuto e il supporto che invece ho ricevuto da ognuna delle persone che ho incontrato lungo la via.

Vorrei partire sicuramente dalla mia famiglia che è composta da due nuclei. Il primo, quello che spero sarà il mio futuro, costituito dalla mia compagna di vita, *Gingy*, che mi ha accompagnato sin dall'idea di iscrivermi al poli, fino alla revisione di questa tesi, sopportandomi e supportandomi negli ultimi sette anni, e da *Artù* che, biscotto permettendo, è un compagno fedele. Il secondo nucleo è invece quello in cui sono nato e cresciuto: *Mamma*, *Papà* a *Luca*. Loro mi hanno sempre spinto ad essere un *pigiatasti* migliore e a non mollare, dando significato al caos che ho generato nella loro vita, pur non cogliendo nemmeno una virgola di quello che stavo studiando, la maggior parte delle volte!

Una menzione speciale va a *Buga* e *Pietro*, quasi fratelli più che semplici amici, con i quali ho condiviso molto e so che, a prescindere da tempo e luogo, sapremo sempre ritrovarci per condividerne ancora. Un altro ringraziamento speciale va agli amici del portico: *Fefi*, *GIULIA (Mela)*, *Mary* e *Matte*. Nonostante loro quasi sicuramente non coglieranno la citazione, posso dire che ovunque saremo nel mondo potremo realizzare questo scenario. Grazie al gruppo dei "cinghiali" *Alo*, *Deme*, *Manu*, *Ciotti*, *Ala*, *Maff*, *Dario* e *Silvioooh!*. Con tutti loro ho condiviso momenti incredibili e soprattutto divertenti in grado di distrarmi, anche se per poco, dai momenti difficili. Grazie ai miei compagni di viaggio *Daniel*, *Marco*, *Francesco* e *Dequi*, che hanno condiviso con me questo percorso impegnativo. Ci siamo aiutati e supportati arrivando alla fine e rendendo il percorso decisamente più interessante e, perchè no, più semplice.

Un grazie speciale lo vorrei dedicare a *Quirum*, *Stefano*, *Piero* e *Mattia*, che mi accompagnano nella mia prima esperienza lavorativa in qualità di capi e colleghi, ma soprattutto amici. Mi hanno permesso di conoscere una, seppur piccola, incredibile realtà che, a mio parere, rappresenta a tutti gli effetti una di quelle PMI di cui si lodano le competenze e la preparazione quando si sente parlare di Italia dall'estero. Insieme a Erre Elle Net mi hanno fatto riscoprire il Piemonte, tramite le sue diverse realtà, apprendendo così la necessità di intervenire per ridurre il, purtroppo, famoso *digital divide*.

Sicuramente va ringraziato anche il mio amico geniale, *Fred*, che sopporta i miei interminabili sproloqui e le mie strane visioni del mondo alimentandole. Un ringraziamento e una scusa ufficiale vanno a *Marco*, il mio incredibile compagno di banco che ha visto nascere tutto questo in prima linea, anche se, alla fine, purtroppo il lato oscuro dell'informatica ha vinto, infrangendo così il nostro sogno. Infine un grazie particolare va a *Monica*, che mi ha aiutato a riordinare i pensieri e a procedere con metodo, altrimenti senza di lei tutto questo non ci sarebbe stato. Ovviamente ci sarebbero un sacco di altre persone da ringraziare e spero che non me ne vorrete, ma comunque sappiate che vi ho comunque nei miei pensieri: *Gian*, *Riki* e *Lovato*, *Edo*, *Alberico* e *Laura*.

Indice

Capitolo 1

Introduzione

Negli ultimi anni uno dei temi nel mondo dell'informazione su cui la ricerca ha dedicato particolare attenzione è quello della privacy, tanto che diversi organi nazionali e sovranazionali hanno creato delle leggi per la sua tutela, ne è un esempio il *General Data Protection Regulation*[1] (GDPR). L'utilizzo di *Internet* è diventato centrale sia nella vita personale che lavorativa di molte persone e tramite queste leggi si cerca di tutelare il consumatore e informarlo su chi detiene le sue informazioni e a quale scopo. Questa attenzione sulla materia comporta quindi non solo un adeguamento burocratico, ma anche tecnologico che ha spinto diverse aziende che si occupano di informazione ad aumentare l'integrazione dei propri prodotti con sistemi più sicuri.

Ne è un esempio Google che, come descrive nel suo *report annuale sulla trasparenza*[2], ha aumentato l'implementazione dei protocolli di sicurezza nei suoi prodotti, come Gmail e tutte le sue altre applicazioni, e investito nella comunicazione per sensibilizzare altre aziende, incentivando l'utilizzo di HTTPS, spiegato nella Sezione 2.1.3, per i siti web migliorandone il posizionamento nel proprio motore di ricerca. Inoltre tramite questo report annuale è riportata una statistica sul rapporto di protocollo sicuro HTTPS rispetto al traffico totale prodotto dal browser Chrome[3] e che viene riportato nella Figura 1.1. Come è possibile notare, nell'ottobre 2020, la quasi totalità del traffico, circa il 95%, generato da utenti che utilizzano il browser di Google avviene tramite il protocollo cifrato. Si può inoltre osservare che l'evoluzione è avvenuta anche in tempi relativamente brevi nonostante lo sforzo necessario a sensibilizzare gli utenti e quello tecnologico per adeguare i sistemi.

L'utilizzo di protocolli cifrati permette di aumentare la sicurezza per l'utente finale, in particolare consente di ridurre il rischio di un possibile attacco di man-in-the-middle (MITM), spiegato nella Sezione 4.1. Tale aspetto rende però più complesse le meccaniche di analisi e di intervento delle grosse società e degli internet service provider (ISP) che infatti, per applicare politiche sulla qualità del servizio (QoS) o politiche di sicurezza personalizzate, utilizzano dei software in grado di elaborare il traffico e distinguerne i diversi servizi. Grazie a questa classificazione è poi possibile intervenire su ognuno di essi per andare incontro alle esigenze del cliente e/o dell'azienda sia che si tratti di renderne prioritario il traffico di uno in particolare oppure di monitorarlo.

Alcune delle soluzioni legate all'analisi del traffico sono i sistemi per l'individuazione e prevenzione delle intrusioni (IDS/IPS), strumenti che, facendo un'ispezione approfondita dei pacchetti di rete, spiegati nella Sezione 2.1.1, sono in grado di risalire a comportamenti sospetti e possono sia avvisare chi si occupa della sicurezza sia intervenire per confinare la possibile intrusione. Questo tipo di analisi è detta *Deep Packet Inspection* (DPI), in quanto analizza il payload dei diversi pacchetti interpretandolo e cercando di comprendere la semantica del contenuto. La DPI si contrappone all'analisi che non ne interpreta il payload del pacchetto, ma ne legge solamente le informazioni presenti nell'intestazione detta *Shallow Packet Inspection* (SPI).

---

[1] https://www.garanteprivacy.it/il-testo-del-regolamento

[2] https://transparencyreport.google.com/https/overview

[3] https://www.google.com/intl/it_it/chrome/

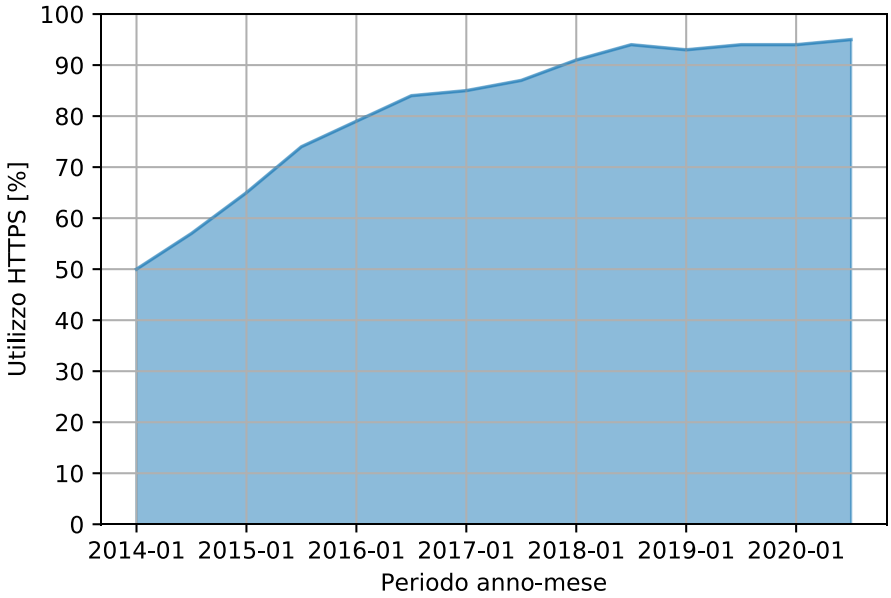


Figura 1.1: Percentuale di utilizzo del protocollo HTTPS rispetto al traffico totale generato da servizi di Google.

Protocolli come SSL, spiegato nella Sezione 2.1.2, cifrano il traffico end-to-end permettendo quindi di scambiare tra due terminali informazioni in maniera sicura. Molti protocolli applicativi hanno adottato questo tipo di sicurezza aggiungendo il supporto per il protocollo HTTPS, rendendo quindi più sicuro un protocollo esistente senza doverlo ripensare. Ne è un esempio il protocollo web HTTP che tramite l'impiego di SSL diventa HTTPS, spiegato nella Sezione 2.1.3. Dal momento che il traffico è cifrato, rende più complesse le analisi DPI, che hanno la necessità di decifrare prima il contenuto dei pacchetti per poi poterlo interpretare. Esistono alcune soluzioni in grado di eseguire ugualmente questo tipo di analisi, esposte nella Sezione 4.1, come per esempio BlindBox [1], ma che richiedono un lavoro maggiore rispetto alle opzioni esistenti su traffico non cifrato e, in alcuni casi, meno efficace vanificando quindi l'utilizzo di protocolli sicuri.

Le analisi SPI subiscono un impatto minore, ma hanno lo svantaggio di poter operare solamente con i dati presenti nelle intestazioni dei pacchetti che contengono una quantità di informazioni ridotte e meno interpretabili rispetto a quelle presenti nel payload. Di conseguenza, è più difficile individuare delle relazioni tra questa tipologia di dati e un particolare contenuto. Il *machine learning* o apprendimento automatico, approfondito nel Capitolo 3, può essere una risorsa valida per mitigare le problematiche appena descritte in quanto è in grado di individuare delle relazioni statistiche tra i diversi dati.

L'apprendimento automatico sta rapidamente cambiando diversi settori nel mondo dell'informazione, permettendo di raggiungere, in alcuni casi, risultati migliori rispetto alla singola capacità umana. Tra i campi maggiormente influenzati ci sono l'economia, l'analisi del linguaggio naturale e la sicurezza informatica. In quest'ultimo caso per esempio l'apprendimento automatico viene molto utilizzato per la creazione di nuovi antivirus che non si basano esclusivamente su di un database finito di virus tra cui riconoscerlo. Inoltre viene impiegato per lo sviluppo di sistemi di filtraggio per la posta elettronica in grado di individuare spam o email indesiderate, spostandole di cartella per evitare di imbattervisi.

Questa tipologia di analisi apre quindi alcuni nuovi scenari nel campo dell'identificazione dei servizi e dell'individuabilità degli attacchi informatici, anche se trasportati tramite protocolli cifrati. Nel campo della sicurezza informatica, come accennato precedentemente, uno strumento molto utile è l'IPS (intrusion prevention system), che ha lo scopo, come suggerisce il nome, di prevenire le intrusioni, ad esempio può riconoscere gli utenti che stanno preparando un attacco. Questa fase solitamente viene eseguita tramite software che scansionano reti e sistemi vittima in modalità automatica e, tramite alcune liste pubbliche, possono individuare vulnerabilità note e

quindi tentare successivamente l'attacco. Questo tipo di strumenti è detto *scanner di vulnerabilità*, approfondito nella Sezione 2.3.

Tali strumenti sono in grado di generare report estremamente dettagliati contenenti diverse informazioni sulla macchina scansionata, frequentemente riescono a riconoscere il sistema operativo utilizzato e in alcuni casi rilevare le versioni dei software che sono esposti all'accesso da remoto. Grazie a questo tipo di informazioni così precise è possibile ricercare, tramite database che collezionano vulnerabilità note, di quali di esse possa soffrire il sistema vittima.

Un esempio di attacco che ha colpito diverse aziende, come spiegato in [2], che utilizzavano terminali con sistemi operativi Windows non aggiornati è stato *EternalBlue*. Tramite questo attacco è stato possibile installare dei *ransomware*[4], bloccando così i sistemi delle aziende. L'attacco sfruttava una vulnerabilità del protocollo Sever Message Block[5] (SMB) presente sul sistema operativo che permetteva di caricare ed eseguire codice arbitrario su di esse. Grazie agli scanner di vulnerabilità gli attaccanti sono riusciti facilmente ad identificare molti sistemi che presentavano tale falla.

Gli scanner però non sono solo utilizzati per eseguire attacchi informatici, ma possono essere preziosi strumenti per i sistemisti che li impiegano per generare report periodici in grado di suggerire aggiornamenti e adeguamenti dei sistemi cercando quindi di anticipare eventuali attaccanti. Inoltre questi strumenti sono difficili da individuare in un traffico di rete, in quanto non eseguono operazioni che farebbero allertare gli IDS e, anche analizzando le singole richieste, sarebbe difficile scoprirne le intenzioni senza osservare l'interezza della connessione. Per questi motivi spesso è difficile bloccare totalmente l'utilizzo di tali scanner, sia per non togliere strumenti utili sia perchè tecniche classiche possono non essere sufficienti per poterli identificare.

Lo scopo di questa tesi è quindi quello di proporre una possibile soluzione in grado di individuare l'utilizzo di scanner di vulnerabilità su di un traffico di rete. Per ottenere tale risultato si utilizza un flusso di esecuzione in grado di allenare alcuni classificatori che uniscono l'analisi SPI con l'apprendimento automatico. In particolare si utilizzeranno modelli di foreste casuali, spiegate nella Sezione 3.2.1, e reti neurali, esposte nella Sezione 3.2.2. Questi due modelli sono stati selezionati per mettere a confronto due tipologie differenti, la prima più classica e la seconda più moderna, rappresentando la base del deep learning.

Oltre all'individuazione dell'utilizzo di tali strumenti questo studio si propone anche di riconoscerne alcuni in particolare. Si selezioneranno quindi degli scanner di vulnerabilità che verranno utilizzati per generare i dati da analizzare e, attraverso i classificatori, si cercherà non solo di identificarne l'utilizzo, ma anche di riconoscere quale sia stato utilizzato. Questo permette di raggruppare differenti connessioni per strumento utilizzato e superare alcune delle limitazioni che una semplice analisi SPI potrebbe avere.

Come anticipato in questo Capitolo, le analisi del traffico stanno cambiando, a causa dei protocolli cifrati, verso l'adozione di tecniche che non eseguono interpretazione del payload, ma solo dell'analisi della sua intestazione e, sempre più spesso affiancate a modelli di apprendimento automatico che permettono di raggiungere tale scopo. Come verrà poi approfondito nel Capitolo 4, si è quindi cercato di sviluppare una soluzione che non fosse applicabile unicamente alla categoria degli scanner di vulnerabilità, ma che fosse più generica e potesse essere utilizzata per altre tipologie di identificazioni. Per questo motivo durante la realizzazione della sperimentazione si è prestata particolare attenzione alla possibilità di poter modificare il flusso di esecuzione il più liberamente possibile, così da poterlo adattare alle diverse esigenze di future analisi.

I quattro diversi classificatori allenati, due foreste casuali e due reti neurali, hanno ottenuto generalmente buoni risultati sia nella distinzione tra traffico legittimo e traffico di scanner di vulnerabilità, sia nel riconoscimento dei diversi software. I modelli basati su foreste casuali sono risultati più precisi, mentre quelli fondati sulle reti neurali hanno mostrato maggiori difficoltà specialmente nella classificazione puntuale degli scanner.

[4] È una tipologia di virus che riduce l'accesso al dispositivo infettato e che richiede un riscatto per poterlo liberare.

[5] Un protocollo utilizzato, soprattutto da sistemi operativi Windows, per la condivisione e lo scambio di file e informazioni tra client e server.

La struttura della tesi è la seguente:

- nel Capitolo 2 verranno esposti i concetti di reti, in particolare i protocolli coinvolti nell'analisi, gli strumenti di analisi esistenti e gli scanner di vulnerabilità disponibili;
- nel Capitolo 3 saranno approfondite le basi sull'apprendimento automatico e i modelli utilizzati per la classificazione, in particolare foresta casuale e rete neurale;
- nel Capitolo 4 verrà presentato il flusso di esecuzione proposto e verranno illustrati i diversi software che lo compongono e come sono collegati tra loro;
- nel Capitolo 5 verrà descritto nel dettaglio l'insieme di dati di partenza e ne verranno evidenziate le caratteristiche principali ed esposte alcune osservazioni preliminari all'esecuzione dell'analisi;
- nel Capitolo 6 saranno esposti i risultati sperimentali ottenuti dai diversi modelli e presentate le performance ottenute da ognuno di loro;
- nel Capitolo 7 verranno presentati alcuni studi collegati con il suddetto lavoro di tesi, soffermandosi sulle caratteristiche comuni;
- nel Capitolo 8 saranno descritte le osservazioni finali in merito allo studio;
- nell'Appendice A, vengono riportate le tabelle presentate nello studio a cui vengono fatti diversi riferimenti;
- nell'Appendice B, sono descritti nel dettaglio i due software scritti per questa tesi, soffermandosi sia sulla parte per l'utente che su quella per lo sviluppatore.

Capitolo 2

Concetti base di reti

In questo Capitolo vengono evidenziati la teoria e i concetti dietro agli strumenti che verranno utilizzati per eseguire lo studio, quali:

- i principali protocolli di rete coinvolti che hanno contribuito alla scelta di alcune soluzioni adottate. I protocolli analizzati sono tre: TCP, TLS e HTTP;
- i principali strumenti per l'analisi di una rete, partendo dal funzionamento della cattura del traffico a strumenti più complessi che permettono di estrarre diverse tipologie di statistiche;
- gli strumenti di vulnerability scanning, una panoramica dei principali strumenti che si possono trovare oggi e che possono essere utilizzati per scansionare le reti e programmi scoprendone alcune caratteristiche;

Tra i protocolli di rete esposti il principale è il TCP su cui si basa lo studio, tra tutti è quello fondamentale e sarà soggetto delle analisi successive. Il protocollo TLS permette di aggiungere protezione e integrità ai messaggi scambiati, per via delle sue caratteristiche e risultando sempre più diffuso non permette un approccio all'analisi di rete fino ad ora utilizzato. Il protocollo HTTP viene esposto in quanto maggiormente diffuso e principale componente del web.

2.1 Protocolli di rete

Una rete di telecomunicazioni, come definito nelle raccomandazioni del International Telecommunication Union[1] (ITU-T), è un insieme di nodi e canali che forniscono un collegamento tra due o più punti per permettere trasmissione e ricezione di segnali tra essi. Il processo di interconnessione e allocazione delle risorse di rete per il trasferimento dell'informazione è detta commutazione.

Ci sono due tipologie di rete in base al tipo di commutazione utilizzata:

- *rete a commutazione di circuito*, la rete usa le risorse disponibili per allocare un circuito, ovvero un collegamento fisico, interamente dedicato alla comunicazione. Le risorse vengono successivamente rilasciate al termine della comunicazione;
- *rete a commutazione di pacchetto*, la rete non alloca risorse esclusive e le condivide tra tutte le comunicazioni. L'informazione da trasferire è organizzata in unità di dati detti Protocol Data Unit (PDU) o pacchetti.

Un pacchetto (vedere Figura 2.1) è composto da: una parte di controllo denominata Protocol Control Information (PCI) o intestazione e una parte di informazione denominata Service Data Unit (SDU) o payload.

[1] https://www.itu.int/rec/T-REC-I.112-199303-I

Figura 2.1: Struttura di un pacchetto di rete.

Le reti a commutazione di circuito consegnano i messaggi direttamente al circuito che le trasporta a destinazione. Invece quelle a commutazione di pacchetto consegnano le PDU alla rete che, prima di giungere a destinazione, attraversano vari nodi. Il pacchetto, passando attraverso ogni nodo, viene memorizzato, elaborato per determinarne il canale da percorrere e successivamente ritrasmesso su quello stesso canale. Questo tipo di comportamento viene detto store and forward.

I protocolli di comunicazione descrivono formalmente le interazioni tra elementi, mentre i processi, le funzioni e le relazioni tra le entità coinvolte nella comunicazione sono definite nell'architettura di rete. La stratificazione permette ad un'architettura di descrivere la suddivisione delle diverse funzionalità in più strati. Ogni strato fornisce un servizio ad uno superiore e integra quello offerto da uno inferiore, le entità coinvolte comunicano tramite questo servizio.

Un servizio in base alla modalità di gestione della comunicazione può essere:

- orientato alla connessione o *connection-oriented*, viene prima stabilita una comunicazione tra gli interlocutori, successivamente viene trasferita l'informazione e infine viene rilasciata la connessione;
- senza connessione o *connectionless*, i dati vengono immessi in rete senza un accordo preliminare e sono trattati in modo indipendente.

Le reti a commutazione di pacchetto, a seconda del tipo di servizio, si dividono in *reti a circuito virtuale*, se utilizzano una comunicazione orientata alla connessione, e *reti a datagramma*, se utilizzano una comunicazione senza connessione. Le reti a circuito virtuale sono concettualmente simili a quelle a commutazione di circuito, in quanto entrambe le tipologie sono caratterizzate da una fase di apertura, scambio delle informazioni e chiusura della connessione, ma differiscono nella gestione delle risorse. Infatti, nelle reti a circuito virtuale le risorse non vengono allocate staticamente.

Una rete a commutazione di circuito, come detto in precedenza, gestisce una connessione fisica tra le entità, mentre una rete a circuito virtuale gestisce una connessione logica, ovvero un percorso che permette il collegamento tra le entità, ma che non è dedicato esclusivamente ad esse. In una rete a circuito virtuale i pacchetti appartenenti alla stessa sorgente e destinazione seguono tutti lo stesso percorso.

Ogni strato inferiore di un'architettura di rete tratta la PDU dello strato superiore come una unità di informazione indipendente a cui aggiungere solo l'intestazione, comportamento che è detto incapsulamento. I protocolli definiscono la struttura, la composizione e l'ordine dei pacchetti.

Il *modello ISO/OSI* [3] è uno standard per le architetture di rete che prevede una struttura logica a strati suddivisa in 7 livelli (o layer). Il modello si presenta sotto forma di pila (o stack) (vedere Figura 2.2), nella quale i livelli, partendo dall'alto, sono così disposti:

- *applicativo*, è l'interfaccia tra utente e macchina e fornisce ai processi applicativi i mezzi per accedere all'architettura;

- *presentazione*, trasforma i dati delle applicazioni in un formato standard e risolve i problemi di compatibilità;
- *sessione*, assicura alle entità di presentazione un supporto per la comunicazione, per farlo struttura e sincronizza lo scambio dei dati legandoli ad una sessione;
- *trasporto*, stabilisce una connessione logica tra le entità di sessione e serve a provvedere ad un servizio di trasporto universale associato al suo strato inferiore;
- *rete*, fornisce i mezzi per instaurare, mantenere e abbattere le connessioni di rete;
- *collegamento*, fornisce i mezzi funzionali e procedurali per il trasferimento di unità dati attraverso una connessione di rete;
- *fisico*, fornisce i mezzi meccanici, fisici e funzionali per attivare e disattivare le connessioni fisiche. Provvede inoltre al trasferimento dei byte e si occupa della parte hardware.

Il *modello TCP/IP* è un'altra architettura di rete che negli anni si è affermata come standard implementativo di Internet, a differenza del modello ISO/OSI che risulta teorico [4]. Il TCP/IP (RFC-1180 [5]) semplifica il modello ISO/OSI accorpando alcuni livelli tra di loro (vedere Figura 2.2). Esso è composto da una selezione di protocolli esistenti, per esempio IP, TCP, ecc.

Internet, e più in generale una rete TCP/IP, adotta principalmente due protocolli di trasporto: TCP e UDP. Il primo è un servizio di trasporto connection-oriented pensato per comunicazioni affidabili e per garantire la completezza dell'informazione; il secondo è un servizio connectionless pensato per essere più semplice e minimale.

Come definito in [6], i dispositivi collegati a una rete, in particolare Internet, che offrono servizi applicativi sono definiti *hosts* o *end systems*.

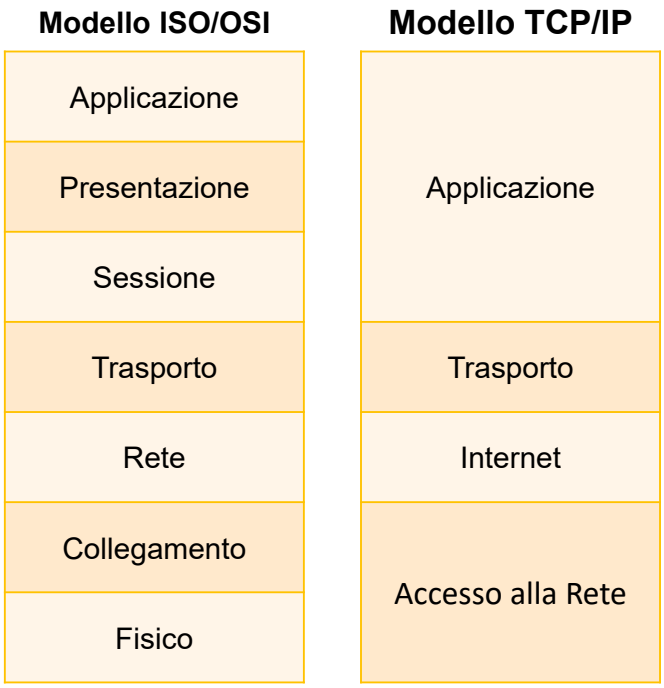


Figura 2.2: Modelli ISO/OSI e TCP/IP.

2.1.1 Transmission Control Protocol

Il *Transmission Control Protocol* (TCP) è uno dei protocolli standard appartenenti al livello di trasporto del modello ISO/OSI e del modello TCP/IP (vedere Figura 2.2) ed è descritto in RFC-793 [7].

Il livello di trasporto, come accennato in precedenza e descritto più approfonditamente in [6], ha il compito di provvedere alla comunicazione logica tra processi end-to-end di due entità. I messaggi del livello applicativo passano a quello di trasporto che si occupa di dividerli per rispettare la dimensione massima del pacchetto che è stata concordata per questo livello; il contenuto del messaggio viene poi utilizzato come payload al quale viene aggiunta l'intestazione del livello di trasporto. Il pacchetto nel layer di trasporto, che si definisce segmento, dopo essere stato generato viene passato al livello di rete che lo incapsula in un altro pacchetto e lo invia a destinazione.

Il livello di trasporto non interagisce direttamente con i processi applicativi, ma attraverso un *socket*, ossia un'interfaccia software che consente la comunicazione tra processi. Un host può gestire contemporaneamente più connessioni tra processi diversi, i servizi che interagiscono con il livello 4 permettono di gestire questa eventualità condividendo i canali trasmissivi dell'host. Questo comporta la necessità di gestire due casistiche:

- una in cui ci sono diversi messaggi applicativi provenienti da più socket che devono condividere il canale in uscita, questi vengono raccolti al livello di traposto che li imbusta in un segmento e li passa al livello di rete. Nell'intestazione del protocollo di trasporto ci sono le informazioni necessarie a identificare l'associazione con il processo, questa operazione prende il nome di *multiplazione*.
- un'altra in cui diversi segmenti arrivano allo stesso host e devono andare in socket differenti, in questo caso il protocollo elaborando le informazioni presenti nell'intestazione può indirizzare i vari messaggi attraverso il giusto socket. Questa operazione prende il nome di *demultiplazione*.

Queste funzionalità sono permesse grazie alle informazioni presenti nell'intestazione del segmento al cui interno sono presenti due campi: uno che descrive la porta di origine e uno che descrive quella di destinazione. Gli indirizzi IP e le porte utilizzate degli host coinvolti nella connessione, insieme al protocollo di trasporto utilizzato, permettono di indentificare univocamente un processo o più precisamente il suo *socket*.

Struttura del segmento

La struttura di un segmento TCP è composta come riportato in Figura 2.3.

I campi delle *porte* identificano il processo. Essendo descritte su 16 bit possono assumere valori tra 0 e 65535. Quelle che vanno da 0 a 1023 sono porte standard (o well-known ports) e sono di norma utilizzate per i servizi internet lato server. L'intervallo di porte tra 1024 e 49151 comprende porte registrate (o registered ports) e porte dedicate a servizi client. Le porte rimanenti sono invece definite dinamiche e vengono utilizzate in modo casuale dal client per associare un processo locale durante una connessione ad un server. La lista delle porte è stata definita e mantenuta aggiornata da ICANN[2]/IANA[3] facendo riferimento al RFC-6335 [8].

Il *numero di sequenza* identifica il numero del primo byte presente nel payload del segmento corrente, riferito al numero complessivo di byte inviati fino a quel momento in tutta la comunicazione su quel canale. Nel caso il flag SYN sia impostato a 1 questo campo assume un significato differente e identifica il numero iniziale della sequenza o initial sequence number (ISN) e il primo byte di dato sarà $ISN + 1$.

[2]Internet Corporation for Assigned Names and Numbers.

[3]Internet Assigned Numbers Authority.

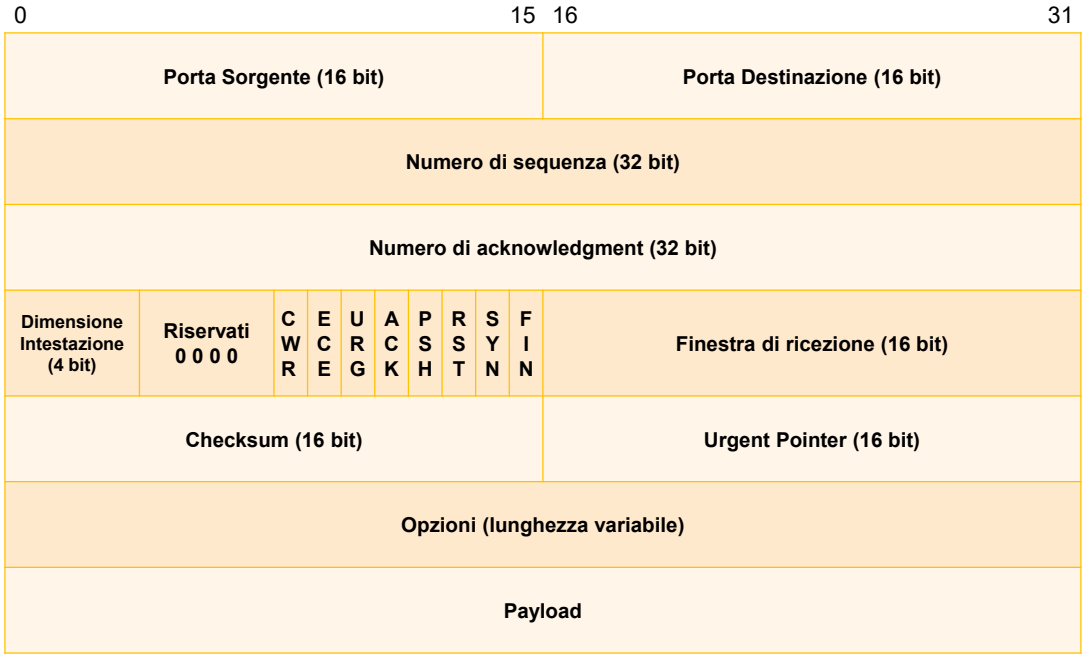


Figura 2.3: Struttura di un segmento TCP.

Il *numero di acknowledgement* identifica il prossimo byte che l'host si aspetta di ricevere nel caso il flag ACK sia impostato a 1. Un host effettua l'acknowledgement fino al primo byte mancante nel flusso cumulando i byte consecutivi precedentemente ricevuti, ragione per cui si definisce cumulative acknowledgment.

La *dimensione dell'intestazione* indica dopo quanti byte dall'inizio dell'intestazione, incluse le opzioni, comincia il payload.

I *bit riservati* devono essere impostati a zero e vengono riservati per utilizzi futuri del protocollo.

Il campo *flag* può assumere diversi significati in base al bit impostato:

- URG indica che il campo urgent pointer è valido;
- ACK indica che il campo numero di acknowledgement è valido;
- PSH indica all'host che deve passare il segmento direttamente al livello superiore senza memorizzarlo nel buffer;
- RST, SYN, FIN vengono utilizzati per aprire e chiudere una connessione;
- ECE, CWR sono stati introdotti successivamente, tramite RFC-3168 [9], per permettere una notificazione esplicita di congestione o Explicit Congestion Notification (ECN).

La *finestra di ricezione* viene utilizzata per controllare il flusso, questo campo indica la dimensione del numero di byte che l'host mittente è in grado di accettare a partire da dopo il numero di acknowledgment. Tale numero è legato allo spazio disponibile nel buffer di rete per la connessione, infatti, nel caso in cui vengano inviati più dati di quelli che si possono ricevere, essi verranno scartati. Durante la connessione questo valore può essere regolato per incoraggiare o scoraggiare l'invio di più informazioni.

Il campo *cheksum* viene utilizzato per controllare la validità del segmento facendo il complemento a uno del payload con l'aggiunta di alcuni dei campi dell'header.

Il campo *urgent pointer* indica la posizione di un dato urgente a partire dal numero di sequenza. Questo campo viene elaborato solamente se il flag URG è abilitato e serve a indicare a un host che nel segmento è presente un'informazione che deve essere elaborata in modo asincrono il prima possibile. Questo però solamente dopo aver verificato che non ci siano altri messaggi urgenti da elaborare prima.

Il campo *opzioni* ha dimensione variabile tra 0 e 40 byte ed è utilizzato per specificare informazioni aggiuntive sul segmento. Alcune delle opzioni disponibili sono:

- *Maximum Segment Size* (MSS), serve per notificare la dimensione massima del segmento che l'host mittente è in grado di elaborare. Questa opzione viene utilizzata solo durante l'apertura della connessione, quando il flag SYN è impostato a 1;
- *Timestamps Option* (TSopt), indica il valore corrente dell'orario dell'host mittente, questa opzione è stata introdotta nel RFC-1323 [10].

Gestione della connessione

Il TCP per instaurare e chiudere una connessione descrive delle procedure specifiche per poter scambiare alcuni parametri preliminari con l'host ricevente.

L'apertura di una comunicazione per lo scambio di dati tra due host (A e B) viene stabilita tramite il three-way handshake (vedere Figura 2.4a), osservando i seguenti passaggi:

1. l'invio di un segmento da A a B che ha il flag SYN impostato a 1, la grandezza della finestra di ricezione impostata, il numero di sequenza iniziale generato casualmente, il campo opzioni utilizzato per indicare la dimensione massima del segmento;
2. la risposta di B ad A avviene tramite un segmento simile a quello ricevuto, con la differenza che i campi sono compilati con i valori di B, il flag ACK è impostato a 1 e il campo Acknowledgment conferma il primo numero della sequenza di A;
3. A conferma i valori tramite un segmento con il flag ACK impostato a 1 e subito dopo può cominciare lo scambio di informazioni.

Al contrario dell'apertura, la chiusura della connessione avviene per canale, quindi è possibile chiudere la comunicazione in una sola direzione (vedere Figura 2.4b), per esempio A-B:

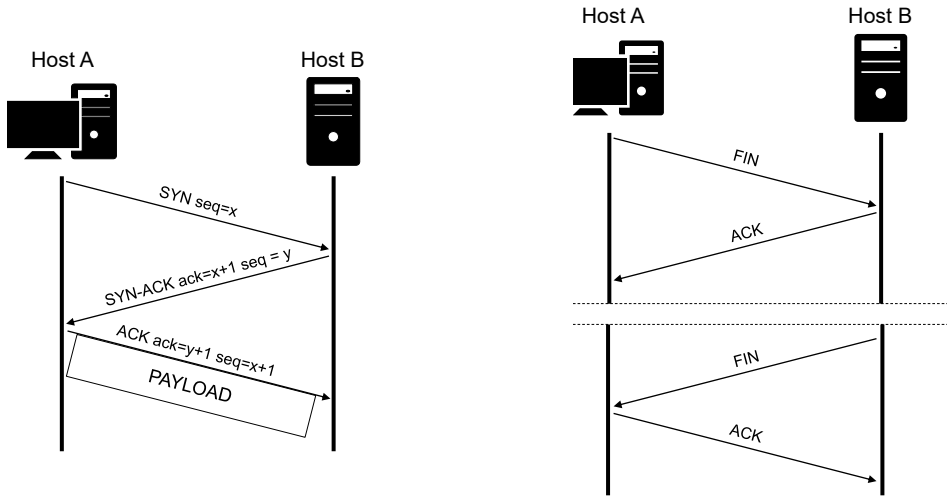
1. A, che vuole chiudere la connessione, manda un segmento con flag FIN impostato a 1.
2. B conferma la chiusura tramite un segmento con flag ACK impostato a 1.

Questa operazione verrà eseguita invertendo gli host nel momento in cui si vuole chiudere anche l'altra comunicazione (B-A).

Affidabilità della comunicazione

Come spiegato da [6], il servizio di Internet a livello di rete (protocollo IP) non è affidabile: esso non garantisce la consegna di un pacchetto e per questo viene definito un servizio best-effort. Il protocollo TCP invece, per offrire maggiori garanzie nella comunicazione, adotta tecniche di rilevamento di errore e di ritrasmissione, per questo rientra tra i protocolli di trasferimento di dati affidabili.

Il protocollo TCP utilizza un sistema di numerazione per tracciare i byte che invia, basato su numeri di sequenza, e uno di notifiche per ottenere un feedback dal destinatario. L'invio delle notifiche di risposta necessita che la comunicazione sia bidirezionale affinché l'host possa rispondere. Il protocollo TCP, utilizzando una connessione bidirezionale simultanea, detta full-duplex, ha le caratteristiche necessarie per implementarlo; inoltre sfruttando questa caratteristica è possibile inviare, insieme alle notifiche di risposta, anche delle informazioni legate al flusso del



(a) Apertura. (b) Chiusura.

Figura 2.4: Apertura e chiusura di una connessione TCP.

canale nell'altra direzione, questa operazione viene detta piggyback. Il funzionamento del livello di rete non garantisce l'arrivo ordinato dei segmenti inviati, ma grazie al numero di sequenza è possibile ricostruire il giusto flusso della connessione anche in presenza di pacchetti ritrasmessi o arrivati fuori sequenza.

Il rilevamento di errore avviene in fase di ricezione di un segmento. L'host ricevente utilizzando gli stessi campi impiegati dall'host mittente calcola il checksum del pacchetto e lo confronta con quello presente nell'intestazione del segmento. Nel caso i due complementi non siano uguali viene *rilevato un errore* e il pacchetto viene scartato.

La ritrasmissione di un pacchetto avviene in presenza di un rilevamento di un errore oppure nel caso venga rilevata la mancanza di una notifica positiva dopo l'invio di un messaggio, anche detta *acknowledgment*. L'host mittente, dopo aver inviato un pacchetto, deve conservarlo fino a quando non ottiene la conferma di ricezione, questo avviene perchè l'host deve poterlo ritrasmettere in caso di errore. Il numero dell'acknowledgment nel pacchetto di risposta viene confrontato con il numero di sequenza del segmento inviato, nel caso non siano compatibili viene rilevato un errore.

I *protocolli a finestra scorrevole* utilizzano i principi appena descritti per implementare una comunicazione affidabile, ottimizzando però l'invio dei pacchetti. Questo è reso possibile introducendo i concetti di *finestra di trasmissione* e *finestra di ricezione*: il primo utilizzato dal mittente per tenere traccia dei segmenti inviati non ancora confermati e quelli nuovi che possono essere inviati; il secondo, come accennato in precedenza nella Sezione 2.1.1, serve per indicare la quantità di pacchetti che il destinatario può ricevere.

Quando l'host riceve un acknowledgment di un pacchetto presente nella finestra, questa scorre fino al segmento con il numero di sequenza confermato lasciando spazio a nuovi pacchetti da poter trasmettere, vedere Figura 2.5. Siccome un host effettua la conferma fino al primo byte mancante nel flusso, la ricezione di un certo acknowledgment garantisce che tutti i byte precedenti siano stati effettivamente ricevuti, questa scelta viene definita *cumulative ACK*. L'host ricevente può controllare il flusso variando la dimensione della finestra di ricezione comunicandolo nell'intestazione dei segmenti come descritto in 2.1.1 .

Il protocollo TCP utilizza dei timer per determinare quando considerare perso un segmento inviato, se al suo scadere non si è ottenuta una notifica di ricezione ritrasmette tutti i byte non ancora confermanti presenti nella finestra di trasmissione, questo timer viene detto *retransmission timeout* (RTO). Il RFC-6298 [11] descrive l'algoritmo per determinare la durata del RTO, esso si basa sul Round Trip Time (RTT), ovvero il tempo che intercorre tra l'invio di un pacchetto e la ricezione del suo acknowledgement all'host mittente.

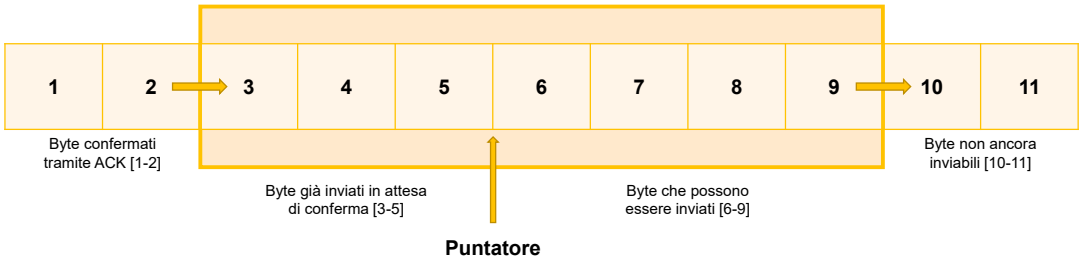


Figura 2.5: Finestra di trasmissione scorrevole.

Essendo il RTT un valore variabile dovuto a diversi fattori della rete, viene campionato durante l'intera comunicazione per essere utilizzato nel calcolo del RTO. La sua misurazione viene detta SampleRTT e facendone una media temporale è possibile ottenerne una stima più realistica:

$$EstimatedRTT = (1 - \alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$

dove α ha come valore raccomandato 0,125 e 1 secondo come valore iniziale di $SampleRTT$.

Come spiegato in precedenza la rete Internet è best-effort, per questo motivo l'effettiva capacità trasmissiva può cambiare durante la trasmissione. Questo tipo di comportamento può incidere negativamente sul calcolo della media del RTT, per questo motivo è importante conoscere anche la sua variabilità calcolandone la deviazione standard:

$$DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT|$$

dove β ha come valore suggerito 0,25.

Grazie a questi valori è possibile ottenere un valore di timeout:

$$RTO = EstimatedRTT + 4 \cdot DevRTT$$

Nel RFC-6298 [11] viene indicato che il timer debba partire dopo l'invio di un segmento cosicché scada eventualmente dopo RTO secondi. Quando l'host mittente riceve un ACK il timer viene resettato e fatto ripartire, se tutti i dati sono stati confermati il timer viene spento. Nel caso il timer scada, il pacchetto con il numero di sequenza minore non ancora confermato viene ritrasmesso e il timer viene fatto ripartire raddoppiandone il tempo.

Principi del controllo di congestione

Si può definire *congestione* quando il traffico dei pacchetti si avvicina alla capacità totale dell'infrastruttura di rete. Questo può capitare quando molte sorgenti inviano più informazioni di quanto i nodi, attraverso i quali passano, siano in grado di gestire. Gli effetti di una congestione sulla rete possono portare a ritardi, creazione di code e perdite di pacchetti comportando quindi anche la possibilità che le notifiche di ricezione non arrivino a destinazione, considerando quindi persi pacchetti in realtà consegnati.

Il protocollo TCP, per il controllo della congestione, implementa quattro algoritmi ideati da Jacobson e descritti nel RFC-5681 [12]. Questi algoritmi descrivono alcuni comportamenti che l'host deve assumere per provare a determinare e gestire una possibile congestione della rete. TCP si appoggia all'utilizzo di alcune variabili per controllare l'evoluzione della connessione, tra queste ci sono la finestra di congestione e un valore di soglia.

Gli algoritmi sono:

1. *Slow Start*, viene utilizzato quando una nuova connessione viene stabilita o quando un pacchetto viene dichiarato perso successivamente allo scadere del timer RTO. L'algoritmo

descrive come l'host mittente debba determinare la massima capacità della connessione individuando il valore della finestra di congestione che permetta di evitare l'insorgere di una congestione. RFC-3390 [13] suggerisce come valore iniziale per la finestra di congestione 1 MSS (vedi Sezione 2.1.1), successivamente per ogni ACK correttamente ricevuto il suo valore viene raddoppiato.

2. *Congestion Avoidance*, regola l'ampiezza della finestra di congestione e descrive il comportamento quando il suo valore raggiunge la soglia impostata dallo Slow Start. In questa fase la finestra, al posto di raddoppiare, viene aumentata di 1 MSS per ogni ACK correttamente ricevuto. Come per lo slow start, nel caso di perdita, il valore della finestra viene impostato a 1 MSS, senza però toccare il valore di soglia.

Il protocollo TCP quando riceve un segmento fuori sequenza con numero maggiore rispetto a quello atteso invia un ACK duplicato, ovvero una notifica positiva che conferma lo stesso numero di sequenza di un'altra notifica. Nel caso invece il numero di sequenza riempia un vuoto lasciato da segmenti fuori sequenza, viene inviato un ACK con il prossimo byte atteso. Questo sistema permette di individuare situazioni in cui una semplice ritrasmissione potrebbe ripristinare la sequenza del flusso e non rallentare eccessivamente la connessione. Il protocollo TCP dall'implementazione Tahoe introduce un altro algoritmo che sfrutta questo comportamento:

3. *Fast Retransmit*, come descritto prima serve ad eseguire un recupero dopo che è stata rilevata una singola perdita ed entra in azione quando vengono ricevuti 3 ACK duplicati. Durante questa fase l'host mittente dimezza il valore della soglia e ritrasmette il primo segmento della finestra non ancora confermato, successivamente rimane in attesa di tutti gli eventuali ACK duplicati fino a quando non ne arriva uno nuovo. Da quel momento ricomincia la fase Slow Start reimpostando la finestra di congestione a 1 MSS.

In una nuova implementazione del TCP denominata Reno, viene introdotto un nuovo algoritmo che permette di rendere più efficiente il recupero andando a modificare il comportamento del fast retransmit:

4. *Fast Recovery*, dopo aver dimezzato la soglia imposta il valore della finestra di congestione allo stesso valore maggiorato di 3 MSS. Durante l'attesa degli ACK duplicati per ognuno di essi la finestra di congestione viene aumentata ancora di 1 MSS fino all'arrivo di un ACK non duplicato. Al posto di ritornare nella fase Slow Start, si riparte dalla fase Congestion Avoidance, questo permette di non azzerare la finestra di congestione.

Equità

Il TCP prevede che, nel caso ci siano più connessioni sullo stesso link, la capacità trasmissiva venga equamente suddivisa tra tutte le connessioni. Per esempio, se la capacità è di R bps[4] e ci sono K connessioni, ogni connessione dovrebbe poter utilizzare solo R/K bps. Dal momento che le connessioni sono dinamiche è difficile mantenere equità in ogni momento, per questo la capacità trasmissiva viene continuamente aggiustata per raggiungere una media. Questo è reso possibile grazie all'incremento adattivo e il decremento moltiplicativo implementati per la gestione delle congestioni, vedi Sezione 2.1.1. All'avvicinarsi della capacità della rete, alcuni pacchetti andranno persi riducendo quindi la finestra di congestione lasciando spazio ad altre connessioni.

2.1.2 Secure Socket Layer e Transport Layer Security

Il protocollo *Secure Sockets Layer* (SSL), come descritto in [6], viene utilizzato per aggiungere riservatezza, autenticazione e integrità a una connessione su reti TCP/IP. Questo protocollo, come mostrato nella Figura 2.6, opera tra il livello applicativo e quello di trasporto, fornendo un'interfaccia simile a quella esposta dal protocollo TCP, aggiungendo un livello di astrazione che incapsula i messaggi applicativi offuscati tramite algoritmi crittografici.

[4] bit per second.

Figura 2.6: Modello TCP/IP con livello SSL/TLS.

Caratteristiche principali

Esistono principalmente due tipologie di crittografia, quella a chiave pubblica e quella a chiave simmetrica. SSL le utilizza entrambe in diverse fasi del protocollo.

La *crittografia a chiave pubblica* o asimmetrica è una tipologia di algoritmi che fa uso di una coppia di chiavi: una privata posseduta solo dal proprietario e una pubblica liberamente distribuita e utilizzabile da tutti. Questi algoritmi si basano su funzioni matematiche unidirezionali, difficilmente invertibili, che permettono di utilizzare una delle due chiavi per cifrare, ottenendo così un messaggio offuscato che possa essere decifrato unicamente utilizzando l'altra chiave e viceversa. La *crittografia a chiave simmetrica* è anch'essa un sistema a chiavi che si serve però di un'unica chiave sia per cifrare che per decifrare un messaggio.

Il protocollo SSL per ottenere la *riservatezza* della connessione e garantire che sia privata utilizza un sistema di crittografia concordato tra gli host durante l'apertura della connessione. Gli algoritmi crittografici utilizzati sui messaggi applicativi sono solitamente quelli a chiave simmetrica (es. RC4, DES, 3DES, AES, ecc.). Invece quelli a chiave asimmetrica (es. RSA, DH o Fortezza-KEA) vengono utilizzati per effettuare lo scambio delle informazioni preliminari per avviare la connessione, una di queste è la chiave simmetrica utilizzata per la cifratura dei messaggi.

Per garantire l'identità degli host vengono utilizzati dei certificati, ovvero documenti elettronici che permettono di verificare l'associazione di un insieme di informazioni ad un soggetto. I certificati vengono rilasciati da un ente riconosciuto, detto *autorità di certificazione* (CA), che autentica le informazioni in esso presenti con una firma digitale[5]. Il protocollo SSL generalmente per i certificati utilizza lo standard X.509 descritto in RFC-5280 [14] per conservare e validare la chiave pubblica legata all'identità dell'host. Insieme a un algoritmo di autenticazione a sfida, può essere utilizzato per garantire che l'utente si stia connettendo veramente al soggetto desiderato e che non ci siano intromissioni nella connessione.

[5]Metodo matematico legalmente attendibile per dimostrare l'autenticità di un documento. Solitamente una firma digitale è il risultato di una cifratura a chiave pubblica di un riassunto, detto digest, del messaggio da validare.

L'*autenticazione tramite sfida asimmetrica* è un algoritmo che consiste nella risoluzione di una sfida utilizzando una coppia di chiavi asimmetriche per validare la propria identità. Con questa tecnica è possibile autenticare solamente il possessore della chiave privata. I passaggi per l'autenticazione possono essere riassunti nei seguenti punti:

- il soggetto B, a cui il soggetto A vuole connettersi, invia la propria chiave pubblica;
- A utilizzando un *nonce*, ovvero un numero casuale utilizzabile una sola volta, lo invia cifrato con la chiave pubblica di B a quest'ultimo;
- B, che è l'unico a possedere la chiave privata, la usa per decifrare il messaggio e inviarlo ad A come prova di aver risolto la sfida;
- A confronta il messaggio con il nonce e nel caso siano uguali B risulta autenticato.

Per preservare l'*integrità* e l'autenticazione di un messaggio si possono utilizzare tecniche di validazione tramite funzioni crittografiche di hash. Una simile funzione permette, partendo da un messaggio originale di dimensione variabile, di ottenerne uno nuovo di dimensione fissa detto digest, la cui caratteristica principale è il legame unidirezionale tra i due messaggi.

Questo tipo di funzione possiede ulteriori importanti caratteristiche tra cui essere difficilmente invertibile[6] ed essere resistenti alle collisioni[7]. Queste peculiarità evitano l'eventualità di ottenere lo stesso digest da due messaggi diversi; inoltre garantiscono che i due digest siano completamente differenti l'uno dall'altro anche qualora il messaggio originale venga minimamente modificato.

Una tipologia di digest che garantisce l'integrità e l'autenticazione è detta Message Authentication Code (MAC), questo è reso possibile grazie all'utilizzo di una doppia funzione di hash.

Struttura del record SSL

Il protocollo SSL, descritto in RFC-6101 [15], è composto da due livelli che si vanno ad inserire tra il livello applicativo e quello di trasporto, come si può vedere nella Figura 2.6. SSL utilizza come base un protocollo affidabile, per esempio il TCP, ereditando quindi le caratteristiche di quest'ultimo senza però doverle gestire.

I due livelli di SSL sono posizionati uno sopra l'altro, il primo a contatto con il protocollo di trasporto, detto *protocollo Record* e l'altro composto da più protocolli posizionato sopra. Le varie tipologie del livello superiore sono:

- *livello di applicazione*, si tratta dello stesso livello del modello TCP/IP accennato nella Sezione 2.1, nel protocollo SSL i messaggi applicativi sono trasportati direttamente al protocollo Record per essere frammentati, compressi e cifrati (il tutto spiegato nel paragrafo successivo). I dati provenienti dal livello superiore vengono quindi trattati come trasparenti e inoltrati al livello Record;
- *protocollo Alert*, utilizzato per notificare messaggi, anomalie e problemi;
- *protocollo Change Cipher Spec*, viene utilizzato per segnalare e gestire il cambio di parametri di sicurezza utilizzati;
- *protocollo Handshake*, utilizzato per gestire l'apertura della connessione e lo scambio dei suoi parametri.

[6] Partendo dal digest non sia possibile risalire al messaggio originale.

[7] Sia difficile trovare due messaggi di partenza che diano lo stesso digest come output.

Il *protocollo Record* si occupa di lavorare i dati provenienti dai livelli superiori e applicare frammentazione, compressione, cifratura e viceversa per i pacchetti in entrata. Le operazioni vengono effettuate in un determinato ordine per non creare vulnerabilità e poter cifrare anche i byte di verifica (vedere Figura 2.7). In ordine sono:

1. frammentazione, è utilizzata per suddividere i dati applicativi in dimensioni compatibili con quelle trasportabili in un record SSL;
2. compressione, è un passaggio opzionale che viene effettuato solo se specificato, utilizzando degli algoritmi concordati in fase di apertura della connessione;
3. calcolo del MAC, in questa fase viene applicata una funzione crittografica di hash al messaggio unito a una chiave simmetrica concordata tra gli host. Il digest ottenuto verrà poi utilizzato come verifica dal ricevente;
4. padding, viene aggiunto del contenuto ignorabile per raggiungere la dimensione massima del pacchetto, questa operazione è necessaria per evitare che il pacchetto sia riconoscibile dalla lunghezza del contenuto;
5. cifratura, viene applicata la funzione crittografica scelta per offuscare il contenuto del pacchetto;
6. aggiunta dell'intestazione, come ultimo passaggio viene aggiunto l'header con le informazioni principali del record, tra cui il tipo di contenuto e la sua lunghezza.

Il pacchetto così costruito può essere poi inviato al destinatario e quest'ultimo, dopo averlo ricevuto e decifrato, dovrà isolare il solo messaggio e calcolare il proprio risultato del MAC e confrontarlo con quello contenuto nel pacchetto. Nel caso in cui non siano uguali il ricevente dovrà inviare un messaggio di errore fatale (bad_record_mac) all'host mittente tramite il protocollo Alert.

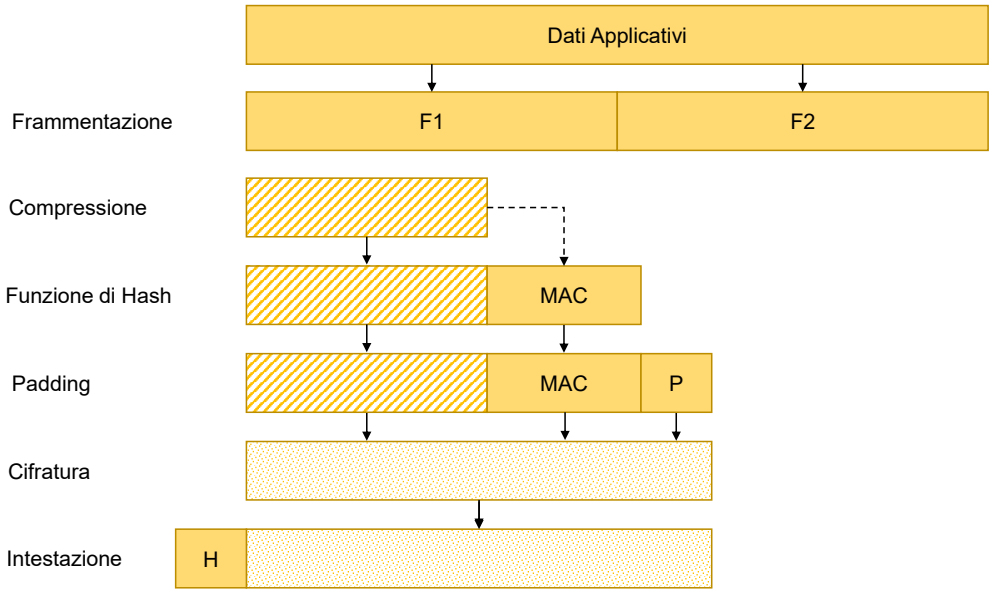


Figura 2.7: Creazione di un pacchetto SSL tramite il protocollo Record.

Il *protocollo Alert* trasporta i messaggi di allarme abbinati ad un livello di priorità, i principali sono "warning" e "fatal". Questo tipo di messaggi essendo incapsulati dentro il protocollo Record sono anch'essi cifrati e compressi, ogni messaggio è composto da due byte. Se il livello del contenuto è fatale implica l'immediata chiusura della connessione e di tutte quelle ad essa legate.

Il *protocollo Change Cipher Spec* consiste in un singolo messaggio di 1 byte con valore 1 ed ha come unica funzione quella di confermare che la negoziazione dei parametri di sicurezza sia andata a buon fine e che gli stessi verranno utilizzati dal messaggio successivo.

Il *protocollo Handshake* permette al client e al server di autenticarsi a vicenda, negoziare gli algoritmi da utilizzare e stabilire le chiavi crittografiche. I record prodotti da questo protocollo sono i primi ad essere scambiati tra gli host per aprire una connessione e creare un canale sicuro. Questo protocollo può inviare diverse tipologie di messaggio, le quali verranno analizzate nel prossimo paragrafo dedicato all'apertura della connessione.

Handshake e chiusura

L'apertura di un canale SSL avviene, come per il protocollo TCP, dal client verso il server, e può verificarsi in due modi tramite negoziazione o tramite ripristino.

La prima modalità prevede lo scambio di informazioni tra i due host per concordare i parametri di sicurezza da utilizzare, la seconda invece, grazie ad un identificativo, permette un'apertura più rapida utilizzando dei parametri precedentemente concordati.

Il ripristino sfrutta l'utilizzo di un identificativo denominato *session id* assegnato alla comunicazione, esso viene associato ai parametri scambiati durante l'apertura della connessione e permette di riutilizzarli fornendolo al server.

Le tipologie di algoritmi che possono essere utilizzati dagli host durante una comunicazione sono descritte nelle *ciphersuite* e vengono comunicati in fase di apertura della connessione. Una ciphersuite è composta da un algoritmo di cifratura, uno per lo scambio di chiavi e uno per il calcolo dell'integrità.

Due esempi di ciphersuite sono: `SSL_NULL_WITH_NULL e SSL_RSA_WITH_3DES_EDE_CBC_SHA`. Il primo è una ciphersuite fittizia che non garantisce nessuna sicurezza; la seconda invece indica che il protocollo utilizzato è TLS, come algoritmo per lo scambio di chiavi e autenticazione RSA, per la cifratura 3DES EDE[8] CBC[9] e come funzione crittografica di hash SHA1.

L'apertura tramite negoziazione avviene tramite le seguenti operazioni (vedere Figura 2.8a):

1. il client invia un *Client Hello*, ovvero un record contenente la versione del protocollo supportata da esso (la più recente possibile), un nonce generato dal client, l'id di sessione impostato a zero, la lista delle ciphersuite e i metodi di compressione supportati;
2. il server risponde con un *Server Hello*, questo record è simile a quello del client e contiene gli stessi campi, ma con le informazioni che verranno poi utilizzate per la comunicazione. La versione del protocollo è la più recente supportata dal server, ma che non superi quella mandata dal client. La ciphersuite e i metodi di compressione sono singoli e vengono scelti tra quelli suggeriti dal client. Anche il server genera un nonce che deve essere indipendente da quello inviato dal client;
3. il server subito dopo il messaggio di Hello, opzionalmente può inviare un record *Server Certificate* che contiene la catena dei certificati X.509 associati al server. Successivamente, nel caso invece il server non abbia un certificato o venga utilizzato un sistema di scambio delle chiavi a parametri fissi (es. RSA), può opzionalmente inviare un record *Server Key Exchange Message* contenente i parametri per lo scambio delle chiavi;
4. il server sempre opzionalmente può richiedere il certificato X.509 del client tramite il record *Certificate Request* per autenticare il client. Infine, per segnalare che il server ha terminato l'invio dei suoi messaggi, invia un record *Server Hello Done* e rimane in attesa delle risposte del client;

[8] Encrypt-decrypt-encrypt, è una modalità di applicazione di algoritmi crittografici sul dato. Un esempio applicato al 3DES è presente nel RFC 1851 https://tools.ietf.org/html/rfc1851.

[9] Cipher block chaining, è una modalità di funzionamento dei cifrari a blocchi. Vedere https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation.

5. il primo record che il client può inviare è opzionale ed è *Client Certificate*, questo messaggio viene inviato solamente se il server ha inviato un Certificate Request e contiene la catena dei certificati del client;

6. il client invia un record *Client Key Exchange Message* che, in base all' algoritmo di scambio di chiavi scelto, contiene i parametri per generare le chiavi.

7. nel caso il client abbia inviato il proprio certificato, invia un record *Certificate Verify* contenenti le informazioni esplicite necessarie al server per poter verificare i certificati del client;

8. sia il client che il server utilizzano i parametri scambiati e i nonce, ottenuti all'inizio della connessione, per generare un Master Secret (MS). Questo codice è il segreto condiviso tra i due host e viene utilizzato per generare le altre chiavi di cifratura necessarie per la comunicazione: due per lo scambio dei dati (una per ogni direzione della connessione TCP) e quelle relative al calcolo del MAC;

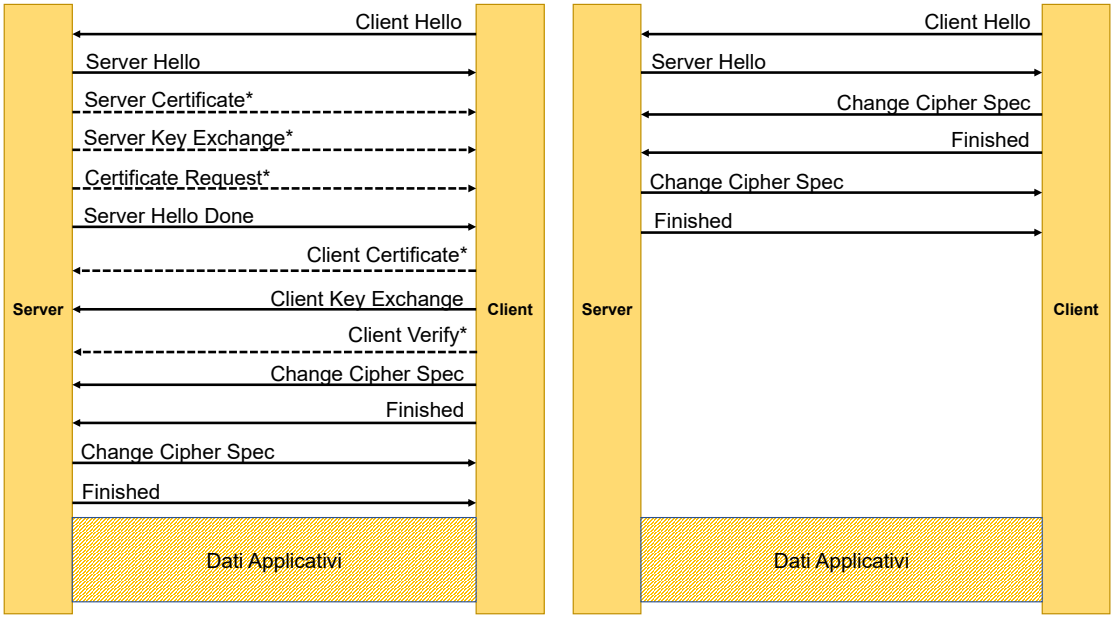
9. il client, tramite il protocollo Change Cipher Spec, comunica che ha calcolato i nuovi parametri della connessione ed è pronto ad utilizzarli.

10. il client immediatamente dopo il Change Cipher Spec invia un record *Finished*, questo è il primo vero record che utilizza i nuovi parametri di sicurezza e contiene un MAC di tutti i messaggi di handshake scambiati fino a quel momento;

11. anche il server a sua volta invia un messaggio Change Cipher Spec seguito da un record Finished con un MAC dei messaggi di handshake;

12. a questo punto la comunicazione è sicura e può cominciare lo scambio dei messaggi applicativi.

Lo scambio dei MAC relativi ai messaggi iniziali contenuti nei record Finished servono a proteggere da eventuali manomissioni.



(a) Handshake tramite negoziazione. (b) Handshake tramite ripristino.

Figura 2.8: Apertura di una connessione SSL tramite handshake (* sono passaggi opzionali).

Nel caso di un'apertura della connessione tramite ripristino, il numero delle operazioni è inferiore (vedere Figura 2.8b) e consistono nei seguenti passaggi:

1. il client invia un record Client Hello contenente l'identificativo di sessione che ha memorizzato da una connessione precedente con quel server, insieme invia anche un nuovo nonce cifrato tramite il vecchio master secret;

2. il server confronta l'identificativo con quelli che ha memorizzato e nel caso sia ancora salvato e considerato valido invia un Server Hello contenente lo stesso identificativo e un nuovo nonce anch'esso cifrato tramite il master secret. Nel caso il session id non fosse valido il server ne genererebbe uno nuovo e i parametri di sicurezza verrebbero rinegoziati;

3. il client se riceve lo stesso identificativo procede a ripristinare le informazioni legate alla precedente comunicazione, genera le nuove chiavi utilizzando il master secret già noto e invia un record Change Cipher Spec seguito da un Finished;

4. il server esegue le stesse operazioni e a sua volta invia un record Change Cipher Spec e un record Finished.

Come per il caso precedente i record Finished contengono i MAC dei primi messaggi della connessione per validare che non ci siano state interferenze.

La chiusura di una comunicazione sicura avviene tramite un record di tipo Alert, nel quale viene indicata la volontà di chiudere la connessione. L'autenticazione e l'integrità dei messaggi garantiscono che solo uno dei due host possa chiudere la connessione e non ci possano essere intromissioni.

Evoluzione da SSL a TLS

Il protocollo SSL, come raccontato in [16], è stato originariamente sviluppato da Netscape, la prima versione non ha mai visto la pubblicazione a causa di alcune imperfezioni ed è stata velocemente rimpiazzata dalla seconda (SSLv2), la quale invece viene distribuita, seppur ancora con qualche problema. Nella sua terza versione (SSLv3) il protocollo è migliorato e si introducono delle novità importanti in tema di sicurezza.

Nello specifico SSLv2, come spiegato in RFC-6176 [17], in alcuni casi riduceva la lunghezza della chiave e utilizzava MAC deboli, mentre la nuova versione impiega chiavi più lunghe e inizia ad adottare HMAC[10], ovvero una versione più robusta del MAC che utilizza funzioni crittografiche di hash (es. SHA e MD5) impiegate più volte con chiavi diverse per garantire che il digest non sia individuabile.

Viste le buone caratteristiche del protocollo l'organo internazionale IETF ha deciso di standardizzarlo creandone una nuova versione modificata, che si potrebbe definire SSLv3.1, rinominato poi Trasport Layer Security (TLS) descritto in RFC-2246 [18]. TLSv1.0 eredita tutte le funzionalità di SSL, ma non per questo i due protocolli sono compatibili; nonostante questo è possibile supportarli entrambi facendo poche modifiche. Il nuovo protocollo pone maggiore enfasi sugli algoritmi crittografici e di digest standard non proprietari.

Il protocollo TLS ha avuto alcune versioni nel tempo, dovute ad aggiornamenti per implementare e migliorare le funzionalità di sicurezza. La versione attuale più recente è v1.3 descritta in RFC-8446 [19], mentre le versione più vecchie della v1.1 sono state deprecate perché ritenute insicure.

[10] Keyed-Hash Message Authetication Code.

2.1.3 HyperText Transfer Protocol

HyperText Transfer Protocol (HTTP) è un protocollo applicativo definito in RFC-1945 [20], che prevede lo scambio di informazioni e risorse tra un client e un server, il primo le richiede, il secondo le fornisce. Per la comunicazione come protocollo di trasporto viene utilizzato il TCP, il server rimane in ascolto su una porta (solitamente la porta 80) in attesa di una richiesta dal client. Il protocollo HTTP si limita a definire il trasferimento delle risorse, senza memorizzare le diverse connessioni che un client effettua, per questo viene definito come protocollo senza memoria (*stateless*).

Come descritto in [6], il protocollo HTTP definisce la struttura dei messaggi e le modalità con cui vengono scambiati. Esso si basa su messaggi di richiesta (*REQUEST*) inviati dal client e messaggi di risposta (*RESPONSE*) inviati dal server.

Formato dei messaggi HTTP: Messaggi di richiesta

I *messaggi di richiesta* sono composti da quattro parti (vedere Figura 2.9): linea di richiesta, intestazione, riga vuota e corpo.

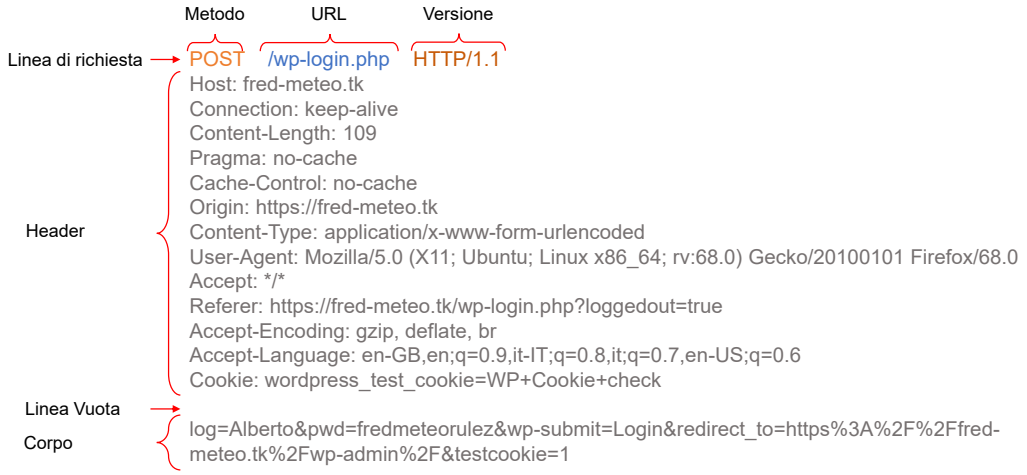


Figura 2.9: Esempio di un messaggio di richiesta.

Nella *Linea di richiesta* sono presenti:

- *metodo*: è un'informazione che può assumere diversi valori (es. GET, POST, HEAD, PUT, DELETE), identifica il tipo di richiesta e la tipologia di operazione che il server dovrà eseguire sulla risorsa;
- *Uniform Resource Locator* (URL): identificata una risorsa specifica in una rete, il suo formato è descritto in RFC-3986 [21].
- *versione*: indica la versione di HTTP in uso (es. HTTP/1.1).

L'*intestazione del messaggio* contiene informazioni necessarie per identificare il server su cui richiedere la risorsa, le tipologie supportate dal client e le informazioni legate al contenuto del corpo.

La *riga vuota* è composta da due caratteri ASCII[11]: Carriage Return (CR) e Line Feed (LF).

Il *corpo del messaggio* viene riempito in fase di richiesta solo quando il metodo è POST.

[11] ASCII: acronimo di American Standard Code for Information Interchange, codice per la codifica dei caratteri.

Formato dei messaggi HTTP: Messaggi di risposta

I *messaggi di risposta* sono anch'essi composti da quattro parti (vedere Figura 2.10): linea di stato, intestazione, riga vuota e il corpo.

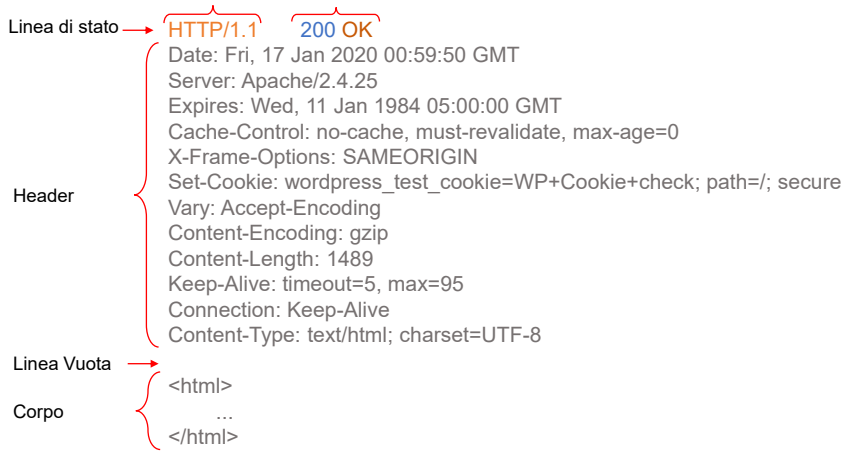


Figura 2.10: Esempio di un messaggio di risposta.

La *Linea di stato* contiene la versione del protocollo utilizzato e un codice. Quest'ultimo identifica lo stato della risposta in seguito ad una richiesta HTTP, ci sono diversi codici ed ognuno identifica una particolare situazione che potrebbe avvenire durante una comunicazione HTTP. I numeri dei codici sono codificati e raggruppati per tipologie, sono composti da tre cifre delle quali la prima ne identifica la categoria, in particolare:

- 1XX: codici informativi;
- 2XX: messaggi di successo, esempio `200 OK`;
- 3XX: messaggi che indicano una redirezione, per esempio `301 MovedPermanently`, l'oggetto richiesto è stato trasferito in modo permanente;
- 4XX: messaggi di errori relativi al client, esempio `404 NotFound`, la risorsa richiesta non è stata trovata;
- 5XX: messaggi di errori relativi al server, esempio `500 InternalServerError`, indica un errore generico del server.

Passaggio da HTTP/1.0 a HTTP/1.1

La prima versione del protocollo HTTP (HTTP/1.0) è stata largamente utilizzata insieme ad altri strumenti per comporre il World Wide Web (WWW). Con la crescita di internet e la sempre maggior adozione del protocollo sono diventati evidenti alcuni suoi limiti, tra cui ad esempio la mancanza di un supporto per il caching delle singole risorse, l'inefficienza dovuta al mancato riutilizzo delle connessioni TCP disponibili e gli scarsi meccanismi per la sicurezza. Per questo viene introdotto HTTP/1.1 descritto in RFC-2068 [22] e aggiornato in RFC-2616 [23].

Sicurezza in HTTP

Una delle caratteristiche dell'HTTP è quella di inviare i messaggi in chiaro e in formato ASCII. Questo permette ai malintenzionati di intercettare alcune informazioni sensibili che vengono scambiate durante la connessione. Inoltre HTTP, essendo *stateless*, non permette nessun controllo di autenticazione. Una delle soluzioni a questi problemi è l'utilizzo di un sistema di sicurezza da

applicare a HTTP come il protocollo TLS, componendo così HTTPS o HTTP over TLS descritto in RFC-2818 [24]. Grazie alle caratteristiche del protocollo TLS (vedere Sezione 2.1.2), il protocollo HTTP ottiene autenticazione, riservatezza e l'integrità per il server web. La porta 443 è solitamente la porta di riferimento per il protocollo HTTPS.

2.2 Statistiche e analisi del traffico

L'analisi del traffico di rete e dei suoi protocolli può essere effettuata tramite strumenti detti *sniffer*. Essi agiscono a livello di rete (vedi Sezione 2.1) catturando il traffico e processandolo; si parte da strumenti più basilari come tcpdump, fino ad arrivare ad altri più complessi come Wireshark o Tstat. Ognuno di questi strumenti è caratterizzato dalla capacità di riconoscere alcuni protocolli e presentarne le statistiche più salienti.

2.2.1 Tcpdump e Libpcap

Tcpdump[12], come spiegato sulla pagina ufficiale del progetto, è uno strumento da riga di comando per l'analisi dei pacchetti di rete che transitano sull'interfaccia di rete scelta. Nello specifico tcpdump permette di catturare e visualizzare il traffico di rete in tempo reale a schermo.

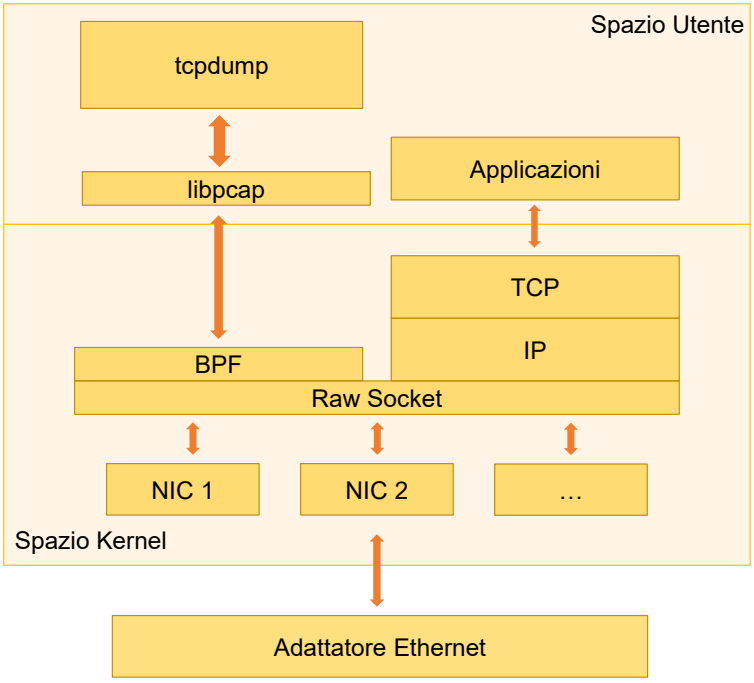


Figura 2.11: Schema libpcap e BPF.

Questo software offre la possibilità di applicare dei filtri al flusso presente sull'interfaccia per visualizzare solo alcuni di questi pacchetti, ciò è reso possibile dal supporto ai filtri BPF e dalla libreria libpcap su cui tcpdump è costruito.

Berkeley Packet Filter (BPF) è un'architettura generica per realizzare filtri di pacchetti direttamente nel kernel del sistema operativo. Nasce in ambiente Unix-like, con lo scopo di ridurre le tempistiche di analisi di rete potendo agire direttamente nello spazio kernel. Successivamente questa architettura è stata implementata anche per altre versioni di sistemi operativi.

[12] https://www.tcpdump.org

Nel kernel è stata aggiunta una macchina virtuale BPF connessa al *layer data link* che interpreta le direttive nel linguaggio BPF per creare i filtri da applicare. Come si può vedere nella Figura 2.11, il BPF non interagisce direttamente con lo stack di rete del sistema operativo (TCP/IP), ma su copie dei pacchetti non ancora elaborati (raw packet), il traffico infatti non viene deviato e continua a venire inoltrato nei livelli successivi (es. IP).

Libpcap è una libreria `C/C++` open source che fornisce un'interfaccia ad alto livello per interagire con i moduli del kernel che si occupano di catturare e filtrare i pacchetti, ovvero il BPF. Questa libreria è stata sviluppata dallo stesso team di tcpdump per poter separare lo strumento e le funzionalità di interazione con il sistema operativo. Infatti, libpcap è alla base della maggior parte degli strumenti di analisi di rete.

2.2.2 Tcptrace

Tcptrace[13] è uno strumento open source per l'analisi di traffico TCP da linea di comando. Diversamente da tcpdump questo software non cattura il traffico, ma lo può analizzare accettando come input dei file pcap (*Packet CAPture*) prodotti da altri strumenti che utilizzano libpcap.

Tcptrace può dividere l'output per ciascun flusso della connessione e calcolarne diversi tipi di statistiche, creare sommari e produrre grafici. Tra le statistiche rilevate che possono restituire un grafico ci sono:

- grafici di sequenze temporali;
- grafici sul throughput[14];
- grafici sul RTT;
- grafici sulla dimensione del segmento.

2.2.3 Tshark e Wireshark

Tshark[15] è un analizzatore di protocolli di rete da riga di comando distribuito con licenza open source. Anche questo software è costruito utilizzando la libreria libpcap, quindi può catturare traffico in tempo reale e interagire con pacchetti in formato pcap. Senza particolari opzioni si comporta come tcpdump e può eseguire catture filtrate grazie alla possibilità di interagire direttamente con il BPF.

Tshark integra diverse funzioni di ordinamento dei pacchetti e di analisi delle connessioni, infatti è in grado di ricostruire i flussi del traffico e riconoscerne i diversi protocolli (es. TCP, UDP, HTTP, TLS, DNS, etc.).

Tra le analisi che può effettuare ci sono:

- il conteggio del numero di pacchetti;
- il conteggio del numero totale di byte di un flusso;
- varie statistiche legate all'intera connessione (es. tempi di RTT e Throughput).

Wireshark[16] è la versione con un'interfaccia grafica di tshark.

[13] http://www.tcptrace.org

[14] Il throughput rappresenta la capacità effettiva di un canale di rete.

[15] https://www.wireshark.org/docs/man-pages/tshark.html

[16] https://www.wireshark.org

2.2.4 Tstat

Tstat[17] è un analizzatore di traffico che si presenta sia come strumento da riga di comando che come libreria scritta in `C` (libtstat). Nasce come evoluzione di tcptrace per supportare i ricercatori nell'automatizzazione della produzione di statistiche. Tstat infatti è in grado di riconoscere diversi protocolli applicativi moderni, misurarne le performance e le principali caratteristiche.

Questo programma può analizzare sia il traffico in real-time che il traffico di pacchetti in formato pcap. Esso è in grado di riunire i flussi bi-direzionali del TCP e ricavare analisi complessive sull'intera connessione, aggiungendo ulteriori misurazioni da poter analizzare. Per via della grande quantità di statistiche prodotte, tstat viene utilizzato come strumento di riferimento per l'implementazione di un altro software di analisi utilizzato in questo lavoro di tesi.

Sui flussi TCP completi, ovvero quelli sui quali è possibile riconoscere sia l'handshake che la chiusura della connessione (descritta nella Sezione 2.1.1), vengono prodotte principalmente le statistiche riportate nella Tabella A.1. Le 52 statistiche, escludendo quelle comprese tra la 40 e la 44, sono quelle ritenute, per l'obbiettivo del seguente studio, quelle più significative a livello TCP. Per fare riferimento a queste caratteristiche da qui in poi verranno definite *formato tstat*.

Per raggiungere l'obbiettivo di questa tesi è necessario selezionare uno strumento di analisi che possa estrapolare statistiche su traffico TCP. Nonostante tstat rappresenti uno strumento valido, si è deciso di procedere alla creazione di uno strumento dedicato. Tstat infatti, come detto in precedenza, esegue analisi anche su protocolli applicativi, operazione che risulta superflua in questo lavoro in quanto non si vuole eseguire *Deep Packet Inspection* (DPI).

2.2.5 Scapy

Scapy[18] è un programma open source scritto in python che permette di inviare, catturare, analizzare e forgiare pacchetti di rete. Questo strumento è basato su una versione wrapped[19] della libreria libpcap in python.

Una delle caratteristiche più interessanti di scapy è la possibilità di manipolare qualunque tipo di pacchetto di rete, anche quelli non validi o fuori standard. Questo è possibile grazie alla sua flessibilità che permette di estendere il software con l'aggiunta di nuovi moduli. Sarebbe infatti possibile implementare nuovi protocolli, manipolare il traffico e creare strumenti di analisi personalizzati.

Uno dei principi chiave di scapy è quello di eseguire solamente la decodifica del pacchetto senza fornirne un'interpretazione, lasciandola alle logiche che possono essere descritte dall'utente che personalizza lo strumento.

Scapy è stato pensato per essere estensibile senza necessariamente andare a modificare il codice sorgente, in quanto può essere utilizzato come una libreria per un nuovo programma in python.

Un'altra sua caratteristica è quella di supportare la cattura del traffico in real-time. Questa, insieme alle altre funzionalità, ha permesso di scegliere scapy come modulo fondamentale per la creazione dello strumento utilizzato nel presente studio.

2.2.6 Considerazioni relative allo studio

Come illustrato in questo Capitolo, tutti gli strumenti per l'analisi di rete analizzati fanno uso della libreria libpcap permettendo loro un accesso diretto ed efficiente al BPF. Questa caratteristica è

[17] http://tstat.polito.it

[18] https://scapy.net

[19] Wrapper nella terminologia informatica si riferisce a un'entità che incapsula e nasconde un'altra entità, in questo caso un software scritto in C/C++ reso compatibile con python.

condivisa anche con altri strumenti di rete che non sono stati presi in esame poichè presentano funzionalità simili a quelli già esposti.

È possibile notare come alcuni degli strumenti citati si fondino maggiormente su analisi più basilari, per esempio tcpdump (vedere Sezione 2.2.1), mentre altri molto su funzionalità più complesse, come Wireshark (vedere Sezione 2.2.3). La maggior parte di essi eseguono analisi DPI, ovvero analizzando anche il contenuto del pacchetto del livello di trasporto. Inoltre, durante lo studio preliminare della scelta dello strumento, si è notato come strumenti come Tstat (vedere 2.2.4) non siano in grado di lavorare con pacchetti TCP malformati, caratteristica presente nel traffico prodotto dagli scanner di vulnerabilità.

Per le ragioni sopra elencate si è scelto per il seguente studio di creare uno strumento di analisi di rete anch'esso basato sulla libreria libpcap che non eseguisse analisi DPI e che fosse modulare. Infatti lo strumento creato può essere modificato facilmente permettendo una maggiore flessibilità durante l'analisi. Per una panoramica delle sue funzionalità e di come è stato utilizzato all'interno dello studio si rimanda alla Sezione 4.3.

2.3 Scanner di vulnerabilità

Uno *scanner di vulnerabilità* è un software in grado di individuare tramite regole e automatismi le debolezze di un determinato sistema e/o rete.

Uno degli scopi principali di un software di analisi è quello di poter generare un report di valutazione o assessment che riporti lo stato del sistema. Invece l'insieme delle operazioni per eseguire un attacco informatico simulato e autorizzato si chiama *penetration test*, il quale permette di scoprire delle vulnerabilità seguendo il punto di vista di un potenziale attaccante.

Alcuni dei rischi più rilevanti legati alla sicurezza delle applicazioni esposte in rete, come descritto nella OWASP Top Ten[20], sono:

- *injection*, è una tecnica che permette di influenzare un'operazione derivante da un input e iniettare comandi non previsti che verranno poi interpretati ed eseguiti permettendo quindi l'attuazione di operazioni o interrogazioni arbitrarie. Questo è reso possibile da uno scarso controllo dei dati ricevuti e che vengono utilizzati all'interno di interazioni con una base di dati o con un sistema. Grazie a questo tipo di attacchi solitamente è possibile interagire con le strutture dati;
- *broken authentication*, la gestione dell'autenticazione e delle sessioni di una connessione possono non essere implementate correttamente esponendo a possibili autenticazioni non autorizzate e compromissione dei dati;
- *esposizione di dati sensibili*, questo tipo di attacco consiste nell'intercettare, in una comunicazione non sicura, dati sensibili, operazione resa possibile da una scarsa protezione di tali dati durante uno scambio tra sistemi;
- *Entità esterne XML* (XXE), consiste nell'inviare messaggi XML[21] con riferimenti esterni a un interprete mal configurato ottenendone impropriamente così l'accesso;
- *configurazione insicura*, l'utilizzo di software non aggiornati o configurazioni obsolete permettono accessi o utilizzi non autorizzati;
- *cross-site scripting* (XSS), consiste nella possibilità di inserire codice arbitrario e dati provenienti da fonti non affidabili che consentono di eseguire codice arbitrario. A differenza dell'injection questo tipo di vulnerabilità permette, oltre a estrapolare dati sensibili, di eseguire script malevoli direttamente nel client dell'utente.

[20] https://owasp.org/www-project-top-ten/

[21] eXtensible Markup Language: un linguaggio di markup che permette di descrivere delle regole in un formato leggibile dall'uomo definito dalla World Wide Web Consortium (W3C) https://www.w3.org/XML/

Le vulnerabilità più diffuse, legate sia a software popolari sia ai diversi sistemi operativi, vengono raccolte e catalogate per tipologia e versione del programma in varie liste pubbliche, come per esempio la *Common Vulnerabilities and Exposures*[22] (CVE).

2.3.1 Nmap

Nmap[23] (Network Mapper) è uno strumento gratuito e open source per la scoperta degli host in una rete e i test di sicurezza. Nasce principalmente per effettuare *port scanning*, ovvero analisi e individuazione delle porte TCP e UDP non filtrate su cui un sistema è in ascolto. Nmap è stato ideato per i sistemisti e gli amministratori di rete per eseguire test di controllo e scansioni di debug.

Questo strumento utilizza alcuni pacchetti di varie tipologie per determinare gli host disponibili in rete, che tipo di servizi offrono, di quale sistema operativo fanno uso e quale tipologia di pacchetto viene filtrata.

Nmap è un software da linea di comando che è stato utilizzato come base di partenza per creare una suite più completa di strumenti di analisi di rete, per esempio una sua versione che utilizza l'interfaccia grafica (Zenmap[24]) oppure una sua versione con finalità di debug (Ncat[25]).

Un'altra importante caratteristica di Nmap è il motore che si interfaccia con le sue funzionalità interne, denominato *Nmap Scripting Engine* (NSE), che è in grado di interpretare degli script scritti in `Lua`[26] ed espanderne le funzionalità.

Gli script integrandosi con il sistema di analisi permettono di eseguire anche alcune tipologie di attacchi informatici direttamente tramite questo strumento. Un esempio presente sul sito di nmap è *ssh-brute*[27], ovvero uno script che permette di effettuare un attacco di forza bruta[28] sulle password del servizio ssh[29].

Un esempio di comando che permette di eseguire l'attacco sopra citato ad un ipotetico bersaglio con indirizzo IP 192.0.2.2 con un servizio ssh configurato sulla porta TCP di default 22 è:

```
nmap -p 22 --script ssh-brute \
  --script-args userdb=users.lst,passdb=pass.lst \
  --script-args ssh-brute.timeout=4s 192.0.2.2
```

dove `users.lst` e `pass.lst` sono due liste contenenti rispettivamente nomi di utenti e password da tentare, mentre `ssh-brute.timeout` indica il timeout da utilizzare nella connessione per rilevare una eventuale disconnessione.

2.3.2 SQLmap

Sqlmap[30] è uno strumento open source di penetration testing che automatizza processi di individuazione di vulnerabilità di tipo SQL injection.

---

[22] https://cve.mitre.org/

[23] https://www.nmap.org

[24] https://nmap.org/zenmap/

[25] https://nmap.org/ncat/

[26] Un linguaggio di programmazione cross-platform di alto livello https://www.lua.org/

[27] https://nmap.org/nsedoc/scripts/ssh-brute.html

[28] Ricerca esaustiva di tutte le combinazioni possibili di caratteri fino a quando non venga trovata una corrispondenza con la password, questo tipo di attacchi solitamente dà precedenza a quelle note.

[29] Secure Shell, è un protocollo di accesso tramite sessione sicura ad un sistema.

[30] http://sqlmap.org/

La *SQL injection* è un caso particolare di tecnica di tipo injection che utilizza istruzioni SQL[31], un tipo di attacco che solitamente permette di scoprire la struttura del database di un'applicazione e recuperare informazioni sensibili.

L'utilizzo di questa vulnerabilità è possibile a causa di uno scarso controllo dei dati ricevuti come input che vengono utilizzati all'interno di query SQL.

A titolo esemplificativo viene riportato un esempio di SQL injection, supponendo che il codice PHP[32] eseguito da una pagina chiamata tramite una richiesta HTTP con metodo GET sia:

```
<?php
    $txtUserId = getRequestString("id");
    $txtSQL = "SELECT * FROM Users WHERE UserId = '" + $txtUserId+"'";
?>
```

dove `getRequestString($campo)` è una funzione che permette di individuare i valori del parametro `$campo` inviato tramite la richiesta GET e che `$txtSQL` sia la query da utilizzare per interrogare il database e scoprire se l'utente richiesto sia esistente o meno.

Ipotizzando che la richiesta alla pagina venga effettuata tramite il seguente URL:

```
http://example.com/app/accountView?id=' or '1'='1
```

La variabile `$txtUserId` assume quindi il seguente valore: `' or '1'='1`, utilizzandola per comporre `$txtSQL` la richiesta al database diventa:

```
SELECT * FROM Users WHERE UserId = '' OR '1'='1'
```

Come è possibile notare in questa query l'espressione: `'1'='1'` è sempre vera e insieme all'`OR` è possibile enumerare tutti gli utenti presenti nella tabella `Users`.

Sqlmap supporta la scansione di diverse tipologie di database e colleziona numerose tecniche per eseguire analisi di vulnerabilità e attacchi di tipo SQL injection. Tra le varie tecniche che utilizza ci sono script per enumerazioni frequenti come: ricerca di utenti, password, ruoli ed esplorazione completa del database vittima. Inoltre può riconoscere la tipologia di hash usata dalle password, il che permette in un secondo momento di trovare una corrispondenza tra l'hash e la password originale utilizzando le rainbow table[33], oppure eseguendo attacchi a dizionario nel caso in cui la funzione di hash utilizzata sia debole.

2.3.3 Wapiti

Wapiti[34] è uno strumento di analisi delle vulnerabilità per le applicazioni web. Automatizza controlli di sicurezza dei siti ed esegue analisi black-box, ovvero test non specifici per la vittima, in quanto non è possibile conoscere il suo codice sorgente. Tramite bot e script che leggono le pagine web, Wapiti cerca di iniettare dati in URL e form per scovare vulnerabilità cross-site scripting (XSS).

Una vulnerabilità XSS è resa possibile da uno scarso controllo e/o una cattiva manipolazione dei dati che un utente può inserire interagendo con il sito.

Un esempio di questa vulnerabilità è una porzione di codice PHP lato server che stampa un valore passato tramite una richiesta HTTP con metodo GET:

```
<?php
    echo $_GET['value'];
?>
```

[31] Structured Query Language è un linguaggio di interrogazione per sistemi di basi di dati.

[32] Un linguaggio di scripting https://www.php.net/

[33] Tabelle contenenti associazioni precalcolate di password note e il rispettivo hash.

[34] http://wapiti.sourceforge.net/

Ipotizzando che l'URL che punta al codice sopra descritto sia:

```
http://example.com/app/page?value=<script>alert("Codice arbitrario")</script>
```

la pagina, quando lo interpreterà, si troverà nella condizione di stampare il contenuto del campo `value` presente nell'URL, ovvero un codice arbitrario `JS`[35] che verrà eseguito nel sistema che ha effettuato la richiesta (il client). La possibilità di eseguire questa tipologia di codice espone il sistema a furti di dati/sessioni o altri rischi.

2.3.4 Burpsuite

Burpsuite[36] è un software che serve per scansionare applicazioni web e per eseguire penetration test. Ha due versioni, una gratuita scaricabile dal sito ufficiale, ma limitata e una pro a pagamento che comprende funzionalità aggiuntive.

Burpsuite può essere configurato come proxy server[37] per interagire con richieste e traffico HTTP. Questo strumento è stato pensato per interagire principalmente con web browser e operare su siti e applicazioni web, ma è possibile configurarlo per lavorare anche con altri strumenti (es. mail e voip).

Burpsuite è dotato di un'interfaccia grafica che ne permette il pieno controllo. Le sue principali funzionalità sono intercettare, mostrare e modificare le richieste HTTP, affinchè sia possibile effettuare test e attacchi. Inoltre può essere esteso integrando funzionalità aggiuntive per gestire anche nuove tipologie di scenari.

2.3.5 OpenVAS

OpenVAS[38] (Open Vulnerability Assessment System) è un framework per l'analisi e la gestione delle vulnerabilità. Lo scanner è stato sviluppato e mantenuto da Greenbone Network ed è rilasciato sotto licenza GNU GPL.

Questo strumento permette di eseguire varie tipologie di test partendo dal basso livello, analizzando i vari protocolli di rete, fino alle applicazioni web e le loro funzionalità. OpenVAS integra il software nmap (vedere Sezione 2.3.1) dal quale è possibile eseguire scansioni delle porte TCP e UDP come operazione preliminare prima di testare la presenza di vulnerabilità.

OpenVAS è in grado di monitore e generare report di sistema con lo scopo di scoprirne lo stato e migliorarlo nel caso siano presenti punti deboli. Inoltre, permette di automatizzare e integrare queste operazioni all'interno di altri strumenti di vulnerability assessment. I diversi test di vulnerabilità automatizzati vengono detti *Network Vulnerability Test* (NVT).

OpenVAS ha un proprio database che viene costantemente aggiornato con i nuovi CVE scoperti, script in grado di individuarli e una classificazione di pericolosità. In questo modo i report prodotti sono più aggiornati possibile e permettono, grazie alla valutazione delle vulnerabilità, di individuare le criticità più importanti.

OpenVAS è un software di tipo client-server costituito da quattro componenti connessi tra di loro come nello schema logico in Figura 2.12. I componenti sono i seguenti:

- *Greenbone Vulnerability Manager* (GVMd) è il servizio al centro dell'architettura di OpenVAS che si occupa di orchestrare le interazioni fra i vari componenti e con i database;

[35] JavaScript un linguaggio di programmazione orientato ad oggetti. https://developer.mozilla.org/it/docs/Web/JavaScript

[36] https://portswigger.net/burp

[37] Un servizio intermediario che si interpone tra il client e il server. Esso prende in carico le richieste del primo e le inoltra al secondo e seguentemente inoltra anche le risposte del server.

[38] http://www.openvas.org/

- *Greenbone Security Assistant* (GSA) è l'interfaccia web di GVMd che permette all'utente di interagire con il manager;
- *OpenVAS Scanner*, è il componente recupera le NVTs e le utilizza per effettuare le scansioni vere e proprie;
- *GVM-Tools* è l'interfaccia web complementare che permette di creare script e programmi che possano interagire e integrarsi con OpenVAS.

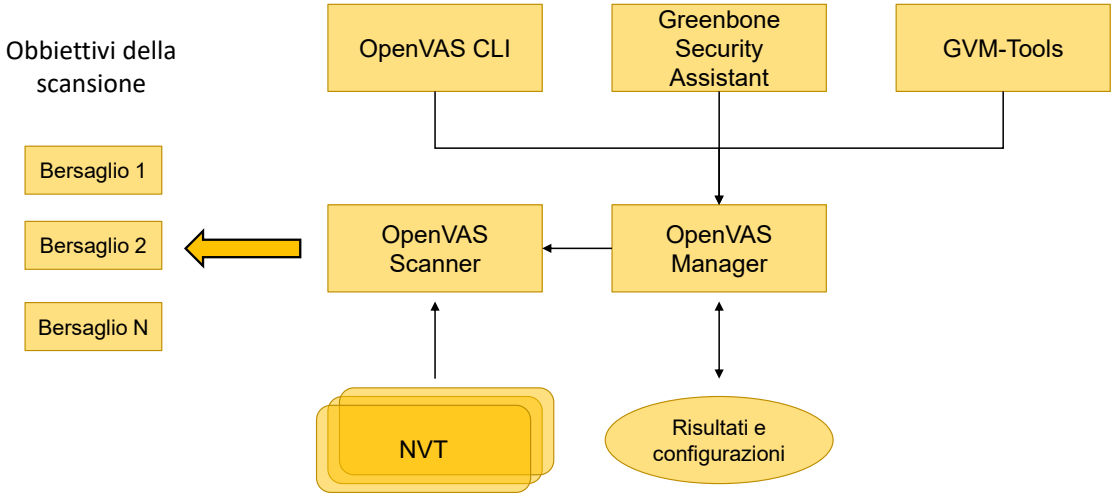


Figura 2.12: Schema logico di OpenVAS.

2.3.6 Nessus

Nessus[39] è un software proprietario di tipo client-server per eseguire scansioni di vulnerabilità, sviluppato e mantenuto da Tenable. Inizialmente era stato rilasciato in una versione open source, per poi essere successivamente trasformato in un software closed source e a pagamento; nonostante ciò ne rimane comunque una versione limitata e gratuita per uso domestico chiamata Nessus Essentials.

Una delle peculiarità di Nessus è il suo enorme database di vulnerabilità e la velocità con cui viene aggiornato grazie al team di ricerca di Tenable che si adopera per creare le nuove regole da introdurre negli automatismi del software.

Altre due importanti caratteristiche di Nessus sono la sua grande configurabilità e la possibilità di essere espanso tramite dei plugin. Come si legge sul suo sito web, il software ha un'ampia quantità di estensioni create dall'utenza, le quali permettono di implementare test e report anche molto specifici per particolari virus o vulnerabilità.

Nessun inoltre offre un sistema di reportistica molto avanzato che può essere personalizzato e consegnato direttamente a un cliente.

2.3.7 Qualys VMDR

Qualys VMDR[40] è una piattaforma all-in-one acronimo di Vulnerability Management, Detection and Response e, come suggerisce il nome, si occupa di molti degli aspetti legati al mondo della

[39] https://www.tenable.com/products/nessus

[40] https://www.qualys.com

sicurezza informatica. Questo servizio è fornito dall'azienda Qualys tramite un pagamento di un canone e segue il modello di *software as a service* (SaaS), ovvero un programma ospitato in una infrastruttura cloud che mette a disposizione le sue funzionalità tramite Internet. Il modello SaaS permette di evitare l'installazione di un software sulle proprie macchine interagendovi invece attraverso opportune interfacce pubbliche.

Tra i vari servizi offerti ci sono sensori che permettono di scansionare la rete e i sistemi di un cliente e generare allarmi e report in caso di comportamenti sospetti. La piattaforma inoltre ha la possibilità di definire operazioni conseguenti a determinati tipologie di allarmi e prendere provvedimenti.

VMDR integra funzionalità di machine learning per determinare, priorizzare e correggere i problemi scoperti in tempo reale. Questo permette non solo di avere una visione completa del sistema, ma anche di avere a disposizione una prima valutazione e la possibilità, se configurato, di intervenire automaticamente.

2.3.8 Considerazioni relative allo studio

Tra gli scanner di vulnerabilità illustrati in questo Capitolo quelli selezionati per lo studio sono: OpenVAS (vedere Sezione 2.3.5), SQLmap (vedere Sezione 2.3.2) e Wapiti (vedere Sezione 2.3.3).

La scelta è ricaduta sui suddetti strumenti in quanto, tra tutti quelli esposti, sono i maggiormente diffusi e open source, si trovano infatti preinstallati all'interno della famosa distribuzione Linux dedicata al pentesting Kali[41][42].

Diversamente, software come Nessus (vedere 2.3.6) e QualysVMDR (vedere 2.3.7) sono soluzioni enterprise che necessitano di relazioni commerciali tra aziende e più difficilmente possono essere utilizzati per un utilizzo malevolo.

Burpsuite (2.3.4) è anch'esso un software gratuito e open source, ma la sua funzione principale è quella di essere utilizzato come proxy applicativo, allontanandosi quindi dallo scopo dello studio.

[41] Sito del progetto: https://www.kali.org/

[42] Elenco dei tool presenti di default: https://tools.kali.org/tools-listing

Capitolo 3

Concetti base di machine learning

Il *Machine Learning* (ML) è una sottocategoria dell'*Intelligenza Artificiale* (AI) che include tecniche statistiche capaci di migliorare le operazioni tramite l'esperienza. Murphy definisce machine learning in [25] come un insieme di metodi che possono automaticamente trovare, attraverso le osservazioni, dei modelli che consentono di predire dati futuri o prendere decisioni in caso di incertezza.

Quando si parla di dati, nell'ambito di problemi statistici, si intende un insieme di osservazioni relative a un determinato spazio descrittivo del fenomeno preso a campione, il che rende i dati il risultato di uno specifico evento.

Le osservazioni sono contraddistinte da *attributi*, o *caratteristiche*, detti in inglese *feature*, che le descrivono. L'insieme dei dati, detto dataset, rappresenta la conoscenza di un fenomeno ed è la base di partenza per trovare dei modelli tramite un approccio di machine learning. In astrazione ogni feature può essere considerata una dimensione del dato, tutte insieme determinano la dimensionalità del dataset.

Il machine learning si divide principalmente in tre tipologie:

- *apprendimento supervisionato*, tra le feature presenti nel dataset ve n'è una detta *target* o obbiettivo dell'apprendimento, gli altri attributi del dataset vengono utilizzati per costruire un modello che possa predire o stimare il target;
- *apprendimento non supervisionato*, consiste nella ricerca di un modello basandosi esclusivamente sullo studio delle relazioni tra i dati e senza conoscerne il target;
- *apprendimento tramite rinforzo*, fa uso di modelli che cercano di raggiungere un determinato obbiettivo in autonomia e, in base alle decisioni, l'azione viene premiata se migliorativa verso l'obbiettivo stesso o punita se negativa.

I metodi di apprendimento supervisionati a loro volta si suddividono in base al tipo di target:

- *classificazione*, l'attributo target è un'etichetta o label e l'obbiettivo della predizione consiste nel trovarla per i dati che ne sono privi. Il target è una variabile discreta;
- *regressione*, l'obbiettivo della predizione è una variabile continua, detta regressione.

Un dataset molto popolare utilizzato come esempio è l'*Iris dataset*[1] che contiene le informazioni legate ad alcuni esemplari di iris. Il dataset è composto da cinque attributi tra cui uno che rappresenta la specie e uno che riporta le misurazioni della lunghezza del petalo in centimetri.

[1] https://archive.ics.uci.edu/ml/datasets/iris

Un esempio di classificazione con questo dataset è riuscire a individuare la specie del dato preso in esame utilizzando gli altri attributi. Invece un esempio di regressione è prevedere la lunghezza del petalo utilizzando le caratteristiche dei dati.

Lo scopo di questa tesi è classificare, tramite dei metodi di apprendimento supervisionati, del traffico di rete distinguendolo tra buono e malevolo.

3.1 Analisi del dataset e classificazione

Un'analisi di machine learning, in questo caso di una classificazione, è solitamente composta dalle seguenti fasi: preprocessamento, valutazione e selezione delle feature, ottimizzazione degli iperparametri e valutazione del modello. Tra le varie tecniche di classificazione in questo studio verranno prese in esame la *foresta casuale* e la *rete neurale*.

Per eseguire una classificazione il dataset viene separato tra dati in *input* (X) e dati in *output* (y), questi ultimi sono composti dalla feature target dell'analisi. L'obbiettivo della classificazione è imparare a mappare la relazione che collega i dati in ingresso X con l'output y. Ogni valore diverso di y rappresenta una classe, in base al loro numero la classificazione si divide in *binaria*, se le classi da predire sono solamente due, e *multiclasse* nel caso siano più di due.

Per valutare i modelli appresi, anche detti *classificatori*, è necessario individuare un sistema che possa determinarne l'efficacia. Un metodo utilizzato consiste nel dividere il dataset in due parti: una detta *training set*, che viene utilizzata per eseguire la fase di apprendimento del modello, e l'altra detta *test set*, ovvero la parte su cui viene valutato quest'ultimo.

La valutazione del modello viene effettuata inserendo i dati di ingresso del test set nel classificatore e tenendo da parte i dati in uscita con le etichette. Il modello predice le classi dai dati di input che andranno poi confrontate con quelle messe da parte all'inizio del test. La percentuale di successi sul totale dei risultati indica l'*accuratezza* del classificatore.

Per evitare che l'accuratezza del classificatore risulti falsata è però necessario che il training set e il test set siano completamente disgiunti. Questo perchè i dati utilizzati nella fase di training vengono appresi attraverso degli algoritmi che comparano il risultato della predizione con il valore vero per effettuare correzioni e se utilizzati anche nella fase di test la predizione del risultato ha una minore probabilità di essere errata rispetto a un dato non utilizzato.

3.1.1 Preprocressamento

I dataset utilizzati sono solitamente frutto di osservazioni della realtà, raccolti con mezzi e modalità a volte anche molto differenti. Per questo motivo i dati non sono sempre direttamente utilizzabili per l'analisi e necessitano di una lavorazione preliminare detta *preprocessamento*.

Nel preprocessamento i dati vengono analizzati e resi omogenei generalmente secondo i seguenti passaggi:

1. pulizia del dataset;
2. uniformazione;
3. standardizzazione e riscalamento.

Pulizia del dataset

Un dataset, per via della sua derivazione anche da diverse sorgenti, potrebbe non essere completo, dunque mostrare campi parzialmente mancanti e valori nulli, e avere osservazioni duplicate. Tutte queste eventualità influiscono negativamente sull'analisi, in quanto alcuni algoritmi potrebbero non essere flessibili alle mancanze oppure condizionare l'apprendimento. Di conseguenza tali problematiche vanno corrette o eliminate.

Inoltre i dataset possono anche contenere degli attributi che non contribuiscono ai fini dell'analisi perché considerati *non-predittivi* oppure eccessivamente correlati con l'output tanto da pregiudicarne una buona generalizzazione.

Nel caso ci siano valori nulli o attributi mancanti in alcuni dati la strategia più semplice da adottare è quella di eliminare le osservazioni che non sono complete. Tale operazione però è possibile solo nel caso in cui il dataset sia abbastanza popolato e non rischi di subire conseguenze.

In alternativa esistono tecniche diverse tra cui il riempimento dei campi attraverso valori medi per non turbare la distribuzione del dataset e l'eliminazione degli attributi dal dataset, come succede per i campi non-predittivi, riducendone così la dimensionalità.

I dati duplicati, eccetto in condizione di studio più specifiche, possono essere rimossi in quanto non apportano nuove informazioni al modello e allungano i tempi di apprendimento.

Uniformazione

La possibile diversità delle fonti dei dati li rende di scale o formati differenti e per questo è necessario uniformarli. Quindi si procede a convertirli per utilizzare la medesima unità di misura al fine di descrivere la stessa caratteristica e individuare la tipologia di variabile informatica da adottare (es. int, float).

Alcuni algoritmi di machine learning lavorano con delle variabili numeriche, per questo è necessario applicare una trasformazione sugli altri tipi di valori e renderli numeri. Lo si fa con un'operazione detta *encoding* che comprende anche il target. Le reti neurali, spiegate nella Sezione 3.2.2, sono un esempio di modello che utilizza algoritmi che necessitano di eseguire un encoding dei dati.

Un esempio di algoritmo di encoding è *one-hot encoding*[2] che consiste nel trasformare gli attributi categorici[3] in numerici. Come si può vedere nella Figura 3.1, l'attributo "colore" contenente i valori rosso, blu e verde viene trasformato in tre attributi, uno per ogni valore distinto di colore, dove 0 e 1 rappresentano la non appartenenza o meno a quel valore.

Dati originali

Quantità	Colore
3	Rosso
1	Blu
6	Blu
8	Verde

One-Hot Encoding

Quantità	Rosso	Blu	Verde
3	1	0	0
1	0	1	0
6	0	1	0
8	0	0	1

Figura 3.1: Esempio di one-hot encoding.

Alla fine dell'analisi verrà eseguita sul target la trasformazione inversa, detta *decoding*, per associare correttamente i numeri individuati e l'etichetta corrispondente.

Da questo punto in poi il dataset sarà omogeneo e lo si potrà trattare come se fosse proveniente da un'unica fonte e compatibile con le specifiche degli algoritmi.

Standardizzazione e riscalamento

Come accennato in precedenza il dataset è composto da diversi dati frutto di osservazioni reali, il che lo rende passibile di errori strumentali o osservazioni fuori scala. Per questo motivo al suo

[2] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[3] Gli attributi categorici possono assumere un numero finito di valori distinti non numerici.

interno si possono trovare anche valori anomali detti *outlier*. In alcuni casi tali anomalie possono consistere in numeri assoluti troppo grandi, tanto da influire sui dati più rappresentativi.

Una pratica importante per alcuni algoritmi, come spiegato anche su Scikit-Learn[4], è quella di standardizzare il dataset, ovvero eseguire una trasformazione sui dati per portarli ad avere la media uguale zero e la varianza uguale a uno. L'operazione di *standardizzazione* rientra tra quelle di preprocessamento, in particolare permette di evitare di attribuire maggiore importanza a valori assoluti troppi grandi, per esempio quelli degli outlier, senza però annullarli.

Essendo x una osservazione del dataset di input X, la standardizzazione viene effettuata secondo questa formula:

$$x_{stand} = \frac{x - \mu}{\sigma}$$

dove μ rappresenta la media del dataset, σ la sua varianza e x_{stand} la variabile standardizzata.

Una tecnica alternativa alla standardizzazione è lo *scalamento delle feature*, che consiste nel riportare tutti i valori in range fissati, solitamente [0,1] oppure [−1,1]. A differenza della standardizzazione, il dataset scalato avrà delle deviazioni standard più piccole e l'effetto degli outlier verrà soppresso.

Essendo x una osservazione di un attributo del dataset di input X, lo scalamento segue la seguente relazione:

$$x_{scal} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

dove x_{min} e x_{max} sono i valori minimi e massimi di quella feature nel dataset e x_{scal} è il valore scalato.

Sbilanciamento

In una classificazione può succedere che il numero dei dati per ciascuna classe sia diverso e quando alcune classi sono fortemente in minoranza rispetto ad altre si dice che il dataset è *sbilanciato*. Questo comporta che il modello, durante l'apprendimento, venga influenzato più spesso dai dati di una determinata classe piuttosto che da quella di minoranza, rischiando quindi di non riuscire a generalizzarla.

Per combattere questo tipo di problema si possono adottare principalmente tre soluzioni: il ricampionamento, l'utilizzo dei pesi e il downsampling.

Nel primo caso, a meno che non sia possibile aggiungere osservazioni al dataset, si utilizzano tecniche che simulano un *ricampionamento*. Ciò si verifica creando nuovi dati della classe in minoranza sulla base delle osservazioni a disposizione cercando di non modificare la distribuzione del dataset.

Nel secondo caso invece per ogni classe viene calcolato un *peso*, ovvero quanta importanza dare al singolo dato di una determinata classe. In questo modo il modello, quando incontrerà i vari dati, apprenderà in maniera proporzionale al peso.

Nell'ultimo caso, il *downsampling*, vengono scartate alcune osservazioni della classe presente in maggiore quantità per allineare le dimensioni delle classi. Questo tipo di soluzione può essere utilizzata solamente nel caso in cui il dataset sia sufficientemente grande. Infatti la rimozione di campioni introduce il rischio di eliminare anche informazioni potenzialmente utili all'analisi se non si hanno abbastanza osservazioni.

3.1.2 Valutazione e selezione delle feature

La classificazione consiste nell'individuare una determinata classe corrispondente a un certo input basandosi sui dati che si possiedono. Nello specifico ciò è reso possibile grazie a un modello

[4] https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

matematico che separa le classi basandosi sul confronto delle distanze tra i dati. Concettualmente quindi un modello è un iperpiano che divide lo spazio dimensionale del dataset, ricordando che la dimensionalità dei dati è rappresentata dal numero dei suoi attributi. Quando essa cresce i dati nello spazio diventano radi, il che comporta diversi problemi dal punto di vista statistico rendendo inefficaci e meno affidabili i modelli costruiti.

Questo problema è noto come *maledizione della dimensionalità* e costituisce una difficoltà dal punto di vista computazionale e di accuratezza. Per questo è necessario trovare dei metodi che permettano di rappresentare i dati in uno spazio con minori dimensioni, senza però al contempo perdere troppe informazioni.

Questa metodologia prende il nome di *selezione delle feature* e ha tre principali benefici:

- riduzione di overfitting, spiegato nella Sezione 3.1.3, una minore ridondanza dei dati permette di ridurre le decisioni basate sul rumore;
- aumento dell'accuratezza, vengono mantenuti nel dataset solo i dati più significativi;
- riduzione del tempo di addestramento, ovvero il tempo impiegato nell'eseguire la costruzione del modello.

La selezione delle feature può avvenire in diversi modi, in particolare ci sono tre tipologie di approcci alla riduzione delle caratteristiche:

- *sottoselezione delle feature*, ovvero algoritmi che identificano un sottoinsieme delle caratteristiche che risultano più importanti ai fini della predizione;
- *shrinkage* o restringimento, azzerando i coefficienti di qualche caratteristica è possibile ridurne la varianza permettendo di regolarizzare il modello predittivo. Un esempio di questa tecnica è il *Lasso*[5];
- *proiezione di dimensionalità*, il dataset viene proiettato in un nuovo spazio dimensionale più piccolo. A differenza delle altre tipologie, le caratteristiche non vengono direttamente selezionate, ma utilizzate per crearne di nuove mantenendo la maggior parte delle informazioni del problema in uno spazio più piccolo. Per questo motivo tale tecnica rientra tra quelle dette *feature extraction*, ne è un esempio il metodo di *analisi delle componenti principali* o principal component analysis (PCA)[6].

La sottoselezione delle feature consiste nell'utilizzare un sistema di punteggi per individuare le caratteristiche più rappresentative del dataset o utili ai fini della predizione. Una di queste tecniche è *l'analisi univariata*[7] che consiste nel valutare e attribuire un punteggio separatamente per ogni feature, rispetto al target, tramite test statistici per poi selezionare le migliori.

Questo tipo di selezione è strettamente dipendente dal test con il quale i punteggi vengono assegnati. In questo studio in particolare si fa uso di *chi quadrato* (χ^2) e *informazione mutua*, entrambi i test stimano il grado di dipendenza tra le variabili, ma il primo è maggiormente influenzato dalla quantità di dati disponibili e accetta solamente valori non negativi.

Il *test chi quadrato* permette di attribuire un punteggio alle varie differenze tra le frequenze ottenute e quelle teoriche. Come descritto in [26] la formula per ricavare il suo valore è:

$$\chi^2 = \sum_{i=1}^{K} \sum_{j=1}^{C} \frac{(n_{ij} - \hat{n_{ij}})^2}{\hat{n_{ij}}}$$

[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[7] https://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

dove K è il numero degli attributi, C è il numero delle classi da predire, n_{ij} è la frequenza congiunta ottenuta e $\hat{n_{ij}}$ è quella teorica.

Un differente approccio probabilistico è quello descritto in [25], esso parte dalla teoria delle probabilità e dalla caratteristica che possiedono due variabili indipendenti. Nello specifico due variabili casuali indipendenti X e Y avranno la probabilità congiunta uguale al prodotto delle probabilità marginali delle singole componenti x e y. Questa caratteristica può essere così descritta:

$$p(x, y) = p(x) \cdot p(y)$$

dove $p(x, y)$ è la probabilità congiunta, $p(x)$ e $p(y)$ sono invece le singole probabilità marginali.

Questo significa che x e y sono due eventi non correlati e al variare della probabilità di uno non si possono fare ulteriori considerazioni sull'altro.

L'*informazione mutua* cerca di individuare il grado di relazione tra due variabili in funzione di quanto si avvicina alla distribuzione congiunta di ogni sua componente, l'equazione che la descrive è:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right)$$

L'analisi univariata fa uso di test statistici, per esempio quelli appena descritti, per ottenere tutti i punteggi che indicano il grado di relazione tra un attributo e il target. Successivamente vengono prelevati alcuni di essi, in base a delle politiche di selezione, per formare un dataset con la stessa quantità di osservazioni, ma con dimensionalità ridotta, uguale al numero di attributi prelevati. Nelle politiche di selezione è possibile indicare un numero massimo di feature che possono essere utilizzate e la tipologia di comparazione da effettuare tra vari punteggi. Un esempio di queste politiche è la selezione tramite ordinamento decrescente oppure tramite soglie di punteggi.

3.1.3 Parametri dei modelli

I vari algoritmi di machine learning possiedono alcuni parametri che possono essere assimilati tramite il rispettivo processo di apprendimento; allo stesso tempo ve ne sono degli altri che vengono regolati solo dall'operatore detti *iperparamentri*. Entrambi i tipi di parametri influiscono sulle capacità di apprendimento del modello e, se non scelti o valutati con cura, possono causare fenomeni di overfitting o underfitting.

Il fenomeno dell'*overfitting* si ha quando un modello statistico si adatta ai dati osservati perdendo la generalizzazione del modello e dunque causando l'effetto opposto a quello che si vorrebbe ottenere durante questo tipo di apprendimento. Due dei fattori che possono portare all'overfitting sono l'eccessiva flessibilità del modello e una quantità di parametri elevata rispetto al numero di osservazioni coinvolte (vedere Figura 3.2).

Il fenomeno dell'*underfitting* è invece l'opposto del precedente, in quanto il modello manifesta scarse prestazioni sui dati di addestramento. Ciò avviene perché il modello non è in grado di individuare le relazioni tra i dati di input e i valori target, problemi potenzialmente causati da un modello troppo semplice o poco flessibile.

Con l'overfitting l'accuratezza del modello risulta essere molto alta sul training set, mentre con l'underfitting bassa; di conseguenza quando i modelli vengono valutati tramite il test set o utilizzati in situazioni reali le performance rischiano di rivelarsi pessime.

Ottimizzazione degli iperparametri

Per decidere i valori ottimali degli iperparametri non ci si può basare sulle performance ottenute sul test set, in quanto si rischia di incorrere in un risultato non valido. Nel caso la base di dati

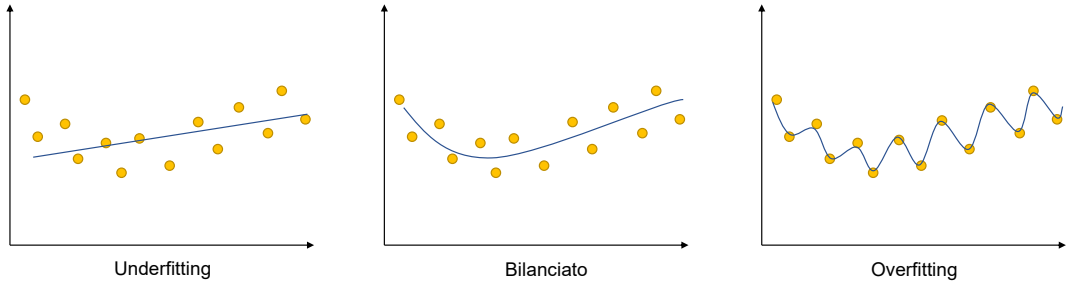


Figura 3.2: Rappresentazione di underfitting, curva bilanciata e overfitting.

utilizzata sia abbastanza grossa, la si può dividere in 3 parti: training set, validation set e test set.

Sul primo viene eseguito l'apprendimento con un certo parametro per poi testarlo sul validation set. Ripetendo questa operazione diverse volte con valori dei parametri differenti si cerca di individuare quello migliore e poi si valuta la sua efficacia sul test set, vedere Figura 3.3.

Quando il dataset non è molto grosso e la sua divisione potrebbe creare problemi si può utilizzare la *k-fold cross validation*, ovvero un sistema secondo il quale il dataset viene diviso in k parti e, a turno, ciascuna viene utilizzata come test set e le altre come training e alla fine si calcola la media per mitigare l'effetto della selezione, vedere Figura 3.3. In questa tesi non se ne fa uso in quanto il dataset è sufficientemente grande, ma è stata comunque presentata in quanto è una tecnica molto diffusa e utile a sottolineare l'importanza sulla quantità dei dati necessari per l'analisi.

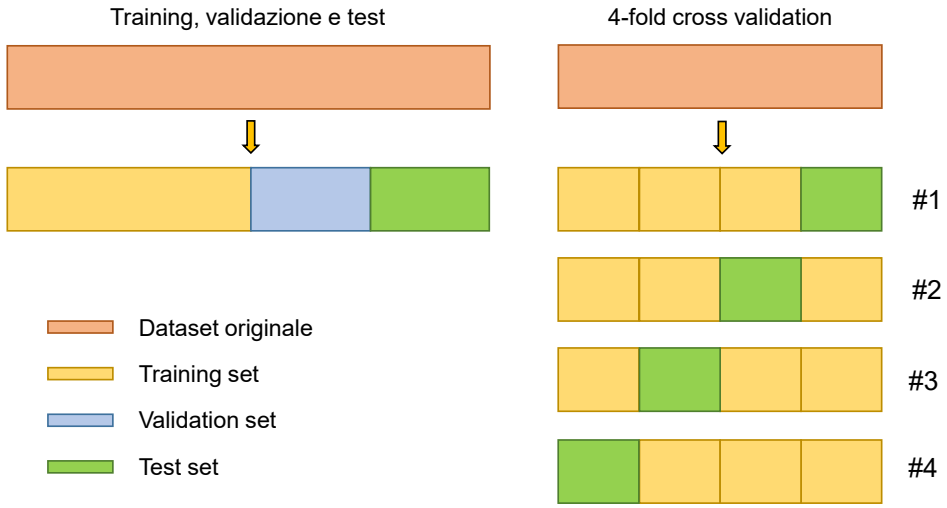


Figura 3.3: Training, validation e test a confronto con 4-fold cross validation.

Entrambi questi approcci permettono di ridurre i fenomeni sopra descritti in quanto consentono di effettuare valutazioni su più porzioni di dataset e senza incorrere in un bias di selezione, ovvero la possibilità di influenzare l'accuratezza adottando specifiche selezioni.

In un'analisi con più iperparamentri il problema di ottimizzazione diventa maggiormente complesso, l'obbiettivo è individuare quelli che permettano di ottenere il modello che rispecchia meglio le caratteristiche desiderate.

Un metodo utilizzato per ottenere questo risultato è *grid search*, ovvero una ricerca esaustiva di tutte le possibili combinazioni dei parametri fino a trovare quella ideale. Questo procedimento

consiste nel reiterare l'apprendimento del modello da analizzare dall'inizio, utilizzando ogni set di iperparamentri sul training set. Ogni modello appreso viene valutato tramite delle metriche di performance, scelte dall'operatore, sul validation set. Nel caso il dataset non sia sufficientemente grande si può effettuare grid search tramite tecniche come k-fold cross validation ed eseguire i test di performance su ogni parte. Al termine dell'esplorazione viene selezionato il modello con le performance più elevate.

3.1.4 Valutazione del modello

Una volta che l'apprendimento è terminato bisogna valutare le performance del modello e per farlo esistono diversi fattori che possono essere presi in considerazione. Come descritto in [25] e in [27], questi valori che fungono da metro di valutazione non sono sempre tutti efficaci per le varie tipologie di studio. Per far fronte a questa problematica esistono alcuni tipi di indice, ognuno dei quali è composto da diverse combinazioni di fattori.

Così ogni indice pone l'accento su caratteristiche differenti del modello e, siccome ogni analisi ha le proprie condizioni di partenza e i propri obbiettivi, bisogna individuare quali sono gli indici più adatti da osservare per descriverlo.

Come accennato in precedenza, le valutazioni del modello vengono svolte sul test set, l'input di test X_{test} viene utilizzato per eseguire le predizioni $\hat{y}$ e poi confrontate con l'output di test y_{test}.

L'accuratezza è il primo fattore, nonchè il più semplice, per valutare un classificatore e rappresenta la percentuale di predizione corrette sul totale delle predizioni sul test set.

Quando la distribuzione delle classi è bilanciata l'accuratezza si dimostra un buon parametro di valutazione del modello. Nel caso invece il dataset fosse sbilanciato l'accuratezza risulterebbe poco significativa a causa della forte minoranza di una o più classi rispetto ad altre più popolate. In questo caso il classificatore apprenderebbe maggiormente dalle classi più numerose e non riuscirebbe a generalizzare le classi minoritarie, a meno di intervenire sullo sbilanciamento (vedere Sezione 3.1.1). Nello scenario appena descritto l'accuratezza, che si basa sulla totalità delle predizioni, risulterebbe a favore della classe maggioritaria sovrastando la scarsa precisione sulla predizione delle classi più piccole.

Per tentare di aggirare questo problema si può ricorrere all'utilizzo dell'*accuratezza bilanciata*. Essa consiste nel calcolo dell'accuratezza per la singola classe, per poi dedurne una media ponderata utilizzando pesi differenti per le varie classi.

Per definire altre metodologie di valutazione verrà preso ora in considerazione l'esempio di una classificazione binaria, nella quale le classi da predire sono negativa (*0*) e positiva (*1*). Data questa premessa sarà possibile ricercare indici in grado di descrivere la capacità del modello di predire correttamente le classi e stimare quanto invece sbagli. Grazie alla possibilità di confrontare i valori predetti e i valori reali si potrà quantificare sia gli errori sia le predizioni corrette.

In base all'output atteso si possono avere due tipi di stime esatte: *veri positivi* o true positive (TP) se il valore atteso era *1*, *veri negativi* o true negative (TN) se *0*. Mentre gli errori nei quali si può incorrere in una classificazione binaria sono due: falsi positivi e falsi negativi. I *falsi positivi* o false positive (FP) sono previsioni di quando viene stimato *1* mentre la predizione corretta è in realtà *0*; i *falsi negativi* o false negative (FN) sono l'esatto contrario.

Per avere una visione completa di questi valori è possibile disporli come mostrato nella Tabella 3.1 formando così la *matrice di confusione*.

		Valori reali	
		1	*0*
Valori predetti	*1*	Veri positivi (TP)	Falsi positivi (FP)
	0	Falsi negativi (FN)	Veri negativi (TN)

Tabella 3.1: Matrice di confusione.

Volendo descrivere l'accuratezza utilizzando questi nuovi indici potremmo utilizzare la seguente formula:

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

Da questi valori è possibile definire ulteriori indicatori che permettono osservare il modello da altri punti di vista.

La *frequenza di veri positivi* o true positive rate (TPR), anche detta *sensibilità*, rappresenta la capacità del modello di individuare la classe positiva *1* rispetto alla quantità totale dei dati che sono positivi. In termini probabilistici si potrebbe dire che TPR rappresenti la probabilità di stimare *1* quando il valore atteso è *1*.

La sensibilità può essere descritta dalla seguente relazione:

$$TPR = \frac{TP}{TP + FN} \approx p(\hat{y} = 1 | y = 1)$$

Allo stesso modo è possibile definire la *frequenza dei falsi positivi* o *falsi allarmi* (FPR), in inglese false positive rate, che rappresenta la probabilità del modello di stimare *1* quando il valore vero è *0*.

Questa frequenza è definita dalla seguente equazione:

$$FPR = \frac{FP}{FP + TN} \approx p(\hat{y} = 1 | y = 0)$$

Combinando questi valori e posizionandoli in un grafico, utilizzando TPR come ascissa e FPR come ordinata, si ottiene uno schema noto come *curva ROC* (Receiver Operating Characteristic). Essa descrive come cambiano le performance in base alla variazione della discriminante interna al classificatore che permette di selezionare la giusta etichetta.

La qualità di questa curva può essere descritta utilizzando la grandezza dell'area sottesa dalla curva stessa, detta AUC (Area Under the Curve), la quale rappresenta il grado di separazione del modello e indica quanto è in grado di distinguere tra le classi.

Per paragonare due modelli di classificazione è possibile misurare le due AUC, quella più grande avrà una classificazione migliore, se ne riporta una di esempio nella Figura 3.4.

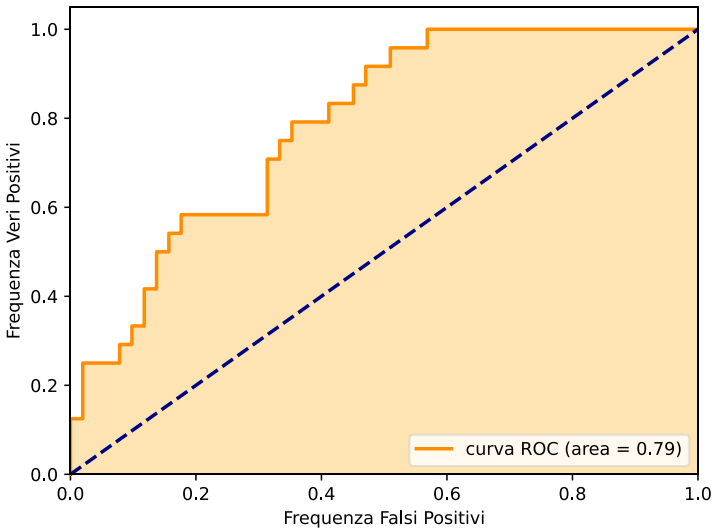


Figura 3.4: Esempio di curva ROC.

Un altro indice che si può costruire è la *precisione* (P) che rappresenta la capacità del modello di individuare la classe *1* rapportata alla quantità di volte che ha sbagliato:

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

Fissati i parametri del modello, si può calcolare un singolo valore di precisione (P) e di sensibilità (R). Questi due indici si possono combinare insieme per ottenerne un altro denominato *valore* F_1 descritto dalla seguente relazione:

$$F_1 = 2 \cdot \frac{P \cdot R}{R + P}$$

Questo valore rappresenta l'insieme degli indici precedentemente descritti e ne misura la bontà ponendo enfasi anche sul parametro della sensibilità che per classificazioni sbilanciate può rivelarsi un parametro importante.

Gli indici appena presentati, come è possibile osservare della loro formule, sono costruiti considerando esclusivamente due classi, quindi validi per i classificatori binari. Tuttavia, come descritto nel già citato [27], possono essere estesi per essere adattati per valutare anche i classificatori multiclasse, nello specifico viene fatta una media degli indici considerando tutte le classi.

L'indicatore dell'accuratezza viene costruito sommando l'accuratezza ottenuta per ogni classe e divisa per il numero delle classi, diventando quindi un'accuratezza media, rappresentata dalla seguente formula:

$$accuratezza\_media = \frac{\sum_{i=1}^{C} \frac{TP_i + TN_i}{TP_i + FN_i + FP_i + TN_i}}{C}$$

dove C rappresenta il numero delle classi, TP_i i veri positivi per la classe i-esima, TN_i i veri negativi per la classe i-esima, FP_i i falsi positivi per la classe i-esima e FN_i i falsi negativi per la classe i-esima.

La precisione, come nel caso dell'indice precedente, viene calcolata sommando tutte le precisioni di ogni classe e divisa per il numero di esse. Viene rappresenta dalla seguente relazione:

$$P_C = \frac{\sum_{i=1}^{C} \frac{TP_i}{TP_i + FP_i}}{C}$$

dove la simbologia è analoga a quella dell'indice precedente. Il seguente indice rappresenta quindi la capacità media del classificatore di non scambiare le classi con le altre.

Invece la sensibilità, anche detta richiamo, può essere espressa dalla seguente relazione:

$$R_C = \frac{\sum_{i=1}^{C} \frac{TP_i}{TP_i + FN_i}}{C}$$

dove la simbologia rimane la medesima e l'indice rappresenta la capacità media del classificatore di individuare la classe corretta rispetto alle altre.

Come per il classificatore binario, l'indice F1 è in grado di esprimere un giudizio medio tra la precisione e il richiamo, in particolare per i classificatori multiclasse la formula che lo descrive è:

$$F1_C = 2 \cdot \frac{P_C \cdot R_C}{R_C + P_C}$$

3.2 Modelli di classificazione

Esistono diversi modelli matematici che possono essere utilizzati per eseguire una classificazione. Per poterlo classificare, l'output del modello deve essere un valore discreto oppure opportunamente trasformato tramite un valore di soglia.

In questo studio vengono presi in esame la foresta casuale e la rete neurale. Entrambi sono modelli con diversi parametri interni che si prestano all'apprendimento anche di dataset sbilanciati. Il primo basato sugli alberi e il secondo basato su percettroni.

3.2.1 Foreste casuali

Come spiegato in [28], le tecniche di classificazione basate sugli alberi sono semplici e nei casi più basilari sono anche facilmente interpretabili. Questo fa sì che si possano individuare e osservare le scelte intraprese dall'algoritmo. L'unità base di questo tipo di modelli è l'albero decisionale.

Albero decisionale

Un *albero decisionale* è il risultato di un algoritmo che, tramite alcuni parametri, costruisce un grafo di decisioni e delle loro possibili conseguenze. I nodi dell'albero rappresentano le decisioni che permettono di ottenere la classificazione e le foglie rappresentano le classi.

Ogni nodo tramite i parametri dell'algoritmo determina quali caratteristiche sono da utilizzare e per quali condizioni è possibile creare un bivio decisionale che potrebbe portare a due classi differenti (vedere Figura 3.5).

La costruzione del grafo parte dall'analisi del dataset, si cerca di individuare la miglior caratteristica che sia in grado di separare al meglio i dati. Questa operazione viene eseguita ripetutamente per ogni nodo, escludendo di volta in volta le caratteristiche già utilizzate per un determinato percorso, ma riutilizzabili per un altro.

La discesa finisce quando sono state utilizzate tutte le caratteristiche per un percorso e si passa a completare gli altri, oppure quando si arriva ad un certo limite imposto dall'operatore. La scelta della caratteristica avviene tramite la valutazione di un indice.

Ogni foglia rappresenta una classe e per decidere la corrispondenza foglia-classe si valutano le frequenze delle etichette del training set che raggiungono una determinata foglia. L'albero decisionale poi associa alle osservazioni la classe appartenente alla foglia su cui ricadono.

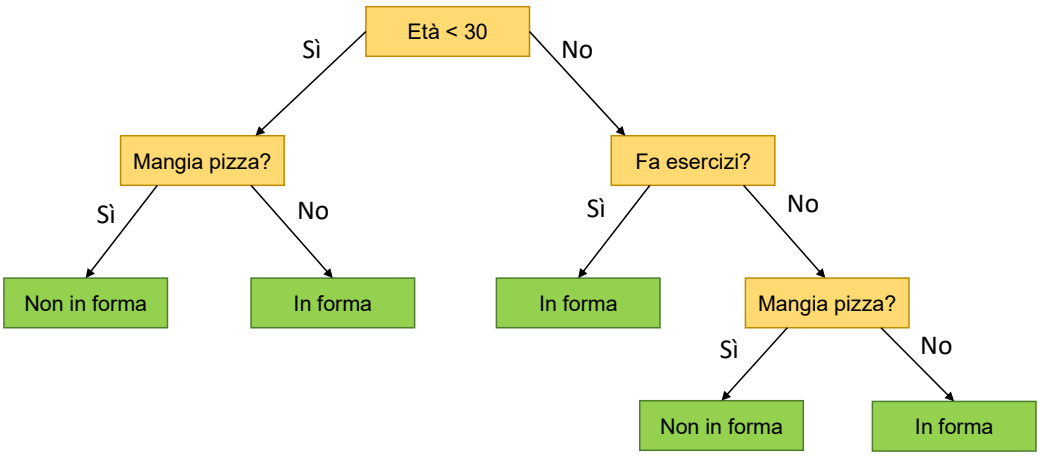


Figura 3.5: Esempio di albero decisionale.

Indici

Gli indici utilizzati da un albero decisionale agiscono in diverse modalità sulla costruzione dello stesso. Questo perchè ogni indice valuta differentemente come una feature possa dividere il dataset.

Tutti gli indici comunemente valutano quanto valga la probabilità che una classe k dell'insieme delle classi K sia presente nel nodo m ed è simboleggiata da $\hat{p}_{mk}$. Tra gli indici ce ne sono principalmente tre: errore di classificazione, indice Gini ed entropia.

L'*errore di classificazione* (E) è l'indice più intuitivo in quanto analizza l'errore commesso osservando quanti dati non sono correttamente classificati. L'errore può essere descritto come l'inverso della probabilità della classe più comune:

$$E = 1 - \max_k(\hat{p}_{mk})$$

Questo indice però non è abbastanza sensibile, dunque solitamente non se ne fa largo uso.

L'*indice Gini* prende in considerazione la varianza tra le K classi ed è definito dalla seguente equazione:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Come è possibile vedere dalla formula, per valori di probabilità vicini a zero o uno l'indice assume valori piccoli. Questo significa che il nodo contiene prevalentemente le osservazioni di una singola classe, caratteristica che può essere definita *purità di un nodo*.

L'*Entropia* è un indice alternativo a quello Gini descritto dalla seguente relazione:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \cdot log_2[\hat{p}_{mk}]$$

Come nel caso precedente, per probabilità prossime a zero o uno l'indice assume valori piccoli.

Costruzione dell'albero

La costruzione dell'albero, dopo aver selezionato l'indice da utilizzare per la scelta delle caratteristiche, avviene attraverso i seguenti passaggi:

1. l'intero dataset viene utilizzato come ingresso nel primo nodo detto *nodo padre*;
2. per ogni attributo non ancora utilizzato si calcola il valore dell'indice che gli è associato;
3. si calcola l'indice di impurità dei dati in ingresso al nodo;
4. tra tutti gli indici calcolati nel passaggio 2 si sceglie quello più basso che poi viene confrontato con quello prodotto nel passaggio 3;
5. il dataset viene diviso dall'attributo scelto producendo due sottoinsiemi oppure diventa una foglia nel caso sia stato scelto l'indice riferito a quello calcolato nel passaggio 3;
6. l'algoritmo ripete i passaggi da 2 a 5 per ogni sottoinsieme fino ad utilizzare tutti gli attributi del dataset oppure fino al massimo raggiungimento di profondità imposto dal limite.

Durante il punto 2 si analizzano tutti gli attributi e il modo in cui suddividono i dati. Per ognuna di queste suddivisioni viene calcolato il valore dell'indice e per poi effettuarne una media ponderata, utilizzando come pesi il numero di dati per sottoinsieme; questa media rappresenta il valore di impurità del nodo.

Nel caso l'attributo sia categorico[8] il calcolo dell'indice avviene per tutte le combinazioni possibili tra la feature e i valori distinti che assume nel dataset. Nel caso invece la feature sia numerica, si prendono tutti i valori distinti che appaiono nel dataset e li si ordina. Una volta ordinati ne si calcolano le medie a due a due per poi utilizzarle come discriminanti dell'attributo. A questo punto, come per le caratteristiche categoriche, si misura l'indice per ogni combinazione possibile.

Uno dei problemi degli alberi decisionali è che, se la quantità degli attributi di un dataset è molto numerosa, si può incorrere nel fenomeno dell'overfitting. Infatti, se lasciato libero durante la costruzione, l'albero cercherà di utilizzare tutte le feature fino a classificare correttamente ogni dato.

Per questo motivo alla costruzione dell'albero decisionale possono essere imposti alcuni vincoli, tra cui un limite sulla profondità, ovvero il numero massimo di suddivisioni accettabili. Questa condizione permette di fermare la costruzione dell'albero e di non utilizzare tutti gli attributi del dataset.

Foresta casuale

Lo svantaggio principale degli alberi di decisione può essere lo scarso livello predittivo in quanto la precisione del modello è interamente basata sulle caratteristiche scelte in fase di costruzione. Infatti gli attributi vengono scelti come ottimi locali e dunque non si ha una visione completa in quanto la stessa caratteristica avrebbe potuto dare un risultato migliore in una posizione differente. Questo avviene in particolare nel caso della feature del nodo padre che risultata essere fondamentale anche per tutte le successive.

Un'altra delle problematiche che coinvolgono gli alberi decisionali è quella di non performare bene quando i valori della varianza legati al dataset sono troppo alti o troppo bassi. Questo significa che il modello può incorrere facilmente in fenomeni di overfitting e underfitting.

Si può tentare di superare questa problematica utilizzando tecniche che permettono di casualizzare l'input, come il bootstrap. *Bootstrap* è una tecnica che permette di creare B sottoinsiemi di dati partendo dal dataset originale. Esso seleziona casualmente alcune osservazioni che vengono utilizzate per comporre i sottoinsiemi. La selezione avviene tramite rimpiazzo, ciò significa che i dati prelevati dal dataset vi vengono poi reinseriti, così alla successiva creazione di un sottoinsieme possono ricapitare le stesse osservazioni presenti in un altro.

Il *bagging* è una tecnica che unisce bootstrap con l'ensemble learning, ovvero la combinazione di più modelli per risolvere un singolo problema di previsione e combinarne i risultati per migliorare la precisione. Infatti il bagging si basa sul generare più modelli detti *stimatori* ognuno dei quali viene addestrato tramite uno dei dataset prodotti dal bootstrap.

Questa tecnica permette così di ridurre la varianza media dei dati osservati comportando una maggiore accuratezza. Quando si effettua una classificazione di un'osservazione la classe da associare viene votata a maggioranza tra tutte le B predizioni ottenute dai vari stimatori, vedere la Figura 3.6.

Nel caso degli alberi decisionali un modello che utilizza il bagging per migliorare l'accuratezza è la *foresta casuale* che è infatti composta da alberi costruiti tramite dataset differenti generati con bootstrap.

[8] Un attributo che assume un numero finito di valori distinti.

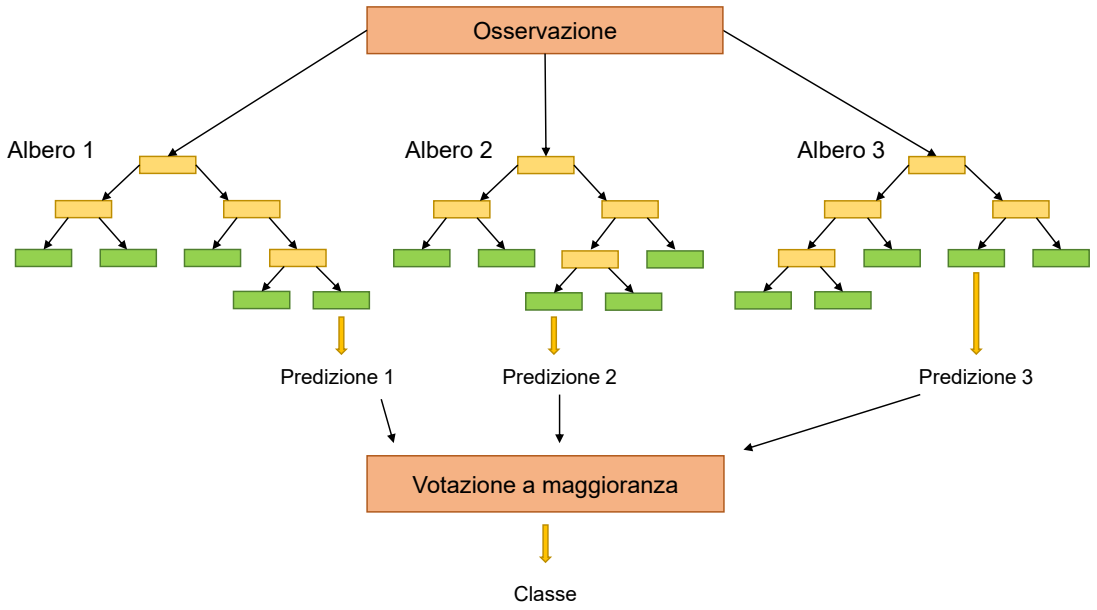


Figura 3.6: Schema di una foresta casuale.

3.2.2 Reti neurali

Una *rete neurale* è un modello computazionale composto da neuroni artificiali, che si ispira a una rete neurale biologica. La sua unità base è il neurone.

Neurone

Un *neurone* nel paragone biologico rappresenta il collegamento tra i dati di input e output. Il neurone, come descritto in [29], è composto da una combinazione lineare tra le caratteristiche dei dati in ingresso e i relativi pesi, come schematizzato nella Figura 3.7. In uscita al neurone c'è una funzione non lineare detta *funzione di attivazione*.

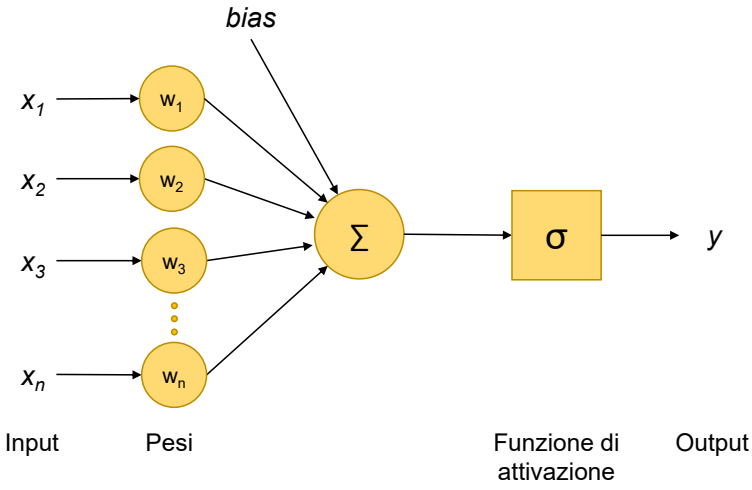


Figura 3.7: Schema di un neurone.

Dal punto di vista matematico possiamo vedere il neurone con la seguente equazione:

$$\hat{y} = \sigma(< w, x > + b)$$

dove σ è la funzione di attivazione, w sono i pesi, x le caratteristiche in input, $< x, w >$ è il prodotto scalare tra w e x, b è il bias ovvero un termine costante che non dipende dall'input e $\hat{y}$ è la predizione.

La *funzione di attivazione* determina il comportamento del neurone in base al suo output. Essa ha la funzione di introdurre una non-linearità sull'output derivante dalla combinazione lineare permettendo di aumentare la capacità discriminativa del modello e conseguentemente di aumentare il potere cognitivo del neurone.

Nella sua versione più classica il neurone utilizza come funzione di attivazione la funzione segno, ovvero una relazione che per valori minori di zero restituisce *0* e per valori maggiori o uguali a zero *1*. Grazie ad essa il neurone è in grado di trasformare gli output della combinazione lineare in valori binari e costituire così un classificatore binario.

In una rete neurale, rispetto al neurone più basilare e classico, la funzione di attivazione utilizzata viene cambiata, la funzione segno viene sostituita con una funzione detta sigmoide.

La curva sigmoide è una funzione matematica che come output ha valori compresi tra zero e uno, come si può vedere nella Figura 3.8, ed è descritta dalle seguente relazione:

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

dove x rappresenta l'input della funzione.

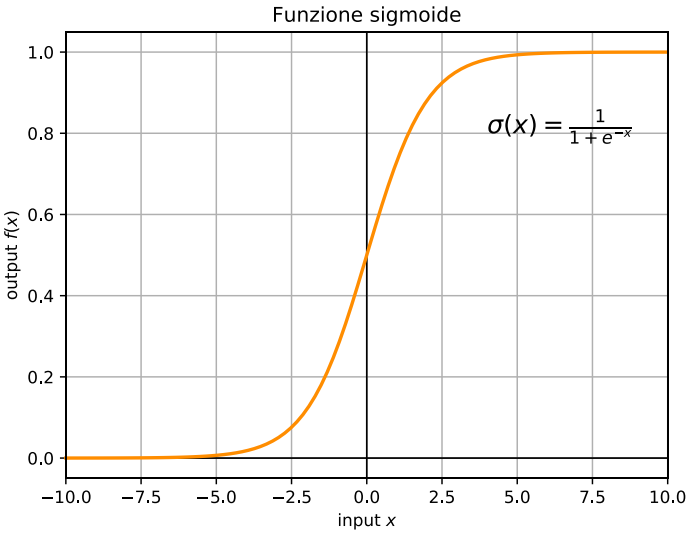


Figura 3.8: Funzione sigmoide.

Come suggerito in [30], la funzione sigmoide nelle reti neurali è divenuta meno frequente come funzione di attivazione in favore della *funzione di rettificazione o rampa* detta Rectified Linear Unit (ReLU). Questa funzione annulla tutti gli output minori di zero e restituisce l'input in caso sia positivo (vedere Figura 3.9) e può essere descritta dalla seguente relazione:

$$f(x) = \max(0, x)$$

dove x rappresenta l'input della funzione.

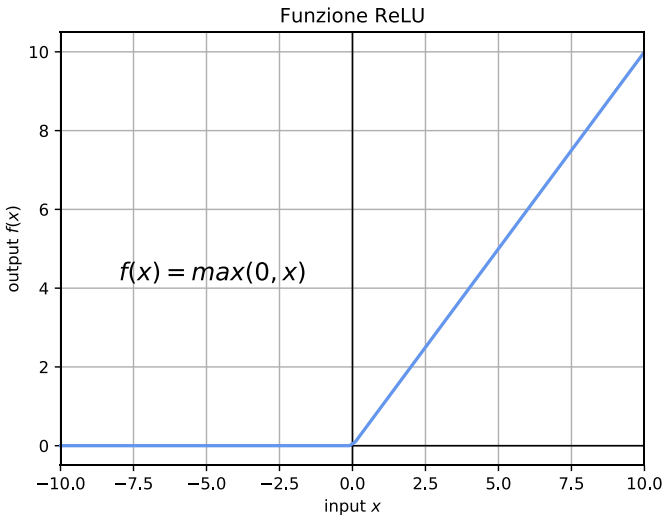


Figura 3.9: Funzione ReLU.

Una funzione di attivazione particolarmente utilizza per il livello di output è quella denominata *softmax*. Si tratta di una funzione che consente di trasformare i valori che escono da una rete neurale in probabilità di appartenenza alle classi. Utilizzando alcune soglie infatti è possibile eseguire classificazioni multiclasse. Softmax può essere descritto dalla seguente relazione:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

dove K è il numero di classi e x è l'input della funzione.

La funzione softmax normalizza il vettore x in ingresso e permette di individuare la distribuzione di probabilità per ogni classe.

Rete neurale completamente connessa

Il neurone si può considerare come il più semplice modello di rete neurale, anche chiamata rete neurale a singolo livello. Le reti neurali sono caratterizzate dall'utilizzo di più neuroni, detti nodi, disposti su uno o più livelli connessi tra di loro.

Il livello di una rete neurale direttamente connesso con i dati in ingresso si chiama livello di input, quello connesso alla predizione livello di output e tutti gli altri vengono denominati livelli nascosti.

Esistono due tipologie di reti neurali una detta *feed forward* e un'altra detta *ricorrente*.

La prima tipologia è una rete che ha connessione tra i nodi solamente con nodi di livelli successivi e non consente connessioni all'indietro.

La seconda invece prevede delle connessioni anche all'indietro creando dei cicli, il che permette di avere un effetto simile alla memoria a breve termine e serve per analisi di dataset in cui la sequenzialità è importante. In questo studio si prende in considerazione solamente il caso di una rete neurale feed forward.

In entrambe le tipologie, nel caso vengano utilizzati diversi livelli nascosti, l'apprendimento automatico prende il nome di *deep learning*. Questa categoria di apprendimento permette di creare modelli in grado di estrapolare diversi livelli di rappresentatività, riconoscendo caratteristiche in modo gerarchico.

Nel caso in cui ogni nodo sia connesso ad ogni altro nodo del livello successivo si dice che è una *rete neurale completamente connessa*, ne è un esempio la Figura 3.10.

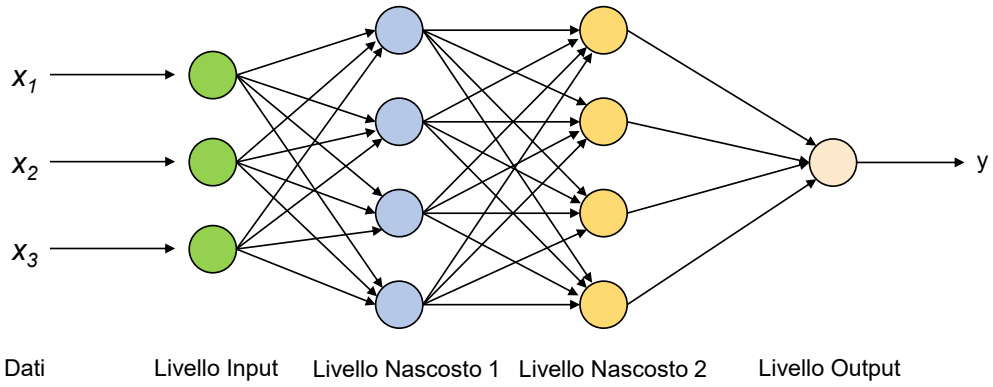


Figura 3.10: Schema di una rete neurale.

Apprendimento

L'apprendimento di una rete neurale consiste nell'individuare il valore dei pesi w e del bias b che permettono di predire correttamente le classi.

Solitamente, per un problema di classificazione con un dataset con d feature, si utilizza una rete neurale che abbia un livello di input composto da d nodi, un livello di uscita con tanti nodi quante sono le classi da predire nel caso di classificazione multiclasse e uno solo nel caso di classificazione binaria.

L'algoritmo si basa sul propagare l'output di ogni neurone in avanti verso quelli successivi ai quali è collegato, utilizzando come input un'osservazione alla volta. Ogni caratteristica del dato in input viene associata ad un nodo del livello di input. Questa operazione è detta *forward propagation* o propagazione in avanti.

La differenza tra l'uscita prodotta dalla rete neurale e il valore vero rappresenta l'errore commesso dalla rete. Tale valore viene utilizzato nell'algoritmo per modificare i pesi della rete. Questo procedimento avviene tramite una tecnica denominata *back propagation* o propagazione all'indietro, partendo dall'output l'aggiornamento si effettua livello per livello fino a ritornare a quello di input.

La *funzione di loss* quantifica quanto la predizione effettuata sia discostata dal valore reale. Questa funzione determina il costo della previsione fatta dal modello, nel caso sia sbagliata rappresenta la penalità da utilizzare. Esistono diversi tipi di funzioni di loss, in questo studio ne viene presa in esame una nota come entropia incrociata.

La funzione *entropia incrociata*, come descritto in [31], è utile per i problemi di classificazione e permette di specificare differenti pesi per ogni classe. Questo la rende adatta anche per dataset sbilanciati perché, utilizzando pesi diversi, cambia l'impatto di ogni dato appartenente ad una determinata classe durante l'apprendimento, permettendo ai dati presenti in minore quantità di non essere sovrastati dagli altri.

Un esempio di un dataset sbilanciato potrebbe essere il seguente: 10000 osservazioni divise in due classi, *blu* e *rosso*, 9500 dati nella prima e 500 nella seconda. Per bilanciare si possono utilizzare diverse tipologie di pesi, tra le più semplici c'è quella che si basa sulla frequenza. I pesi si ottengono calcolando l'inverso del numero di osservazioni di una determinata classe. In questo esempio il peso per la classe *blu* sarebbe di $w_{blu} = \frac{1}{9500} = 0.000105$ e per la classe *rosso* sarebbe $w_{rosso} = \frac{1}{500} = 0.002$.

Entropia incrociata (J_{cce}) può essere descritta dalla seguente relazione:

$$J_{cce} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} w_i \cdot p(y_{nc}) \cdot log[p(\hat{y}_{nc})]$$

dove N sono il numero delle osservazioni, C il numero delle classi, w il vettore dei pesi selezionati per ogni classe, $p(y_{nc})$ e $p(\hat{y}_{nc})$ la probabilità che la previsione sia la classe c.

Come è possibile osservare nella formula appena descritta, i pesi delle classi modificano l'impatto dell'errore sull'apprendimento. Se applicata all'esempio precedente gli errori di classificazione della *rossa* sono significativamente più penalizzanti di quelli della classe *blu*.

Come descritto nella Sezione 3.2.2, la tipologia di rete neurale utilizzata in questo studio prende il nome di *feed forward*, questo nome è lo stesso utilizzato per la procedura base della predizione del modello. Essa consiste in due passaggi principali: determinare la predizione utilizzando i parametri della rete neurale e calcolare l'errore tramite una funzione di loss ($J(w, x)$) in funzione dei pesi della rete neurale (w) e dell'input (x).

L'insieme di questi passaggi permette sia di calcolare l'output del modello attuale sia di individuare l'errore compiuto per poter eseguire poi delle correzioni tramite la procedura di back propagation.

L'*ottimizzatore* invece determina le regole di aggiornamento dei pesi, la scelta di questo metodo influenza anche i tempi di apprendimento in quanto può accelerare la convergenza della rete.

La scelta della modalità di aggiornamento dei pesi avviene in base alla selezione della funzione loss e dell'ottimizzatore. Come spiegato in [32], tali funzioni, insieme ai nodi e ai livelli, sono le componenti principali che costituiscono una rete neurale.

Esistono diversi tipi di ottimizzatori ma, come descritto in [33], i migliori sono quelli che fanno uso del gradiente.

La *discesa stocastica del gradiente* (SGD) è un ottimizzatore per implementare la back propagation tramite gradiente. SGD è un metodo iterativo che cerca di minimizzare la funzione di loss scelta, ogni iterazione sul dataset è detta *epoca*. Per ogni epoca il dataset viene ordinato casualmente e poi suddiviso in porzioni di uguali dimensioni dette *mini-batch*, che verranno inseriti nella rete sequenzialmente.

Questo metodo va unito ad un parametro detto *tasso di apprendimento* (η) che identifica quanto la rete deve apprendere ad ogni correzione. Tale parametro influenza i tempi di apprendimento e della convergenza: nel caso sia troppo piccolo sarà lenta, se invece fosse troppo grande l'apprendimento potrebbe oscillare o divergere.

Utilizzando SGD il processo di apprendimento segue i passaggi espressi tramite pseudo-codice nell'Algoritmo 1.

La procedura di back propagation invece consiste nell'aggiornamento dei pesi dei vari livelli della rete: si calcola il gradiente della funzione di loss rispetto al peso del nodo che si sta aggiornando, moltiplicato per il tasso di apprendimento. L'aggiornamento del peso può essere descritto dalla seguente relazione:

$$w = w - \eta \cdot \frac{\partial J}{\partial w}$$

dove J è la funzione di loss, w il vettore dei pesi di tutti i neuroni di un livello e η il tasso di apprendimento.

Il *momento* è una variante di SGD che permette di tenere traccia dell'ultimo aggiornamento dei pesi effettuato e utilizzarlo per migliorare il processo di apprendimento.

Da SGD derivano inoltre anche altri ottimizzatori tra cui: *gradiente adattivo* o adaptive gradient (AdaGrad) e *propagazione quadratica media della radice* o root mean square propagation (RMSProp).

Il primo utilizza un tasso di apprendimento indipendente per ogni parametro e, grazie a questa caratteristica, migliora i problemi con gradienti piccoli; il secondo utilizza anch'esso un tasso di apprendimento adattivo per ogni parametro che però si adatta in base alla media mobile del momento, il che gli permette di operare con buoni risultati anche con dataset con del rumore.

L'ottimizzatore *Adam* (stima del momento adattiva o adaptive moment estimation), come descritto in [34], è considerato lo stato dell'arte tra questa tipologia di ottimizzatori ed estende

Algoritmo 1 Algoritmo SGD

Inizializza w casualmente
Impostare num_{epoch} a un numero fissato
$epoca, totErr \leftarrow 0$
ripeti
    $epoca \leftarrow epoca + 1$
    riordinamento casuale del dataset
    suddivisione del dataset in mini-batch
    per ogni mini-batch **esegui**
        ri-inizializzazione gradiente
        $totErr \leftarrow 0$
        per ogni osservazione in mini-batch di dimensione n **esegui**
            procedura FEED FORWARD(x)
                $\hat{y} \leftarrow net(x)$
                $totErr \leftarrow totErr + J(w, x)$
            fine procedura
            procedura BACK PROPAGATION(x)
                aggiornamento pesi nascosti-ouput
                aggiornamento pesi input-nascosti
            fine procedura
        fine per ogni
        aggiornamento dei pesi w con il tasso di apprendimento η
    fine per ogni
    calcolo dell'errore medio $loss \leftarrow totErr/n$
    calcolo dell'accuratezza su training set e validation set
fino a che non c'è convergenza e $epoca < num_{epoch}$

i due appena descritti. Adam utilizza un tasso di apprendimento (η) adattivo basandosi non solo sulla media mobile del primo momento (m), ma anche su quella del secondo momento (v). Il momemento, come descritto precedente, è un termine di aggiornamento dipendente dalle iterazioni e dal gradiente della funzione di loss.

Questo ottimizzatore utilizza due parametri β_1 e β_2 per controllare il tasso di decadimento dei momenti calcolato tramite la funzione di loss (J). Introduce inoltre un fattore di *smoothing* (ϵ) per la stabilità numerica[9] dell'algoritmo.

L'applicazione dei diversi ottimizzatori incide sulla procedura di back propagation, nel caso di Adam la procedura si trasforma come descritto nell'Algoritmo 2, dove ∇ rappresenta il gradiente e J_t la funzione di loss calcolata nell'iterazione t.

Algoritmo 2 Algoritmo Adam

Inizializza m_0 e v_0 a 0
per ogni $t = 1, \ldots, T$ **esegui**
    $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla J_t(w_{t-1})$ stima del primo momento
    $v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla J_t(w_{t-1})^2$ stima del secondo momento
    $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ correzione del bias del primo momento
    $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ correzione del bias del secondo momento
    $w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$ aggiornamento dei pesi
fine per ogni
Ritorna w_T

[9] È una proprietà auspicabile degli algoritmi numerici e solitamente ne indica la capacità di non divergere, anche in presenza di dati in ingresso con delle fluttuazioni.

Da esperimenti effettuati descritti in [34], l'algoritmo di Adam converge prima rispetto a quelli precedentemente citati. Può essere inoltre utilizzato per problemi di ottimizzazione non convessi[10] ottenendo buoni risultati.

[10]Un problema di ottimizzazione che può avere più soluzioni locali ed è difficile verificare se la soluzione trovata sia l'unica.

Capitolo 4

Progettazione della soluzione

L'obbiettivo di questa tesi è dimostrare la possibilità di individuare l'utilizzo di strumenti di scanner di vulnerabilità su di una rete tramite modelli di apprendimento automatico che non eseguono Deep Packet Inspection[1] (DPI).

Questo è reso possibile dalle caratteristiche che i modelli di machine learning possiedono, in particolare la capacità di correlare i dati in ingresso con un determinato output.

In questo capitolo vengono quindi illustrati i seguenti punti:

- una panoramica delle possibili soluzioni;
- l'architettura della soluzione proposta;
- lo strumento utilizzato per estrapolare le metriche impiegate per l'analisi;
- lo strumento usato per l'automatizzazione;
- i modelli utilizzati come classificatori.

4.1 Panoramica

Come introdotto nel Capitolo 1, la tendenza delle tipologie di traffico nel mondo sta cambiando spostandosi verso un maggior impiego di protocolli cifrati, per esempio il protocollo HTTPS. La natura di questi protocolli prevede spesso l'utilizzo di crittografia end-to-end, ovvero tra client e server, rendendo impossibile per gli ISP[2] o per software aziendali l'utilizzo di strumenti di analisi che eseguono DPI per determinarne la bontà.

I protocolli cifrati, come visto nella Sezione 2.1.2, utilizzano diverse tecniche per cifrare, autenticare e validare i pacchetti di rete. Questa peculiarità però rende impossibile riuscire, per macchine che non siano il client finale, decifrare il contenuto dei pacchetti di rete per poterlo analizzare.

Esistono alcune alternative che permettono di superare questa limitazione, di seguito ne vengono riportate tre tipologie:

- installare strumenti di protezione direttamente su tutti i client;
- utilizzare sistemi di proxy che permettono di fare da intermediario tra il client e il server;
- utilizzare strumenti di analisi tra client e server che non utilizzano DPI.

[1] Analisi dettagliata del payload di un pacchetto di rete.

[2] Internet Service Provider, fornitori di connettività.

La prima alternativa prevede di modificare tutti i dispositivi che devono essere protetti installando direttamente software per la difesa, i quali dovranno, una volta decifrato il traffico, determinarne la natura. Questa tipologia di soluzione è difficile da manutenere, in quanto per ogni aggiornamento e/o modifica da effettuare al software si richiede l'intervento su ogni singolo client, soprattutto per le grosse aziende e per gli ISP data la portata dei terminali connessi. Inoltre i client sono spesso macchine non omogenee e con ridotte capacità computazionali, ragione per cui non possono garantire una protezione in tempo reale.

La seconda alternativa è l'utilizzo di proxy, ovvero un servizio intermediario che si interpone tra il client e il server, prendendo in carico le richieste del primo e inoltrandole al secondo e viceversa. Questi strumenti permettono quindi di fare da collegamento tra il client e il server decifrando e cifrando nuovamente il traffico per poterlo analizzare.

Tale tipo di soluzione è poco praticabile per via dei carichi computazionali elevati e perchè attuandola verrebbe a crearsi un *man-in-the-middle*[3] (MITM). Il proxy è infatti in grado di modificare e manipolare tutte le connessioni tra il client e il server senza che questi possano accorgersene nonostante entrambi ritengano affidabile il canale cifrato. Se poco protetto si potrebbe creare una superficie di rischio elevata che permetterebbe diversi tipi di attacchi informatici tra cui il phishing[4] e la modifica del software durante un download all'insaputa del client. Inoltre è in grado di leggere in chiaro i vari protocolli di rete, pratica che potrebbe entrare in conflitto con alcune normative vigenti in Europa in materia di privacy, come ad esempio il *Regolamento Generale sulla Protezione dei Dati*[5] (GDPR).

La terza alternativa è quella percorsa dal seguente studio, utilizzando le caratteristiche dell'apprendimento automatico si cerca di costruire le basi per un software in grado di individuare la natura del traffico senza eseguire analisi dei payload nei segmenti TCP. Questo tipo di soluzione permette quindi di superare le difficoltà introdotte dai protocolli sicuri, dovendo però sostituire i software di analisi esistenti.

4.2 Flusso di esecuzione

Utilizzando l'approccio descritto nella Sezione 4.1, l'analisi per il riconoscimento degli scanner di vulnerabilità è composta principalmente dall'esecuzione di un flusso TCP nel quale è necessario individuare una traccia lasciata da particolari strumenti.

Il flusso di esecuzione proposto per l'analisi, partendo dal traffico di rete fino alla sua classificazione, è connesso come riportato nella Figura 4.1. Tale flusso è composto da una parte di elaborazione di rete tramite Netstatpy e una di addestramento dei classificatori tramite Vulnscanpy, spiegati successivamente.

La soluzione proposta prevede quindi l'impiego di uno strumento in grado di calcolare delle metriche per ogni flusso TCP, vedere Sezione 2.1.1, a partire dai segmenti che lo compongono, senza però interpretare il loro payload. In questo modo si crea un nuovo dataset nel quale vengono enfatizzate le caratteristiche intrinseche del flusso.

Il passo successivo consiste nell'utilizzare questo dataset per addestrare, tramite machine learning, dei classificatori in grado di distinguere traffico legittimo e traffico contenente scansioni di vulnerabilità.

Estrapolando l'essenza di questo tipo di architettura è possibile notare come sia composta principalmente da uno strumento in grado di calcolare metriche TCP, e dunque non legato a protocolli applicativi specifici, e da un dataset correttamente etichettato. Questa architettura

[3]Tipologia di attacco informatico secondo il quale l'attaccante si pone tra due interlocutori e ne intercetta la connessione. Attacco che protocolli cifrati come HTTPS cercano di contrastare.

[4]Il phishing è una truffa informatica nella quale un malintenzionato è in grado di sfruttare la disattenzione dell'utente fingendosi un ente affidabile e permettendo così il furto di dati sensibili quali password o informazioni personali.

[5]https://www.garanteprivacy.it/il-testo-del-regolamento

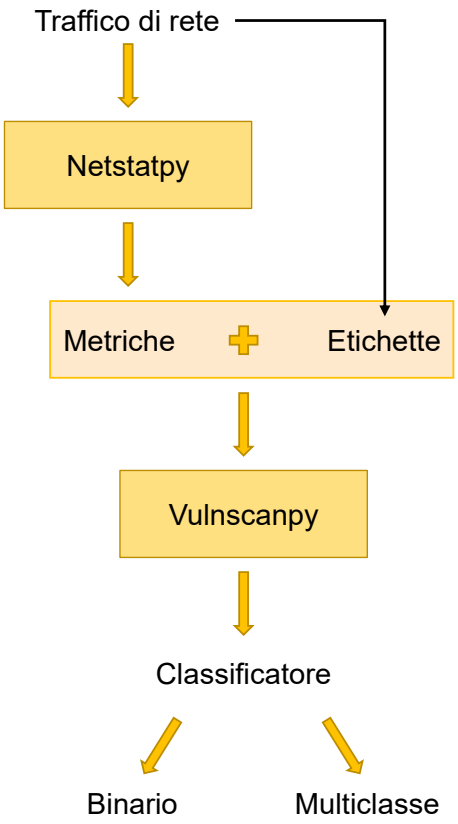


Figura 4.1: Schema del flusso di esecuzione utilizzato per l'addestramento dei classificatori.

può essere utile anche per analizzare altri tipi di tracce, differenti dagli scanner di vulnerabilità, come ad esempio il traffico generato da attacchi DDoS[6].

Tenendo conto di queste caratteristiche e della possibilità di impiegare nuovamente questa architettura anche per studi futuri, si è deciso di creare da zero degli strumenti in grado di essere facilmente riutilizzabili e che possano essere espansi per rispondere alle diverse esigenze. Infatti, per mantenere l'architettura più semplice, si è scelto di dividerla in due parti creando due software, uno per l'estrazione delle metriche e uno per la parte di apprendimento automatico.

Come primo passaggio dell'analisi il traffico di rete viene analizzato tramite lo strumento *Netstatpy*, descritto nella Sezione 4.3, che compone il nuovo insieme di dati. Successivamente, utilizzando il traffico di partenza, il dataset viene etichettato con due tipologie diverse di classi. La prima tipologia serve per eseguire una classificazione binaria volta alla distinzione tra traffico buono e malevolo, la seconda serve per eseguire una classificazione multiclasse per riuscire a distinguere i diversi scanner di vulnerabilità.

Uno degli scopi di questa analisi è riuscire a classificare il traffico senza basarsi su attributi che possano essere facilmente aggirabili tramite delle tecniche di spoofing[7]. Tra questi vi sono l'indirizzo IP delle macchine e le porte TCP utilizzate sia per esporre un servizio di un client che dall'attaccante per eseguire test. Il dataset con le metriche va quindi anonimizzato rimuovendo i campi che potrebbero identificare la particolare connessione e non permettere una corretta generalizzazione. In seguito il dataset viene preprocessato per l'analisi e suddiviso nei vari sottoinsiemi utilizzati per l'apprendimento, la validazione e il test.

[6]Distributed Denial of Service, una tipologia di attacco che utilizza un vasto numero di client per eseguire molte connessioni ad un server vittima con l'obbiettivo di creare un disservizio.

[7]Lo spoofing è una tipologia di attacco informatico che permette di falsificare l'identità di un sistema, cercando di presentarsi come un altro.

Una parte fondamentale è la ricerca degli iperparametri più adatti per i modelli, vedere Sezione 3.1.3, affinchè possano classificare al meglio il traffico e, una volta individuati, si esegue la valutazione del modello e delle performance ottenute. Per automatizzare questi processi legati alla lavorazione del dataset e all'apprendimento dei classificatori è stato utilizzato il programma *Vulnscanpy*, descritto nella Sezione 4.4.

Una volta che i modelli sono pronti possono essere utilizzati per eseguire classificazioni anche di traffico non etichettato.

4.3 Netstatpy

Netstatpy è uno strumento di estrazione delle metriche ed è stato realizzato appositamente per questa tesi. Questo tool è scritto in `Python` ed è un programma utilizzabile singolarmente oppure come libreria integrabile in altri progetti e lo si può sia espandere che agganciare in processi più complessi.

Netstatpy utilizza come base la libreria Scapy, illustrata nella Sezione 2.2.5, ed è quindi in grado di manipolare i pacchetti di rete grezzi, ovvero non ancora elaborati dallo stack di rete in quanto intercettati dal BPF e copiati direttamente nello spazio utente.

Questa caratteristica del tool permette di creare filtri personalizzati e analizzare solamente i pacchetti interessati. Il che permette inoltre di agire sul pacchetto e analizzare solamente l'intestazione del segmento senza utilizzare risorse della macchina per elaborare i protocolli applicativi.

Tale funzione ha giocato un ruolo importante nella scelta di creare un nuovo strumento invece di utilizzare un software di analisi come Tstat. Come spiegato e anticipato nella Sezione 2.2.4, Tstat è un analizzatore di traffico difficilmente espandibile e poco personalizzabile, caratteristiche che rendono complessa l'aggiunta di nuove statistiche o metriche.

Nella Figura 4.2, che riporta lo schema del flusso di esecuzione di Netstatpy, si può osservare come il software elabora e analizza il traffico tramite tre passaggi principali: *preprocessamento*, *estrazione delle statistiche* e *generazione dei risultati*.

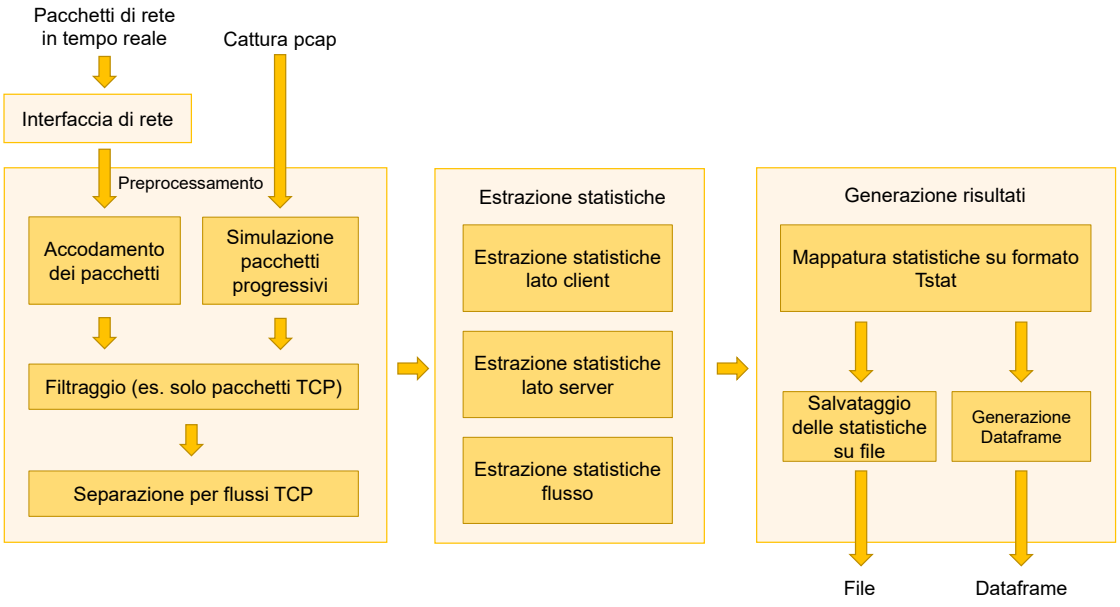


Figura 4.2: Schema del flusso di esecuzione di Netstatpy.

Il preprocessamento si occupa di trattare il dataset per renderlo idoneo alle parti successive; l'estrazione delle statistiche, tramite delle funzioni, calcola le metriche dei pacchetti di rete; infine la generazione dei risulti si occupa di trasformare i valori calcolati sia in formati predefiniti sia in formati compatibili con flussi automatizzati.

4.3.1 Preprocessamento

I pacchetti di rete possono essere passati a Netstatpy sia tramite l'interfaccia di rete sia tramite catture preesistenti in formato pcap (vedere Sezione 2.2.1). Nella parte di preprocessamento i pacchetti vengono accodati nel caso arrivino uno alla volta dall'interfaccia di rete, oppure vengono elaborati uno alla volta durante la lettura del file pcap simulando quindi un flusso continuo. Tali operazioni servono per permettere l'analisi dell'evoluzione del flusso TCP.

L'analisi di Netstatpy, a differenza di Tstat, può essere facilmente effettuata per pacchetto e non direttamente su un'intera cattura. Tale impostazione è stata implementata per poter simulare un flusso reale di traffico e permettere ai classificatori di osservare l'evoluzione degli attacchi. Per ulteriori dettagli sull'utilizzo di questo strumento si rimanda al Manuale B.2.

Netstatpy, grazie alla sua integrazione con la libreria libpcap e l'interazione con il BPF, può creare dei filtri a livello kernel che permettono di analizzare solamente i pacchetti interessati. Oltre a questo può anche applicare ulteriori filtri in fase di elaborazione per scartare pacchetti con caratteristiche personalizzabili.

Nel flusso di esecuzione Netstatpy ha dei filtri preconfigurati che permettono di analizzare solo pacchetti che possiedono un'intestazione TCP e, come spiegato precedentemente, scartare gli altri. Successivamente i pacchetti vengono raggruppati e suddivisi in flussi TCP e Netstatpy può quindi procedere al calcolo delle metriche.

4.3.2 Estrazione delle statistiche

Netstatpy estrae le stesse metriche di Tstat presenti nella Tabella A.1, escluse quelle relative a protocolli applicativi, e a queste aggiunge anche quelle presenti nella Tabella A.2. L'insieme costituito dalle metriche di Tstat e dai campi extra, esclusi quelli dal 40 al 44 compresi, viene da qui in poi chiamato *formato tstat esteso*. Quindi in totale vengono estratte 63 metriche dal traffico.

Le metriche che sono state aggiunte rispetto a Tstat sono relative a conteggi dei flag presenti nell'intestazione dei pacchetti TCP, per esempio PSH o URG, vedere Sezione 2.1.1. Supponendo che gli scanner di vulnerabilità sfruttino qualsiasi tipologia di vulnerabilità per individuare falle nei sistemi, si è deciso di valutare l'utilità di monitore anche questi flag per caratterizzare meglio il dataset.

Netstatpy è stato pensato per essere personalizzabile e consentire di aggiungere facilmente ulteriori metriche. Il traffico di rete, come spiegato nella Sezione 2.1.1, è composto da una doppia comunicazione bidirezionale tra due host e per questo, in base a dove viene effettuata l'analisi, alcuni dei parametri possono cambiare. Per tale ragione le metriche sono state suddivise in tre categorie: client, server e di flusso.

Le statistiche client e server sono delle metriche che appartengono ciascuna al punto di vista della macchina che rappresentano: quelle client della macchina attaccante o che naviga regolarmente e quelle server della macchina che riceve le richieste TCP. Le statistiche di flusso invece sono metriche proprie dell'intera connessione TCP e tengono in considerazione entrambe le direzioni delle comunicazioni, per esempio la durata della connessione.

Vista la flessibilità dello strumento è possibile integrare delle metriche aggiuntive intervenendo sulle singole categorie. Ulteriori dettagli sono disponibili nel Manuale C.1.

4.3.3 Generazione dei risultati

Netstatpy per offrire la massima personalizzazione anche nella fase di generazione dei risultati, in particolare sul formato in uscita, utilizza un doppio sistema di mappatura. Esso è composto da due configurazioni separate: la prima indica quali metriche sono da calcolare durante l'analisi e la seconda serve a indicare come disporre ed etichettare le metriche.

La prima configurazione crea quindi un insieme di informazioni con uno schema non fisso, rendendo difficile il confronto, nel caso siano state cambiate le metriche ricercate, tra due output di analisi diverse. La seconda invece serve per superare il problema appena descritto introducendo la possibilità di mappare e ridisporre i dati calcolati in un formato standard di output, più precisamente il formato tstat esteso.

Nella Figura 4.3 viene riportato un esempio della mappatura appena descritta utilizzando la funzione `flags_stats`. Questa restituisce un oggetto contenente un dizionario che ha come chiave le iniziali dei flag TCP e come valori il conteggio per ognuno di essi in un flusso `session_c`. In un primo passaggio interno i flag vengono mappati, come da Figura, dalla funzione stessa che esporta queste metriche in altri passaggi del flusso di esecuzione. La seconda mappatura viene utilizzata in fase di formazione del formato di uscita, i campi nella Figura sono un estratto del formato tstat esteso. Come è possibile osservare, i vari flag vengono riordinati e alcuni di essi potrebbero anche non essere riportati nel formato finale.

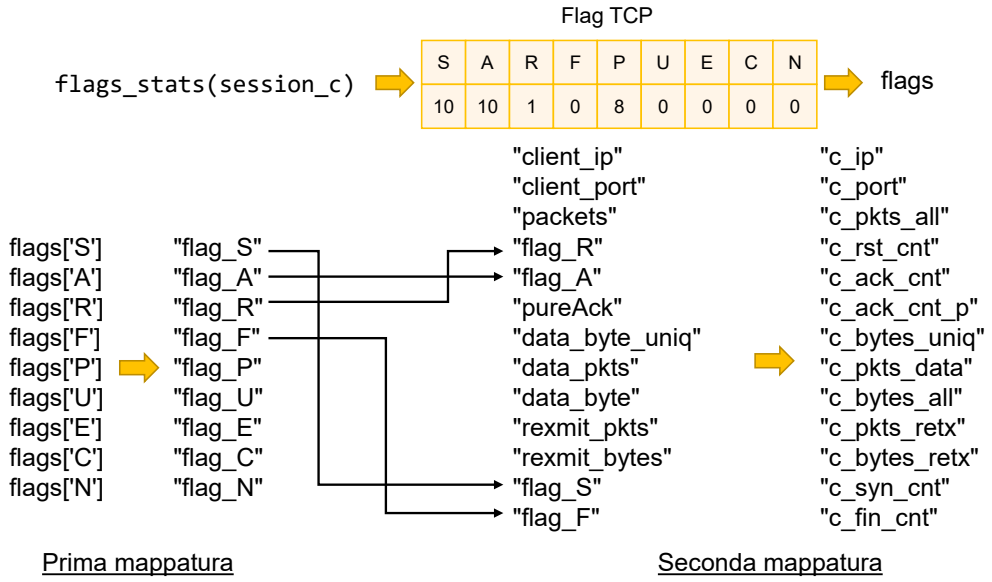


Figura 4.3: Esempio di doppia mappatura in Netstatpy della funzione `flags_stats`.

Tutte le eventuali metriche aggiuntive possono essere accodate al formato tstat esteso, in questo modo è anche possibile eseguire confronti con Tstat per valutare la validità del calcolo. Anche questo componente è personalizzabile potendo quindi implementare un differente formato di uscita. Nella Figura 4.3, come spiegato precedentemente, la funzione calcola i diversi flag TCP, alcuni di questi non sarebbero presenti nel formato tstat (vedere Tabella A.1), vengono quindi aggiunti tramite queste personalizzazioni (vedere Tabella A.2).

Le metriche calcolate vengono quindi salvate su file e/o, nel caso Netstatpy venga utilizzato come libreria all'interno di un processo automatizzato, possono essere esportate come un oggetto `Python` di tipo `Dataframe` definito nella libreria *Pandas*[8].

[8]Pandas è una popolare libreria per l'elaborazione e l'analisi dei dati. Sito del progetto: https://pandas.pydata.org/

Netstatpy è predisposto, grazie alla libreria Scapy, all'integrazione in un processo di analisi dei pacchetti in tempo reale, infatti sono previste e parzialmente implementate soluzioni per permettere questo tipo di elaborazione.

Questa funzionalità permette quindi a Netstatpy non solo di essere uno strumento utile durante la fase di studio, ma anche una possibile base di partenza per una eventuale implementazione di un sistema di rilevazione in tempo reale.

4.4 Vulnscanpy

Per automatizzare il processo di apprendimento dei classificatori è stato realizzato un altro programma chiamato *Vulnscanpy*, un tool scritto in `Python` che integra le librerie per machine learning *Scikit-learn*[9] e *Pytorch*[10]. La prima è una popolare libreria open source per eseguire diversi tipi di analisi sui dati, tra cui classificazione, regressione e clustering; offre altresì diverse funzioni per ulteriori parti dell'analisi come il preprocessamento del dataset e la valutazione del modello. La seconda è anch'essa una libreria open source per `Python` orientata all'addestramento di modelli più complessi rispetto a Scikit-learn come quelli di Deep Learning. Pytorch è stata primariamente sviluppata da Facebook[11] per applicazioni di computer vision e processamento di linguaggio naturale.

Questo strumento si basa su alcune opzioni che permettono di eseguire una serie di operazioni preconfigurate e tramite questo sistema è possibile creare dei flussi di esecuzione personalizzati. La flessibilità delle passaggi permette di creare delle biforcazioni all'interno della stessa configurazione che prendono in esame specifiche diverse. Lo schema del flusso di esecuzione del software è riportato nella Figura 4.4.

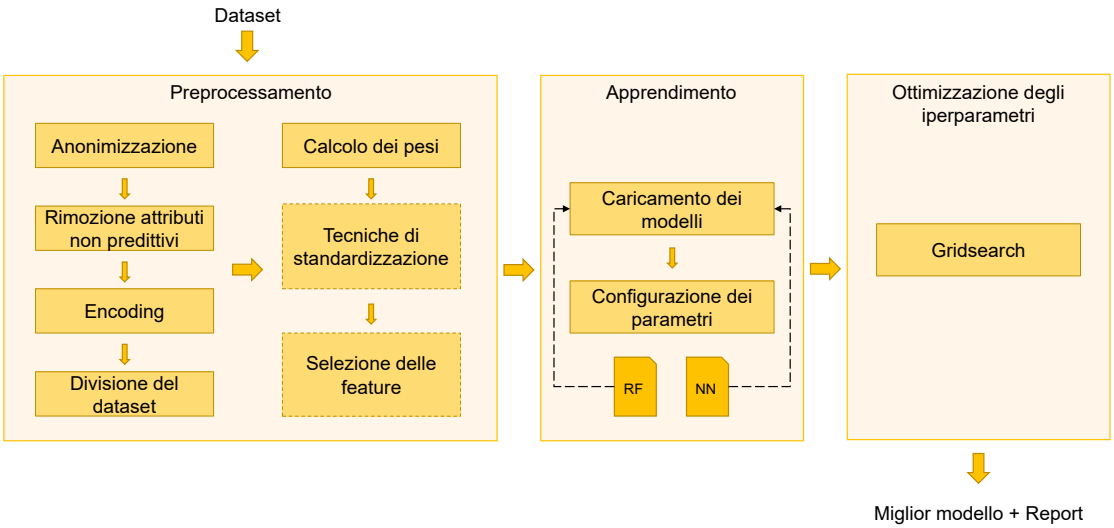


Figura 4.4: Schema del flusso di esecuzione di Vulnscanpy.

Come per Netstatpy, Vulnscanpy è composto da più parti per permettere una maggiore flessibilità, in particolare i dati sono analizzati in tre fasi consecutive: *preprocessamento*, *apprendimento* e *ottimizzazione degli iperparametri*.

[9] https://scikit-learn.org/

[10] https://pytorch.org/

[11] https://www.facebook.com/

Vulnscanpy sfrutta l'oggetto denominato *Pipeline*[12], definito nella libreria Scikit-learn, per predisporre le operazioni da eseguire in sequenza. I primi due moduli del flusso di eseguzione di Vulnscanpy servono a comporre gli argomenti dell'oggetto pipeline e l'ultimo ad eseguirlo per addestrare e valutare i classificatori.

Uno dei punti di forza di questa sequenza di operazioni è la possibilità di specificare più opzioni per gli stessi iperparametri, permettendo quindi di valutare più scenari che condividono una parte delle operazioni del flusso, ma cambiano il valore dell'iperparametro per ogni opzione.

Vulnscanpy cerca di standardizzare il processo di lavorazione del dataset e della parte di classificazione. A tale scopo lo strumento è stato pensato per poter aggiungere facilmente altri modelli di classificazione, modificare la pipeline di preprocessamento e le metriche di valutazione del modello. Si rimanda al Manuale C.2 per ulteriori approfondimenti legati alle modifiche del software.

Come per Netstatpy questo strumento è stato realizzato sia per poterne fruire singolarmente al fine di eseguire analisi su file pcap, sia per essere integrato in altri processi. Per una spiegazione più dettagliata sul suo utilizzo fare riferimento al Manuale B.3.

4.4.1 Preprocessamento

Il primo passaggio dell'analisi, quello di preprocessamento, è a sua volta composto da alcune funzionalità base che sono comuni durante analisi di machine learning.

Sono previste nel flusso di elaborazione diverse funzioni e per ognuna di esse è possibile specificare tecniche e parametri da testare. In particolare il modulo di preprocessamento, ha preesistenti le seguenti funzionalità:

- anonimizzazione dei dati rimuovendo indirizzo IP e porte TCP;
- rimozione degli attributi maggiormente influenzati dall'ambiente di test (es. attributi temporali);
- encoding delle etichette;
- divisione del dataset in quello di training, di validazione e di test.
- ribilanciamento del problema;
- tecniche di scalamento dei dati;
- selezione delle feature.

Le prime due operazioni consistono nel preparare il dataset per l'addestramento dei classificatori, verranno spiegate durante l'analisi del dataset nella Sezione 5.2. L'encoding delle etichette, o più in particolare il one-hot encoding come spiegato nella Sezione 3.1.1, è una funzione che permette di codificare i dati in valori numerici in quanto richiesto da alcuni modelli di apprendimento automatico, come la rete neurale. Come descritto nella Sezione 3.1.3, il dataset viene suddiviso in tre gruppi per le diverse fasi dell'analisi e successivamente vengono calcolati i differenti pesi per ribilanciare il dataset, come spiegato nella Sezione 3.1.1. Infine il dataset viene scalato e ridotto di dimensionalità per cercare di evitare il fenomeno dell'overfitting, come descritto nella Sezione 3.1.1 e nella Sezione 3.1.2.

Queste ultime operazioni hanno alcune implementazioni, che verranno esplorate nella Sezione 6.1.1, Vulnscanpy permette di specificare quali sono da utilizzare o quali, eventualmente, saltare. Inoltre è possibile selezionare per ogni configurazione, tramite l'oggetto pipeline, implementazioni diverse permettendo di individuare quelle che consentono di ottenere il risultato migliore.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

4.4.2 Apprendimento

Nella sezione dedicata all'apprendimento vengono gestiti i vari classificatori. In questo modulo vengono caricati i modelli che verranno addestrati con i diversi parametri di configurazione. Infatti è possibile aggiungere differenti classificatori ognuno dei quali può specificare alcuni iperparametri direttamente all'avvio.

Tramite una sezione dedicata ai modelli è possibile scegliere quali si vuole utilizzare per eseguire l'analisi. Per ognuno di essi è possibile specificare una configurazione nella quale vengono espressi diversi valori per gli iperparametri da ricercare e, successivamente al preprocessamento, questi vengono testati per individuare la combinazione che restituisce il risultato migliore.

Per questo studio sono stati creati due classificatori: foresta casuale e rete neurale, che verranno approfonditi rispettivamente nella Sezione 4.5.1 e nella Sezione 4.5.2.

4.4.3 Ottimizzazione degli iperparametri

Nell'ultimo modulo, viene finalizzato l'oggetto pipeline costruito dai moduli precedenti e selezionata le metriche di valutazione da adottare e quali si vogliono massimizzare.

Come è stato descritto nella Sezione 3.1.4, ci sono diversi tipi di indici che si possono osservare per valutare un modello. Nello specifico in questo studio è più rilevante massimizzare una corretta classificazione del traffico malevolo a discapito di accuratezza generale, questo perchè la quantità di traffico buono è molto elevata e sbilancia le percentuali di accuratezza.

Vulnscanpy, in base alla tipologia di classificazione binaria o multiclasse, prima di eseguire le operazioni descritte nella pipeline, calcola a runtime alcuni parametri come il numero delle classi o il tipo di encoding da utilizzare. In questo modo il flusso di esecuzione dell'apprendimento rimane invariato per entrambe le tipologie di analisi.

Terminata la fase di preparazione, vengono elaborate le operazioni descritte nella pipeline, in particolare vengono eseguite grazie alla funzione *grid search*, spiegata nella Sezione 3.1.3. Questa utilizza l'insieme di dati di apprendimento e per ogni alternativa dei parametri calcola il punteggio ottenuto sull'insieme di validazione, anche detto validation set. Poi per ogni classificatore utilizzato viene generato un report dettagliato con le diverse performance ottenute ed eseguito un test finale utilizzando il dataset di test.

4.5 Classificatori

I modelli utilizzati per eseguire l'analisi sono una foresta casuale e una rete neurale, in questa Sezione vengono spiegati i dettagli implementativi di questi due modelli.

In entrambi si fa utilizzo di parametri legati al peso delle classi in quanto, come verrà poi mostrato nel Capitolo 5, il dataset dell'analisi è sbilanciato. Questo serve anche per rappresentare al meglio la realtà, infatti, come è facilmente intuibile, la quantità di traffico legittimo è nettamente superiore alla quantità di traffico contenente gli strumenti di scansione di vulnerabilità che si vogliono individuare.

4.5.1 Foresta casuale

Come descritto nella Sezione 3.2.1 dedicata alla foresta casuale, questo tipo di modello può essere utilizzato per eseguire una classificazione sia binaria che multiclasse. Per realizzare questo modello si è utilizzato quello offerto da Scikit-learn `RandomForestClassifier`[13].

[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Questo modello offre diversi iperparametri, tra quelli più importanti per l'analisi c'è `class_weight`. Esso permette di associare un peso alle classi e di cambiarne quindi l'influenza. Se l'iperparametro è utilizzato nella configurazione *balanced*, i pesi si baseranno su una relazione inversamente proporzionale alla frequenza di diverse classi espressa dalla seguente formula:

$$weight_{class_1} = \frac{n_{sample}}{n_{class} \cdot class_1}$$

dove n_{sample} è il numero di osservazioni introdotte nel modello, n_{class} il numero distinto delle classi, $class_1$ è il numero di occorrenze per una classe $class_1$ e $weight_{class_1}$ il peso associato alla classe $class_1$.

Altri iperparametri utilizzati sono:

- *n_estimators* utilizzato per indicare il numero di stimatori impiegati;
- *max_depth* utilizzato per limitare il livello massimo di pronfondità dell'albero;
- *criterions* utilizzato per scegliere l'indice che calcola il valore di purezza dei nodi.

Queste ultime caratteristiche dell'albero verranno poi esplorate per individuare quelle migliori per i due tipi di classificazioni, tali iperparametri verranno approfonditi nella Sezione 6.1.1.

4.5.2 Rete Neurale

In questo studio si è deciso di utilizzare una rete neurale completamente connessa, descritta nella Sezione 3.2.2, e in particolare è stato impiegato il modello offerto da Pytorch `torch.nn`[14].

La scelta è ricaduta su questa libreria per via della sua popolarità, come descritto da He in *"The State of Machine Learning Frameworks in 2019"*[15], per il livello di personalizzazione offerto e per la quantità di documentazione reperibile.

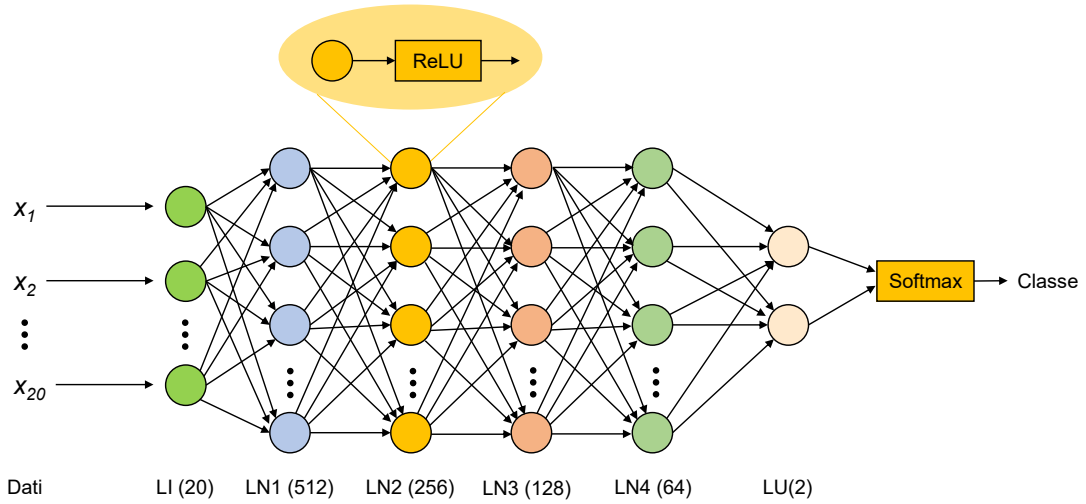


Figura 4.5: Schema della rete neurale utilizzata per l'analisi.

La struttura del modello utilizzata è così composta:

[14] https://pytorch.org/docs/stable/nn.html

[15] https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

- livello di ingresso con tanti nodi quanti solo gli attributi dell'osservazione introdotta nel modello, valore che viene calcolato a runtime[16] in quanto il dato viene preprocessato e ridotto di dimensione;
- all'uscita del livello di ingresso è stata collegata una funzione non lineare, ReLU o sigmoide – vengono valutate entrambe in fase di ottimizzazione tramite due percorsi della pipeline;
- livelli nascosti, in particolare la configurazione di essi, numero di nodi e il numero dei livelli, viene cambiato in base alle opzioni impostate per la rete neurale, come per il livello di input anche in questo caso è stata aggiunta la stessa funzione non lineare in uscita;
- livello di uscita nel quale il numero di nodi è fisso ed è pari al numero di classi da individuare, per esempio due nel caso del classificatore binario;
- nella parte finale del livello in uscita si è utilizzata la funzione lineare softmax (vedere Sezione 3.2.2).

La struttura appena descritta è possibile osservarla nella Figura 4.5, dove LI è il livello di ingresso, LN livello nascosto, LU livello di uscita e i numeri nelle parentesi rappresentano il numero di nodi totali del livello. Ogni nodo di LI e LN ha in uscita la funzione non lineare ReLU.

Come anticipato, il numero di livelli nascosti è variabile e dipende dalla configurazione adottata, infatti i numeri dei livelli e la quantità di nodi per ognuno di essi viene valutata tramite le opzioni descritte nella pipeline. Tra tutte quelle possibili viene scelta la combinazione che permette di ottenere il risultato migliore.

[16] Mentre il programma è in esecuzione.

Capitolo 5

Analisi del dataset

In questa capitolo verranno approfondite l'origine e la composizione del dataset utilizzato per l'analisi presente in questa tesi. In particolare saranno presi in esame gli aspetti legati:

- alla provenienza dei dati e la descrizione dettagliata del dataset;
- alla quantità di dati presenti e la loro distribuzione nelle varie classi;
- alle feature che sono state estratte tramite Netstatpy e la loro distribuzione;
- al confronto tra le estrazioni del traffico normale e del traffico malevolo.

5.1 Descrizioni dei dati di partenza e classi

Il gruppo di ricerca TORSEC[1] ha fornito alcune catture di traffico di rete sia legittimo sia malevolo. Il traffico malevolo è stato ottenuto eseguendo scansione di vulnerabilità tra due terminali in due reti diverse, utilizzando alcuni degli strumenti citati in precedenza, per poi essere catturato tramite tshark (spiegato nella Sezione 2.2.3). In particolare gli strumenti di scansioni utilizzati sono: OpenVAS, SQLmap e Wapiti rispettivamente presentati nelle Sezioni 2.3.5, 2.3.2 e 2.3.3.

Nello specifico le catture legittime sono pacchetti di navigazione ordinaria prodotti da diversi browser presenti su macchine con sistemi operativi differenti. Tali catture sono state raggruppate secondo la coppia costituita da browser e sistema operativo utilizzato. In totale i pacchetti di rete presenti sono 818616. Per il dettaglio di ogni coppia vedere Tabella 5.1.

Browser	*Sistema operativo*	*Numero pacchetti*
Chrome (versione 48.0.2564.109)	Windows 10	73566
Chrome (versione 68.0.3440.84)	Windows 10	67679
Firefox (versione 42.0)	Windows 10	61572
Firefox (versione 62.0)	Windows 10	123434
Edge (versione 42.17134.1.0)	Windows 10	240750
Chrome (versione 48.0.2564.109)	GNU/Linux 4.17.0	81127
Chrome (versione 68.0.3440.84)	GNU/Linux 4.17.0	52483
Firefox (versione 42.0)	GNU/Linux 4.17.0	67024
Firefox (versione 62.0)	GNU/Linux 4.17.0	50981

Tabella 5.1: Numero di pacchetti per coppia browser-sistema operativo presenti nelle catture di traffico legittimo.

[1] https://security.polito.it/

Invece le catture di traffico malevolo sono state effettuate da un terminale con sistema operativo Kali Linux, una distribuzione basata su GNU/Linux già citata nella Sezione 2.3.8, verso un altro terminale con sistema operativo Ubuntu Server[2], un'altra distribuzione basata su GNU/Linux, con installate due applicazioni web volutamente vulnerabili. Le webapp utilizzate sono DVWA[3] e Juiceshop[4]. La prima è un sito `PHP/Mysql` pensato per aiutare i professionisti di sicurezza ad eseguire test di attacchi informatici, la seconda è anch'esso un sito pensato per essere obbiettivo di attacchi realizzato in `NodeJS/Express/Angular`. Entrambi i siti sono costruiti come una sfida e per guadagnare punti bisogna scovare le vulnerabilità nascoste.

Il traffico è stato suddiviso in base al programma utilizzato per la scansione di vulnerabilità e alla tipologia di interazione avvenuta. In totale sono presenti 102776 pacchetti di rete. Il dettaglio è presente nella Tabella 5.2.

Scanner di vulnerabilità	*Webapp vittima*	*Tipologia di scansione*	*Numero pacchetti*
OpenVAS	Juiceshop	Scansione delle porte TCP	31859
OpenVAS	Juiceshop	Scansione di vulnerabilità	11264
SQLmap	DVWA	Scansione applicazione	2229
Wapiti	DVWA	Scansione applicazione	57424

Tabella 5.2: Numero di pacchetti del traffico malevolo suddiviso per programma utilizzato, obbiettivo e tipologia di scansione.

Il dataset finale è composto dall'unione dei due gruppi di catture e contiene un totale di 921392 pacchetti di rete.

Come spiegato precedentemente, l'analisi consiste in una classificazione sia binaria che multiclasse. Nel primo caso le etichette utilizzate sono *buono* e *malevolo*; nel secondo caso le classi sono associate al nome del programma di scansione utilizzato, in particolare: *buono*, *openvas*, *sqlmap* e *wapiti*.

5.2 Descrizione dei dati estratti da Netstapy

Partendo dalle catture pcap descritte nella Sezione 5.1, seguendo il flusso di esecuzione descritto nel Capitolo 4, vengono estratte le metriche che andranno a comporre il dataset per l'analisi grazie al software Netstatpy. Tramite quest'ultimo vengono prodotte 281816 osservazioni legate a traffico legittimo e 29776 osservazioni legate a traffico malevolo, per un totale di 311591 osservazioni. Ognuna di queste contiene le metriche calcolate per un flusso TCP.

Browser	*Sistema operativo*	*Numero di osservazioni*
Chrome (versione 48.0.2564.109)	Windows 10	29157
Chrome (versione 68.0.3440.84)	Windows 10	27611
Firefox (versione 42.0)	Windows 10	22344
Firefox (versione 62.0)	Windows 10	38077
Edge (versione 42.17134.1.0)	Windows 10	91948
Chrome (versione 48.0.2564.109)	GNU/Linux 4.17.0	26512
Chrome (versione 68.0.3440.84)	GNU/Linux 4.17.0	13301
Firefox (versione 42.0)	GNU/Linux 4.17.0	18314
Firefox (versione 62.0)	GNU/Linux 4.17.0	14552

Tabella 5.3: Numero di osservazioni per coppia browser-sistema operativo estratti da Netstatpy.

[2] https://ubuntu.com/

[3] Damn Vulnerable Web App, sito originale: http://www.dvwa.co.uk/

[4] https://owasp.org/www-project-juice-shop/

Ai dati appena estratti viene affiancata la classe corrispondente ottenendo così un dataset completo di etichette da utilizzare per l'apprendimento dei classificatori. Il dettaglio del traffico legittimo è riportato nella Tabella 5.3, mentre il traffico malevolo nella Tabella 5.4.

Scanner di vulnerabilità	*Webapp vittima*	*Tipologia di scansione*	*N. di osservazioni*
OpenVAS	Juiceshop	Scansione delle porte TCP	5623
OpenVAS	Juiceshop	Scansione di vulnerabiltà	11524
SQLmap	DVWA	Scansione applicazione	938
Wapiti	DVWA	Scansione applicazione	11691

Tabella 5.4: Numero di osservazioni estratte dal traffico malevolo tramite Netstatpy suddivise in base a: programma utilizzato per la scansione di vulnerabilità, obbiettivo e tipologia di scansione.

Come descritto nella Sezione 4.4, l'insieme dei dati viene utilizzato come ingresso del programma Vulnscanpy composto da più moduli che permettono di automatizzare l'analisi. La prima fase del programma è quella relativa al preprocessamento di cui l'*anonimizzazione* fa parte. Vengono quindi rimossi, osservando la Tabella A.1 del formato tstat, gli attributi numero: 1, 2, 15, 16, 38, 39. Questa operazione risulta necessaria per eliminare riferimenti a indirizzi IP, porte TCP e generalizzare meglio il problema.

Netstatpy, per questioni di compatibilità con il programma Tstat e il suo formato di output, presenta anche informazioni di alcuni campi che poi non vengono effettivamente utilizzati nell'analisi. Infatti sono presenti estrapolazioni di informazioni legate alla tipologia di IP, in particolare se un indirizzo è pubblico o privato.

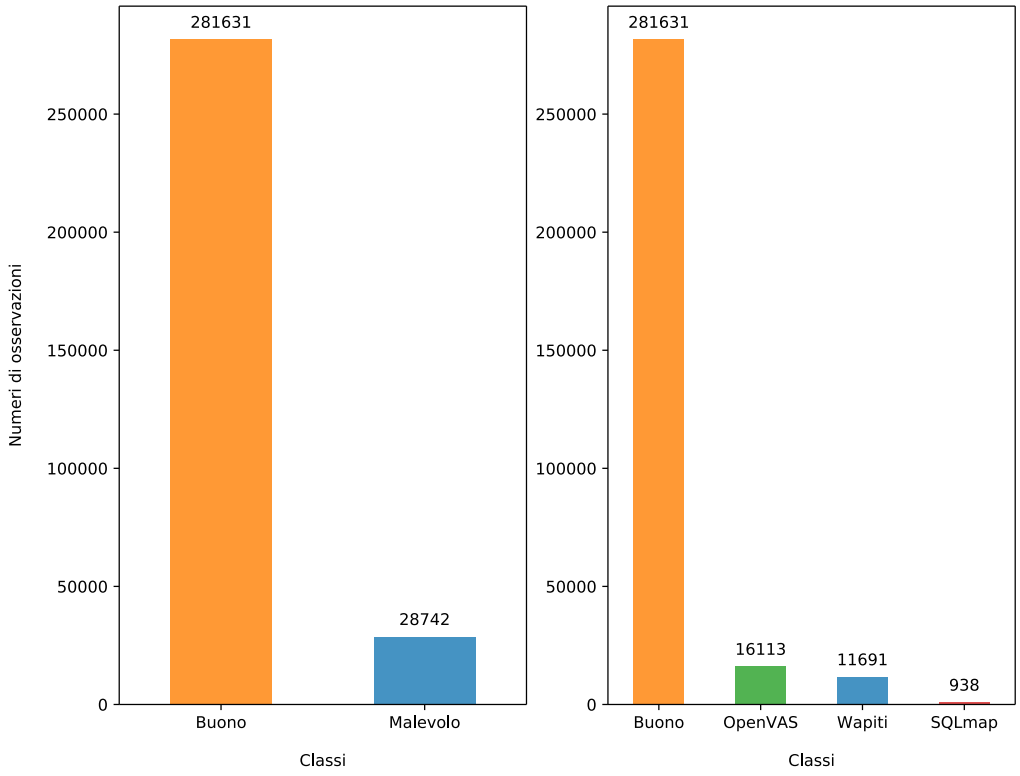


Figura 5.1: Distribuzione delle osservazioni per le diverse classi. A sinistra suddivisione binaria e a destra suddivisione multiclasse.

A queste operazioni si aggiungono anche funzioni che permettono la rimozione dei dati duplicati portando il dataset a 310373 osservazioni. Nella Figura 5.1 è possibile vedere la distribuzione delle osservazioni delle varie classi successivamente alle operazioni di rimozione sopra citate. Inoltre, dallo stesso grafico è possibile osservare come il seguente studio sia sbilanciato.

5.3 Confronto tra traffico normale e malevolo

Ponendo a confronto le statistiche dei dati estratti da traffico normale e traffico malevolo si può notare, come riportato nella Tabella 5.5, che alcuni valori hanno ordini di grandezza differenti. Tale aspetto è deducibile osservando le loro medie e le loro deviazioni standard, esse risultano sufficientemente differenti da riuscire a determinare la classe di appartenenza solamente tramite una veloce analisi visiva.

Cod. attrib	*Tipologia*	*Min*	*Max*	*Media*	*Dev. Standard*
c_ttl_min	Buono	64.00	128.00	111.48	28.00
	Malevolo	51.00	242.00	52.33	2.17
c_ttl_max	Buono	64.00	128.00	111.48	28.00
	Malevolo	51.00	242.00	52.36	3.12
s_ttl_min	Buono	29.00	247.00	87.10	63.26
	Malevolo	64.00	64.00	64.00	0.00
s_ttl_max	Buono	29.00	247.00	95.66	66.85
	Malevolo	64.00	64.00	64.00	0.00
first_time_abs	Buono	$15821.85 \cdot 10^8$	$1.5835.80 \cdot 10^8$	$15826.99 \cdot 10^8$	$6.151 \cdot 10^8$
	Malevolo	$15823.01 \cdot 10^8$	$15823.13 \cdot 10^8$	$15823.07 \cdot 10^8$	$5.566 \cdot 10^6$
last_time_abs	Buono	$15821.85 \cdot 10^8$	$15835.80 \cdot 10^8$	$15826.99 \cdot 10^8$	$6.151 \cdot 10^8$
	Malevolo	$15823.01 \cdot 10^8$	$15823.13 \cdot 10^8$	$15823.07 \cdot 10^8$	$5.566 \cdot 10^6$
rtt_c_min	Buono	0.00	2759.00	25.18	65.69
	Malevolo	0.00	55.00	0.90	0.77
rtt_c_max	Buono	0.00	321653.31	524.80	8117.16
	Malevolo	0.00	8985.00	31.03	339.58
rtt_c_avg	Buono	0.00	105405.50	99.09	984.43
	Malevolo	0.00	4047.68	7.20	92.65
rtt_c_std	Buono	0.00	139053.34	126.14	1978.91
	Malevolo	0.00	4235.10	10.76	138.09
rtt_s_min	Buono	0.00	135659.15	6.26	578.61
	Malevolo	0.00	92.98	0.90	0.97
rtt_s_max	Buono	0.00	321963.31	1891.81	17255.26
	Malevolo	0.00	5714.00	12.02	143.31
rtt_s_avg	Buono	0.00	135659.15	145.08	1991.75
	Malevolo	0.00	1764.99	2.48	34.06
rtt_s_std	Buono	0.00	142184.45	357.77	3723.71
	Malevolo	0.00	2473.80	2.83	55.85

Tabella 5.5: Comparazione di alcuni campi temporali tra traffico legittimo e traffico malevolo.

Analizzando la natura del dataset e ragionando sulle caratteristiche dell'analisi è possibile individuare delle motivazioni in grado di spiegare l'enorme differenza dei valori. Uno di questi attributi è per esempio l'attributo *rtt_c_avg*, che rappresenta il valore medio del round trip time (RTT), spiegato nella Sezione 2.1.1, calcolato sulle richieste generate dal client. Osservando il valore medio si può notare che è molto più alto nel traffico legittimo rispetto a quanto sia in quello malevolo. Tale aspetto potrebbe essere dovuto al fatto che nel primo caso le catture sono state prodotte su connessioni di flussi video streaming e navigazione web eseguita da un utente fisico, mentre nel secondo caso sono generate da software automatici. Una persona che interagisce con delle pagine web può generare delle tempistiche più dilatate rispetto a un software. Questo tipo di spiegazione è supportata anche dagli attributi *rtt_c_min* e *rtt_c_max* e le loro deviazioni standard, a dimostrazione del fatto che l'interazione umana è più variabile.

Invece, per quanto riguarda i campi relativi al time to live (TTL), spiegati nella Sezione 2.1.1, è possibile notare che la differenza tra traffico legittimo e malevolo è dovuto ad una maggiore stabilità nei valori per quest'ultimo rispetto al primo. Infatti è possibile osservare come la deviazione standard sia molto piccola. Suddetta caratteristica potrebbe essere data dal traffico legittimo composto da vari flussi di navigazione non omogenei, a causa dei collegamenti a server dislocati in posti diversi e tramite infrastrutture di rete differenti. Invece le catture del traffico malevolo sono

state effettuate da macchine più omogene e con distanze meno variabili, questo è particolarmente evidente negli attributi *s_ttl_min* e *s_ttl_max*.

I campi temporali, per via delle differenze tra il traffico legittimo e il traffico malevolo, se utilizzati come discriminante in una classificazione, permettono di identificare immediatamente la classe di appartenenza. Questa situazione permette quindi al modello di classificare perfettamente il traffico sia sul dataset di apprendimento che su quello di test. Tuttavia, essendo caratteristiche fortemente dipendenti dalla infrastruttura di rete e dai terminali, se utilizzati su di un traffico di rete reale il classificatore potrebbe non funzionare correttamente.

Per i motivi appena spiegati si è deciso di intervenire su questi attributi rimuovendone alcuni per rendere il classificatore più indipendente da questo tipo di metriche. Si è scelto di non rimuovere interamente tutte le caratteristiche citate nella Tabella 5.5, ma solo quelle che influenzavano maggiormente i risultati, permettendo così di mantenere all'interno del dataset alcuni riferimenti temporali.

In particolare, facendo sempre riferimento al formato tstat[5], vengono rimossi i campi numero: 29, 30, 45, 47, 48, 50, 51, 54, 57, 58. Gli attributi da rimuovere sono stati selezionati tramite alcune prove sperimentali, eseguendo una selezione delle feature, come descritto nella Sezione 3.1.2, e scegliendo quelle maggiormente influenti.

Applicando le rimozioni degli attributi non predittivi descritte nella Sezione 5.2 e nella Sezione corrente, rimangono 47 campi, numero sufficiente per proseguire con lo studio e caratterizzare il problema.

Nella Tabella A.3 vengono riportate le statistiche principali estratte dal dataset, escludendo quelle già rimosse e quelle già mostrate. La tabella presenta le seguenti informazioni per ogni caratteristica:

- il valore minimo nel dataset;
- il valore massimo presente nell'insieme di dati;
- la media dei valori;
- la deviazione standard.

Come spiegato nel Capitolo 4.4, nella fase di preprocessamento il dataset viene suddiviso in tre parti per le diverse fasi dell'apprendimento, in particolare vengono creati tre nuovi insiemi: training, validation e test.

Suddivisione	*Tipologia*	*Quantità*	*Totale*
Training	Buono Malevolo	197056 20205	217261
Validation	Buono Malevolo	42252 4304	46556
Test	Buono Malevolo	42323 4233	46556

Tabella 5.6: Numero di osservazioni nelle diverse suddivisioni.

Il primo insieme contiene circa il 70% del dataset originale, mentre gli altri due il 15% ognuno. Il dettaglio della suddivisione viene riportato nella Tabella 5.6.

[5]Tabella A.1

Capitolo 6

Risultati sperimentali

In questa sezione verranno esposti i risultati ottentuti dall'analisi, ponendo enfasi su:

- accuratezza e punteggi ottenuti dai classificatori;
- un confronto tra le tecniche di classificazione;
- efficacia di individuabilità degli attacchi.

6.1 Accuratezza e statistiche dei classificatori

Il dataset, dopo essere stato suddiviso in tre gruppi per l'apprendimento, la validazione e i test, come descritto nella Sezione 5.3, viene elaborato da diverse operazioni descritte in pipeline, come spiegato nella Sezione 4.5. Per ogni classificazione ne viene costruita una, due per le foreste casuali e due per le reti neurali.

6.1.1 Foresta Casuale

Per l'addestramento delle due foreste casuali, sono state composte due pipeline per cercare di massimizzare gli iperparametri esposti nella Sezione 4.5.1. Essi sono ricercati all'interno di un insieme composto da una loro combinazione, dove ciascuno iperparametro dei quali può essere anche ripetuto, ma con valori differenti. In particolare vengono ricercati:

- il criterio da utilizzare per eseguire la selezione delle caratteristiche;
- il valore k rappresentante la quantità di caratteristiche massime mantenute dalla funzione di selezione degli attributi;
- l'indice da adottare per la costruzione degli alberi;
- il valore massimo della profondità degli alberi (*max_depth*);
- il numero di stimatori utilizzati nella foresta casuale (*n_estimator*);
- la funzione di trasformazione da applicare che si adatta meglio al classificatore.

Per entrambe le foreste verranno riportati i risultati ottenuti dopo aver eseguito l'addestramento e testato il classificatore.

Classificazione binaria

La classificazione binaria ha come obbiettivo quello di distinguere le due classi del problema: traffico legittimo e traffico malevolo, come spiegato nella Sezione 5.2.

Per testare gli iperparametri che non sono numerici, ne vengono scelte delle implementazioni disponibili tramite la libreria Scikit-learn. Nello specifico le funzionalità utilizzate sono la stima dell'informazione mutua (descritta nella Sezione 3.1.2), che è disponibile tramite la funzione `mutual_info_classif`[1] e la standardizzazione del dataset, che è implementata da `StandardScaler`[2].

Nella Tabella 6.1 vengono riportati i valori migliori degli iperparametri ricercati partendo dal dataset dello studio. Questi valori sono abbastanza in linea con le aspettative, in particolare il numero di caratteristiche individuato, vista la complessità del problema per comprenderlo meglio. L'accuratezza ottenuta utilizzando questo set di opzioni sull'insieme di validazione è del 98,307%. Invece l'accuratezza finale ottenuta sull'insieme di test è del 98,847%.

Iperparametro	*Valori testati*	*Miglior valore*
Criterio di selezione delle caratteristiche	[Inf. mutua, Chi quadr.]	Informazione mutua
Numero di caratteristiche k	[10, 15, 20]	20
Indice per la costruzione degli alberi	[Gini, Entropia]	Entropia
Profondità massima degli alberi	[2, 4, 6, 8]	8
Numero di stimatori utilizzati	[200, 500, 800]	200
Funzione di trasformazione	[Standard., MinMax]	Standardizzazione

Tabella 6.1: Valori migliori per gli iperparametri per il modello di foresta casuale utilizzato come classificatore binario.

Nella Tabella 6.2 viene riportata la matrice di confusione del classificatore binario, questa permette di mostrare come il modello possa distinguere le tipologie di traffico e quanti falsi positivi e falsi negativi produce sul dataset di test.

		Valori reali	
		B	M
Valori predetti	B	41132	1191
	M	10	4223

Tabella 6.2: Matrice di confusione del classificatore, dove B è il traffico legittimo e M il traffico malevolo.

Nella Tabella 6.3 viene riportato un report sulle performance del classificatore. In particolare vengono mostrati i diversi indici presentati nella Sezione 3.1.4, mettendo in evidenza la capacità del classificatore.

Indice	*Buono*	*Malevolo*
Precisione	1.00	0.78
Richiamo	0.97	1.00
F1	0.99	0.88

Tabella 6.3: Report del classificatore binario.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Come si può osservare da queste tabelle, i valori ottenuti dal classificatore binario indicano un'ottima capacità di discriminare il traffico in legittimo e malevolo. Inoltre tale modello genera pochi falsi negativi, qualità apprezzata in questi tipi di classificazione legati alla sicurezza informatica.

Come descritto nella Sezione 3.1.4, la curva ROC e l'area sottesa dalla curva AUC permettono di mostrare in un grafico il classificatore e di renderlo poi confrontabile con un altro, la curva della foresta casuale è mostrata nella Figura 6.1.

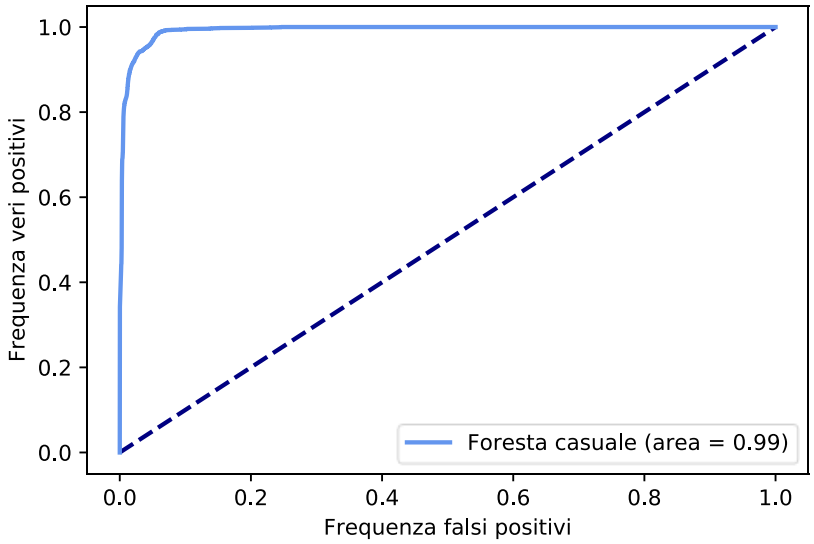


Figura 6.1: Curva ROC della foresta casuale.

Classificazione multiclasse

La classificazione multiclasse consiste nel riuscire a identificarne una classe tra le quattro differenti: buono, openvas, sqlmap e wapiti, descritte nella Sezione 5.2.

Come per il classificatore binario, nella Tabella 6.4 vengono riportati i valori migliori degli iperparametri ricercati. Utilizzandoli, l'accuratezza ottenuta sull'insieme di validazione è del 95,816%. Invece l'accuratezza finale ottenuta sull'insieme di test è del 94,288%.

Iperparametro	*Valori testati*	*Valore del parametro*
Criterio di selezione delle caratteristiche	[Inf. mutua, Chi quadr.]	Informazione mutua
Numero di caratteristiche k	[10, 15, 20]	20
Indice per la costruzione degli alberi	[Gini, Entropia]	Indice Gini
Profondità massima degli alberi	[2, 4, 6, 8]	8
Numero di stimatori utilizzati	[200, 500, 800]	500
Funzione di trasformazione	[Standard., MinMax]	Standardizzazione

Tabella 6.4: Valori migliori per gli iperparametri per il modello di foresta casuale utilizzato come classificatore multiclasse.

Nella Tabella 6.5 viene riportata la matrice di confusione del classificatore multiclasse, dove sono mostrate le diverse classi da distinguere rappresentanti i differenti strumenti di scansione di vulnerabilità.

		Valori reali			
		Buono	*OpenVAS*	*SQLmap*	*Wapiti*
Valori predetti	*Buono*	41371	636	115	201
	OpenVAS	12	2247	100	13
	SQLmap	1	17	115	0
	Wapiti	3	21	7	1697

Tabella 6.5: Matrice di confusione del classificatore multiclasse.

Nella Tabella 6.6 viene riportato un report sulle performance del classificatore mettendo in evidenza la capacità del classificatore di identificare i diversi strumenti di scansione di vulnerabilità.

Indice	*Buono*	*OpenVAS*	*SQLmap*	*Wapiti*
Precisione	1.00	0.77	0.34	0.89
Richiamo	0.98	0.95	0.86	0.98
F1	0.99	0.85	0.49	0.93

Tabella 6.6: Report del classificatore multiclasse.

Come per il classificatore binario i risultati ottenuti sono buoni, ma mostrano come la classe *sqlmap* generi un elevato numero di falsi positivi. Invece le altre classi mantengono performance più in linea con il precedente modello.

6.1.2 Reti Neurali

L'addestramento delle due reti neurali avviene con modalità simili a quelle viste per le foreste casuali, spiegate nella Sezione 4.5.2, in particolare cambiano solamente alcuni degli iperparametri dei modelli. Anche per questa tipologia di classificatori gli iperparametri sono ricercati tramite le combinazioni di ognuno di essi, in particolare vengono esplorati:

- il criterio da utilizzare per eseguire la selezione delle caratteristiche;
- il valore k rappresentante la quantità di caratteristiche massime mantenute dalla funzione di selezione degli attributi;
- il valore del coefficiente di apprendimento;
- determinare la configurazione dei livelli nascosti della rete neurale;
- la funzione non lineare all'uscita di ogni nodo;
- la funzione di trasformazione da applicare che si adatta meglio al classificatore.

Per entrambe le reti neurali verranno riportati i risultati ottenuti dopo aver eseguito l'addestramento e testato il classificatore.

Classificazione binaria

Come descritto per la foresta casuale, la classificazione binaria consiste nel distinguere traffico legittimo da traffico malevolo, inoltre le funzioni che permettono sia di stimare l'informazione mutua che di standardizzare sono le stesse e sono fornite dalla libreria Scikit-learn. Invece la funzione non lineare ReLU è fornita dalla libreria Pytorch tramite la funzione `nn.ReLU`[3].

[3] https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

La configurazione dei livelli nascosti è descritta tramite una lista, il numero di elementi al suo interno rappresenta il numero dei livelli nascosti, invece il valore di ogni elemento rappresenta il numero di nodi presenti per quel livello.

Per esempio la configurazione [5, 10, 5] rappresenta un'opzione con tre livelli nascosti, dove il primo e l'ultimo hanno 5 nodi e quello centrale 10. Il primo elemento della configurazione si collega all'uscita del livello di ingresso, mentre l'ultimo si collega all'ingresso del livello di uscita della rete neurale.

Nella Tabella 6.7 vengono riportati i valori migliori dei parametri ricercati partendo dal dataset dello studio. Utilizzando questi iperparametri l'accuratezza ottenuta sull'insieme di validazione è del 96,095%, mentre l'accuratezza finale ottenuta sull'insieme di test è del 95,341%.

Iperparametro	*Valori testati*	*Valore del parametro*
Criterio di selezione delle caratteristiche	[Inf. mutua, Chi quadr.]	Informazione mutua
Numero di caratteristiche k	[10, 15, 20]	20
Coefficiente di apprendimento	[0.05, 0.1, 0.2]	0.2
Configurazione dei livelli nascosti	[[10, 5], [20, 10], [20, 10, 5], [20, 30, 10], [15, 30, 15, 5]]	[20, 10, 5]
Funzione non lineare	[Sigmoide, ReLU]	ReLU
Funzione di trasformazione	[Standard., MinMax]	Standardizzazione

Tabella 6.7: Valori migliori per gli iperparametri per il modello di rete neurale utilizzato come classificatore binario.

Nella Tabella 6.8 viene riportata la matrice di confusione del classificatore binario. Essa mostra che, come per il classificatore binario della foresta casuale, questo modello produce una quantità di falsi negativi relativamente bassa, qualità apprezzata in questo studio.

		Valori reali	
		B	M
Valori predetti	B	39889	2434
	M	151	4082

Tabella 6.8: Matrice di confusione del classificatore binario, dove B è il traffico legittimo e M il traffico malevolo.

Invece nella Tabella 6.9 viene riportato un report sulle performance del classificatore mettendo in evidenza la sua capacità di identificare le due tipologie di traffico. Come già anticipato, questo modello dimostra delle buone performance perdendo un po' di precisione nel riconoscere il traffico malevolo rispetto alla foresta casuale.

Indice	Buono	Malevolo
Precisione	1.00	0.63
Richiamo	0.94	0.96
F1	0.97	0.76

Tabella 6.9: Report del classificatore binario.

Come per il modello di foresta casuale, nella Figura 6.2 viene mostrata la curva ROC e l'area sottesa dalla curva AUC del classificatore binario basato su una rete neurale.

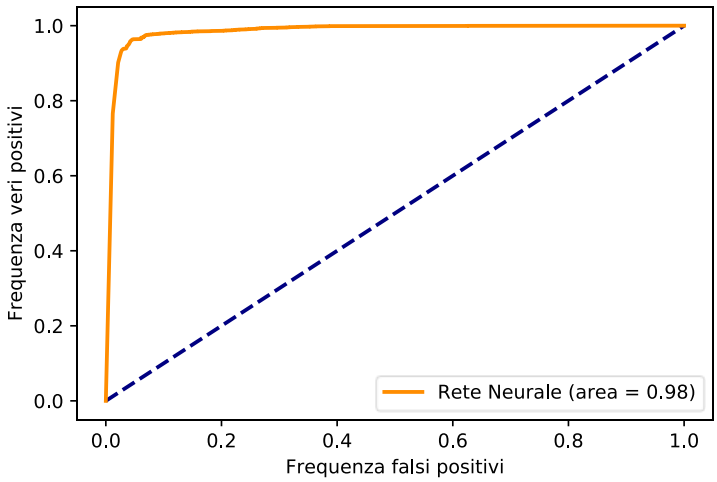


Figura 6.2: Curva ROC della rete neurale.

Classificazione multiclasse

Come per il modello della foresta casuale, le classi che questo classificatore deve riuscire a riconoscere sono: buono, openvas, sqlmap e wapiti. Diversamente dagli altri modelli, in questo classificatore multiclasse, basato su una rete neurale, la funzione di trasformazione e il criterio di selezione delle feature sono differenti.

Come si può vedere nella Tabella 6.10, dove sono riportati i migliori valori per gli iperparametri del classificatore, il criterio di selezione utilizzato è Chi quadrato, spiegato in 3.1.4, che nella libreria Scikit-learn è implementata tramite il metodo `chi2`[4].

Invece la funzione di trasformazione adottata è quella di *scalamento delle feature*, descritta in 3.1.1, la quale, nella libreria Scikit-learn, è implementata tramite il metodo `MinMaxScaler`[5]. Utilizzando i parametri appena descritti l'accuratezza ottenuta sull'insieme di validazione è del 81,902%. Invece l'accuratezza finale ottenuta sull'insieme di test è del 83,029%.

Iperparametro	*Valori testati*	*Valore del parametro*
Criterio di selezione delle caratteristiche	[Inf. mutua, Chi quadr.]	Chi quadrato
Numero di caratteristiche k	[10, 15, 20]	20
Coefficiente di apprendimento	[0.0007, 0.05, 0.1, 0.2]	0.0007
Configurazione dei livelli nascosti	[[512, 128, 64], [512, 256, 128, 64]]	[512, 256, 128, 64]
Funzione non lineare	[Sigmoide, ReLU]	ReLU
Funzione di trasformazione	[Standard., MinMax]	MinMax

Tabella 6.10: Valori migliori per gli iperparametri per il modello di rete neurale utilizzato come classificatore multiclasse.

Nella Tabella 6.11 viene riportata la matrice di confusione del classificatore multiclasse, dove sono riportate le diverse classi da distinguere rappresentanti i diversi strumenti di scansione di vulnerabilità.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

		Valori reali			
		Buono	*OpenVAS*	*SQLmap*	*Wapiti*
Valori predetti	*Buono*	38300	2524	1419	80
	OpenVAS	118	1613	641	0
	SQLmap	0	24	109	0
	Wapiti	5	126	13	1584

Tabella 6.11: Matrice di confusione del classificatore multiclasse.

Nella Tabella 6.12 viene riportato un report sulle performance del classificatore mettendo in evidenza la capacità del classificatore di identificare i diversi strumenti di scansione di vulnerabilità.

Indice	*Buono*	*OpenVAS*	*SQLmap*	*Wapiti*
Precisione	1.00	0.38	0.05	0.95
Richiamo	0.90	0.68	0.82	0.92
F1	0.95	0.48	0.09	0.93

Tabella 6.12: Report del classificatore multiclasse.

In generale, come è possibile osservare dalle tabelle, questo classificatore ha delle performance per ogni classe abbastanza diverse. Nello specifico riesce a riconoscere molto bene il traffico legittimo e la classe *wapiti*, confonde spesso la classe *openvas* con il traffico legittimo e si riscontra una marcata difficoltà ad individuare correttamente la classe *sqlmap*. Quest'ultima caratteristica è condivisa con il classificatore multiclasse della foresta casuale, anche se in questo caso l'impatto è più evidente.

6.2 Confronto tra tecniche di classificazioni

Per riuscire a confrontare i modelli tra di loro, vengono affiancate delle metriche comuni. In particolare, per i due classificatori, nella Tabella 6.13 sono presenti i valori dei punteggi medi della precisione, del richiamo e dell'indice F1.

Modello	*Precisione media*	*Richiamo medio*	*F1 medio*
Foresta casuale	0.890	0.985	0.935
Rete neurale	0.815	0.950	0.865

Tabella 6.13: Punteggi medi a confronto tra i due classificatori binari.

Come si può osservare da questi indici, il modello di foresta casuale ha un valore più alto, dimostrano entrambi di essere in grado di distinguere molto bene il traffico legittimo da quello malevolo. Inoltre, come già detto nella Sezione 6.1, entrambi hanno una buona capacità di generare un numero basso di falsi negativi, ovvero segnalare come malevolo traffico che non lo è.

Come ulteriore metrica per paragonare i due classificatori è possibile sovrapporne le due curve ROC. Come si può vedere nella Figura 6.3, la foresta casuale ha un'area leggermente maggiore rispetto alla rete neurale, anche se lo scarto è minimo ed è quindi possibile considerarli equivalenti.

Come per i classificatori binari, i punteggi medi per i modelli utilizzati per la classificazione multiclasse sono riportati nella Tabella 6.14. Anche in questo caso il modello con punteggi maggiori è la foresta casuale.

Da queste perfomance è possibile osservare che entrambi i modelli hanno buone capacità di riconoscere le diverse classi, ma che per la rete neurale l'impatto dovuto a *sqlmap* ha abbassato molto le sue statistiche generali.

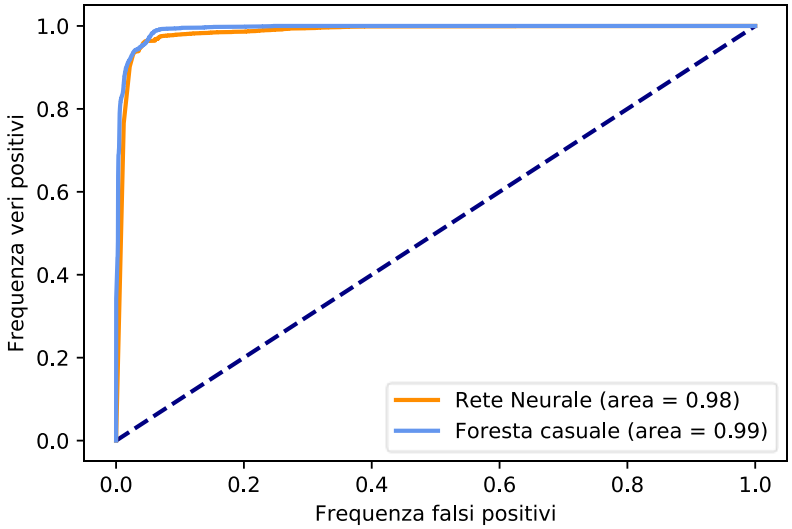


Figura 6.3: Confronto tra le curve ROC dei due classificatori binari.

Modello	*Precisione media*	*Richiamo medio*	*F1 medio*
Foresta casuale	0.750	0.943	0.815
Rete neurale	0.595	0.83	0.613

Tabella 6.14: Punteggi medi a confronto tra i due classificatori multiclasse.

6.3 Discussione della individuabilità degli attacchi

Come mostrato nella Sezione 6.1 e nella Sezione 6.2, in generale i diversi classificatori hanno ottenuto valori di accuratezza, precisione, richiamo e F1 molto buoni. Essi infatti riescono a classificare correttamente il traffico in una buona percentuale dei casi. La classificazione binaria risulta essere più precisa rispetto a quella multiclasse e con una frequenza di falsi negativi bassa.

Anche i classificatori multiclasse hanno delle buone accuratezze, soprattutto quello basato su una foresta causale, il quale ha ottenuto migliori risultati rispetto a quello basato sulla rete neurale utilizzando i rispettivi iperparametri ottimali individuati.

Tra gli strumenti di scansione di vulnerabilità l'unico che viene classificato con più difficoltà è *SQLmap*, soprattutto il classificatore basato sulla rete neurale. Infatti, come è possibile vedere dai risultati ottenuti ed esposti nella Sezione 6.1, nella rete neurale i punteggi ottenuti su questa classe hanno abbassato il livello generale delle performance del classificatore. Una possibile causa principale di questa riduzione dei punteggi è dovuta alla scarsa quantità di dati presenti relativi ad essa, come descritto nella Sezione 5.2, che hanno portato ad un minore apprendimento della classe. Inoltre, come si vede dalla matrice di confusione, SQLmap viene spesso classificata erroneamente come del traffico legittimo.

Al fine di ottenere uno studio efficace, non è solamente importante l'ottenimento di una classificazione corretta del traffico minimizzando i falsi negativi, ma anche l'identificazione del traffico malevolo con il minor numero di segmenti, puntando a mantenere una precisione che non si attenui con l'aumentare dei pacchetti analizzati.

Per osservare questa caratteristica si è incrementato progressivamente il numero di pacchetti analizzati dal classificatore e per ogni passaggio si è calcolato il valore AUC ottenuto. Come si può osservare dalla Figura 6.4, le performance del classificatore basato sulla rete neurale iniziano con un valore elevato, osservando i valori ottenuti si può notare che il grafico arriva a un minimo del 91.18%, ma successivamente si stabilizza attorno a un valore medio di 95.54%.

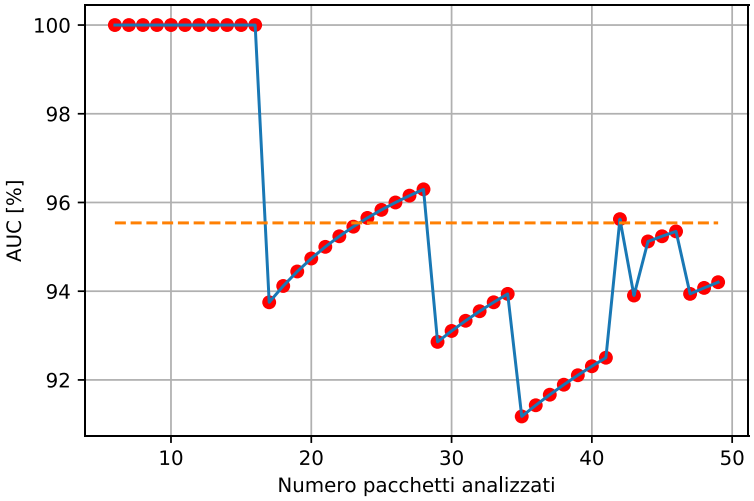


Figura 6.4: Andamento del AUC all'aumentare dei pacchetti di rete analizzati per il modello della rete neurale.

Analogamente i valori progressivi per il classificatore basato sulla foresta casuale sono riportati nella Figura 6.5. Come il modello precedente, questo comincia con un valore di AUC molto elevato scendendo fino a un minimo di 93.75%, all'aumentare dei pacchetti il valore si aggira intorno al valore medio di 96.88%.

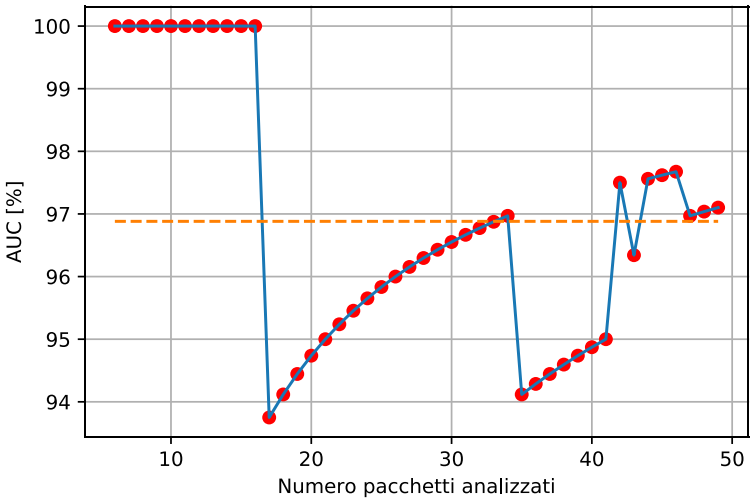


Figura 6.5: Andamento del AUC all'aumentare dei pacchetti di rete analizzati per il modello della foresta casuale.

Questo tipo di performance permette di utilizzare il classificatore come sensore in applicazioni reali in quanto è in grado di identificare il traffico malevolo entro un numero di pacchetti ragionevole.

Capitolo 7

Lavori collegati

Nei Capitoli precedenti è stata presentata e illustrata un'architettura che ha lo scopo di individuare strumenti di scansione di vulnerabilità su di un traffico di rete. Come approfondito nella Sezione 4.2, volendo generalizzare maggiormente il problema, si può osservare che si tratta della capacità di individuare una traccia utilizzando un'analisi intelligente di un flusso di rete.

Questa capacità è ricercata anche in altri campi, affetti a loro volta da problematiche legate alla maggiore adozione di protocolli cifrati. In questo Capitolo vengono quindi esposti alcuni ambiti di studio che sono legati a questa analisi, in particolare:

- gli strumenti di individuazione e prevenzione delle intrusioni;
- l'unione dell'apprendimento automatico e della sicurezza informatica.
- l'unione dell'apprendimento automatico e dell'analisi del traffico;

Nell'ambito specifico degli strumenti di vulnerabilità, non sono stati individuati degli studi che sfruttino questo tipo di analisi intelligente. Infatti nei lavori presentati in questo Capitolo lo scopo sarà quello di individuare direttamente una tipologia di attacco o un particolare protocollo piuttosto che uno strumento di scansione.

7.1 Sistemi per l'individuazione e prevenzione delle intrusioni (IDS/IPS)

Il mondo della sicurezza informatica ricopre diversi aspetti e permette di intervenire a differenti livelli dei vari sistemi, ragione per cui gli studi in questo ambito sono molti, diversi dei quali rivolti allo studio di applicazioni di apprendimento automatico. Ne sono un esempio la creazione di strumenti per il rilevamento delle anomalie e delle intrusioni, anche noti come *intrusion detection system* (IDS), tali software sono in grado di riconoscere comportamenti non ordinari e di segnalarli. L'unione di questi sistemi di rilevamento con le architetture di rete SDN[1], in grado di prendere decisioni localmente per l'intera rete, permettono quindi di reagire a tali fenomeni eseguendo direttamente delle prime azioni difensive.

Gli IDS/IPS per riconoscere possibili violazioni e anomalie di sicurezza analizzano i pacchetti di rete e per questo motivo possono essere associati agli strumenti di analisi, ma a differenza della soluzione proposta in questa tesi, gli IDS non sono solitamente in grado di rilevare scansioni di vulnerabilità. Come descritto da Jackson et al. in *"Intrusion detection system (IDS) product*

[1]Software-defined networking, architettura di rete orientata al cloud computing che punta a centralizzare il monitoraggio della rete e il piano di controllo. Questa caratteristica permette di prendere decisioni per la rete basandosi sull'intero stato della stessa grazie al monitoraggio centralizzato.

survey"[2], gli scanner sono considerati strumenti per gli amministratori di sistema per eseguire test e analisi interne, invece gli IDS devono rilevare solamente tentativi espliciti. Inoltre gli strumenti per la scansione sono spesso integrati all'interno degli IDS stessi rendendone il riconoscimento ancora più complesso.

Alcuni dei software che hanno funzione di IDS/IPS come *Suricata*[3], *Snort*[4] e *Zeek*[5], riescono invece a rilevare gli scanner, ma hanno alcune limitazioni, per esempio nel caso di scansioni molto lunghe non sono in grado di riconoscerne la natura, oppure se presenti più IP che eseguono le scansioni in contemporanea non riescono a riconoscere il fenomeno collettivamente. Le problematiche appena elencate vengono superate con una soluzione come quella proposta in questa tesi in quanto l'analisi avviene per flusso TCP a prescindere dalla sua lunghezza senza contare che l'analisi non si basa su metriche come l'indirizzo IP, ma esclusivamente su metriche estratte dalle intestazioni dei pacchetti.

7.2 Apprendimento automatico e sicurezza informatica

Come già citato nell'introduzione di questo Capitolo, l'aumento di utilizzo di protocolli cifrati rende necessario adeguare anche i software di sicurezza informatica che effettuano analisi di traffico di rete. Anche in questo ambito l'impiego di tecniche di apprendimento automatico può essere una soluzione in grado di superare queste limitazioni. Tale tipo di modelli è studiato, oltre che per utilizzi su traffico di rete, anche per esaminare e riconoscere altre tipologie di attacchi informatici effettuati direttamente sui terminali da proteggere.

Un esempio di questo tipo di sistemi è quello studiato in [35], lavoro che cerca di trovare un modo per riconoscere intrusioni su protocolli cifrati come *SSH*[6]. L'idea alla base dello studio è riuscire a riconoscere sequenze di comandi attribuendo ad ognuno di essi un punteggio che, superata una soglia, determina l'appartenenza ad un possibile attacco informatico. Una caratteristiche comune a questa tesi è la decisione di non utilizzare DPI e quindi di non effettuare decifrazione dei pacchetti di rete, lavorando dunque solo sulle statistiche del traffico. In particolare i tipi di attacco che cercano di riconoscere sono: *remote-to-local* (R2L) e *user-to-root* (U2R).

Il primo è un attacco che ha come obbiettivo quello di introdursi all'interno di un sistema vittima da una sorgente remota sfruttando vulnerabilità presenti sul sistema stesso. Questo tipo di attacco è possibile quando i software accessibili da remoto presentano bachi di sicurezza o non aggiornati oppure configurati malamente, quindi l'attaccante riesce ad ottenere un accesso locale nel sistema così da poter superare eventuali filtri di rete. Il secondo è idealmente il passaggio successivo al primo, in questo l'attaccante, che ha già ottenuto un accesso, cerca di individuare la possibilità di acquisire i privilegi di amministrazione (*root*) e poter eseguire quindi operazioni privilegiate. Anche in questo caso è possibile, sfruttando le eventuali vulnerabilità dei software che possono accedere a zone riservate, superare le limitazioni di permessi che un normale utente possiede, potendo successivamente agire liberamente sulla macchina.

Il classificatore utilizzato è basato sul *clustering*, ovvero una tecnica di classificazione fondata sul raggruppamento delle osservazioni simili, come per esempio il k-NN citato nella Sezione 7.3, modello grazie al quale è possibile individuare la vicinanza che hanno due osservazioni, in questo caso comandi trasportati tramite SSH.

Lo scenario proposto dallo studio precedente è piuttosto limitato operando su traffico SSH. Uno studio che si avvicina di più al lavoro svolto in questa tesi è [36], nel quale si cerca di costruire

[2] http://www.cs.unibo.it/~montreso/master/materiale/ids/ids-product-survey.pdf

[3] https://suricata-ids.org/

[4] https://www.snort.org/

[5] https://zeek.org/

[6] Secure Shell, è un protocollo di comunicazione in grado di stabilire una sessione remota cifrata tramite interfaccia a riga di comando.

un metodo per il rilevamento dei *web crawler*, che rientrano nella categoria dei *bot*. Trattasi di software in grado di interagire autonomamente con il web per raccogliere informazioni.

I modelli utilizzati vengono allenati analizzando traffico di rete creato sia tramite interazione umana che tramite software. Uno degli obbiettivi dello studio, oltre a quello di riconoscere i web crawler, è riuscire a farlo in tempo reale. Per cercare di ottenere questi risultati vengono utilizzati come modelli di apprendimento automatico gli *alberi decisionali*, spiegati nella Sezione 3.2.1, per classificare il traffico in richieste originate da umani e da crawler. Lo studio dimostra che grazie a questa architettura è possibile distinguere questo tipo di interazione potendolo fare già con pochi pacchetti, più precisamente 15 ottenendo un punteggio F1 del 91.2% con un rateo di falsi positivi del 4%. Aumentando invece il numero di pacchetti analizzati fino a 50, diminuisce l'indice arrivando al 62%, ma con nessun falso positivo.

Una tipologia che si avvicina ancora di più a quella proposta nel Capitolo 4, è quella presentata in [37]. Essa sfrutta un flusso di esecuzione simile a quello proposto in questa tesi, composto prima dall'estrazione di metriche dal traffico di rete, senza fare deep packet inspection, da impiegare poi nella fase successiva nella quale viene utilizzato per allenare i modelli di apprendimento automatico. Lo studio citato ha come obbiettivo quello di riconoscere attacchi informatici di tipo *DDoS*, già citati in precedenza, tipologia di attacco resa complessa da individuare a causa della difficoltà nel distinguerlo da traffico legittimo.

Uno degli altri fattori che rendono difficile il riconoscimento di questi attacchi è l'evoluzione temporale dello stesso, poiché essendo composto da molteplici flussi e da risorse differenti è difficile predire quali saranno l'entità e la durata dell'attacco. Per cercare di attenuare questi problemi si è deciso di utilizzare i seguenti modelli di apprendimento automatico: modelli di *reti neurali convoluzionali*, modelli con memoria detti *long short-terme memory* (LSTM) e modelli di *reti neurali ricorrenti*, in grado di ottenere buoni risultati nonostante le limitazioni sopra citate. Questi classificatori sono molto utili nell'analisi di problemi temporali, infatti includono tecniche in grado di influenzare l'apprendimento basandosi non solo sull'osservazione corrente, ma anche su quella precedente.

Nonostante in questo studio si faccia analisi di rete tramite *shallow inspection*, vengono estratte metriche anche da protocolli non di trasporto, come ICMP, e presi in considerazione anche protocolli applicativi come HTTP. Nello studio, nonostante l'utilizzo di un dataset contenente un attacco DDoS, è stato possibile ridurre l'errore di classificazione rispetto ad altri modelli più classici. In particolare un modello di foresta casuale ha ottenuto un punteggio F1 del 93.893% con un rateo di errore pari al 6.373% contro il 97.601% ottenuto dal modello ricorrente con un rateo di errore del 2.394%.

Un esempio di apprendimento automatico applicato alla sicurezza e agli IDS/IPS è quello presentato in [38], questo lavoro utilizza il deep learning per creare un sistema in grado di rilevare potenziali attacchi e intrusioni grazie all'identificazione di anomalie nel traffico di rete. Per raggiungere tale obbiettivo utilizza delle reti neurali convoluzionali, delle reti neurali ricorrenti e degli autoencoders, una tipologia di modello in grado di trasformare i dati in ingresso comprimendoli e poi tramite una decompressione ricondurli all'output. Tutti questi modelli sono evoluzioni avanzate delle tecniche spiegate nella Sezione 3.2.2.

Il dataset utilizzato contiene diverse tipologie di attacchi informatici, come DoS, U2R, R2L e Probe, una tipologia di attacco che permette di ispezionare un sistema e rilevarne eventuali impronte digitali permettendo di identificare i servizi offerti. All'interno dello studio vengono paragonati i diversi modelli di deep learning con modelli più tradizionali allenati con lo stesso dataset, evidenziando come i primi aumentino l'accuratezza generale nonostante un notevole aumento del tempo di apprendimento per i modelli. In generale tutti i modelli hanno ottenuto accuratezze medie tra il 95% e il 98%.

Un ultimo esempio legato al mondo della sicurezza e al machine learning è quello descritto in [39], questo studio utilizza modelli di rete neurali convoluzionali per il riconoscimento di malware su di un traffico di rete. Essa utilizza una tecnica simile a quella esplorata in questo lavoro di tesi, in quanto analizza i pacchetti non ancora lavorati, ne estrapola le caratteristiche principali e le impiega per allenare i modelli. A differenza però di questa tesi vengono utilizzati tutti i diversi livelli del modello TCP/IP, spiegato nella Sezione 2.1.

I dati sono analizzati disponendo le diverse caratteristiche in una griglia, questo perchè le reti convoluzionali usano questo tipo di dati in ingresso, il modello viene allenato per riconoscere dei malware e distinguerli da traffico legittimo. I risultati dei diversi modelli allenati, in media, si aggira intorno al 99.4%, inoltre questo lavoro compara i propri risultati con altri studi soffermandosi sulla sua possibilità di individuare virus in molti più scenari rispetto agli altri.

7.3 Apprendimento automatico e analisi del traffico

Gli analizzatori del traffico, soprattutto in ambienti aziendali o negli internet service provider (ISP), costituiscono componenti fondamentali per poter classificare i diversi protocolli e creare politiche di *qualità del servizio*[7] (QoS). In origine la classificazione veniva eseguita tramite il riconoscimento delle porte TCP utilizzate da un particolare servizio, vedere Sezione 2.1.1. Tale pratica è stata poi velocemente sostituita da strumenti in grado di analizzate il payload del pacchetto di rete e riconoscerne i servizi, operazione nota come deep packet inspection (DPI), già citata in precedenza. Trattasi, come spiegato nel Capitolo 4, di una tecnica poco efficiente in termini computazionali e, se affiancata al crescente utilizzo di protocolli cifrati, più complessa da realizzare rendendola quindi una soluzione non praticabile.

Una tecnica alternativa è quella basata sull'elaborazione delle statistiche del flusso di comunicazione, la quale analizza le intestazioni dei pacchetti di rete e ne estrapola delle metriche da utilizzare per l'analisi. I modelli di apprendimento automatico, fondandosi su basi probabilistiche e statistiche, vedere Capitolo 3, risultano particolarmente adatti per questo tipo di problemi, inoltre possono aumentare in modo significativo l'accuratezza della classificazione.

Questa tipologia di tecnica è la stessa che viene utilizzata in questa tesi e nello studio condotto da Lopez-Martin et al. in [40], il quale ha come obbiettivo la creazione di un modello in grado di classificare i servizi per il traffico *internet of things*[8] (IoT). Esso è caratterizzato da flussi eterogenei in quanto prodotti da terminali anche molto diversi tra di loro, per questo motivo strumenti di analisi del traffico classici potrebbero non essere efficaci. Nel lavoro appena citato si ricorre a modelli che combinano tecniche di deep learning come *reti neurali convoluzionali*[9] (CNN) e *reti neurali ricorsive*[10] (RNN). Questo tipo di modelli permette un'analisi orientata all'evoluzione temporale della connessione, infatti nello studio vengono elaborati i primi 20 pacchetti di ogni flusso, operazione utile per distinguere meglio servizi riconoscibili dalla loro evoluzione, come per esempio i flussi streaming. Nello studio l'utilizzo di questi modelli ha portato dei buoni risultati nelle varie classificazioni, ottenendo punteggi F1 che variano tra il 75% e il 95.53% nei diversi scenari.

Un ulteriore esempio di analisi tramite apprendimento automatico è quello esposto in [41], questo lavoro descrive come un classificatore basato su un modello di Markov nascosto, una particolare catena di Markov solitamente in grado di riconoscere schemi temporali, possa identificare particolari pagine web e dedurne il contenuto nonostante queste siano cifrate.

L'approccio utilizzato è stato quello di analizzare le intestazioni del traffico in entrata e in uscita dal sito. Grazie a questa tecnica sarebbe possibile risalire a dati sensibili degli utenti che navigano queste pagine e, per sottolineare l'entità della possibile minaccia, si fa presente che all'interno del dataset per l'analisi vi sono siti di servizi per la sanità, la finanza e legali, dimostrando come sia possibile rubare le informazioni che un individuo scambia con essi. Nello

[7] La configurazione personalizzata di parametri di rete permette di migliorare le prestazioni per un determinato servizio.

[8] L'internet of things è un particolare traffico di rete che si distingue da un flusso internet classico in quanto è generato da terminali sempre connessi e che hanno una particolare posizione geografica.

[9] Fanno parte della famiglia delle reti neurali, spiegate nella Sezione 3.2.2, e sono caratterizzate dal tipo di connessione dei livelli che, al posto di essere totalmente connessi, sono collegati tramite convoluzioni.

[10] Appartengono anch'esse alla famiglia delle reti neurali e sono caratterizzate dalla possibilità di mantenere informazioni sullo stato precedente durante l'analisi di ogni osservazione. Questa caratteristica permette di eseguire studi su fenomeni che evolvono nel tempo.

stesso studio viene anche descritta una possibile mitigazione per limitare la possibilità che questo tipo di identificazione venga utilizzato per rubare dati sensibili, soluzione che sfrutta un metodo che offusca il traffico. Le accuratezze ottenute sui diversi tipi di traffico variano tra il 71% e il 90%.

Un altro ambito nel quale i dispositivi presenti sono molto eterogenei e nel quale si sta anche verificando una veloce migrazione verso protocolli cifrati è quello *mobile*. L'utilizzo sempre più frequente degli smartphone ha reso questo settore il principale vettore di traffico internet, in particolare hanno avuto un ruolo fondamentale le applicazioni di messaggistica istantanea e i social. In generale queste app, per garantire una maggiore sicurezza, hanno implementato protocolli crittografici end-to-end, riducendo quindi possibili ottimizzazioni a livello di rete a causa di un più complesso riconoscimento dei servizi utilizzati.

Le app di messaggistica sono solitamente composte da più servizi in-app che risultano a tutti gli effetti tipologie di connessioni differenti, in particolare sovente permettono di inviare messaggi, foto e audio. Lo studio [42] ha come obbiettivo quello di riuscire a riconoscere questi servizi in-app con lo scopo finale di poter aiutare gli operatori di rete a creare politiche di QoS in base a queste tipologie e permettere un'esperienza utente migliore. Per farlo vengono utilizzati dei modelli di *foresta casuale*, vedere Sezione 3.2.1, come classificatori per distinguere i diversi servizi. Per l'allenamento è stato impiegato un dataset formato da traffico generato da famose applicazioni di messagistica, quali Whatsapp[11] e WeChat[12], raggiungendo performance che, in tale ambito, paiono risultare promettenti. In particolare per la prima app si è ottenuta accuratezza media del 96.09%, simile a quella della seconda che ha ottenuto un punteggio medio del 97.40%.

Nonostante l'utilizzo di protocolli cifrati, in alcune situazioni è comunque possibile ricavare informazioni relative alla navigazione tramite l'identificazione di *impronte digitali* delle applicazioni visitate. Lo studio [43], partendo da questi presupposti, cerca di creare tramite apprendimento automatico un modello in grado di individuare la pagina web con cui si ha avuto interazione. Le pagine web, essendo trasmesse tramite lo stesso protocollo applicativo, risultano avere statistiche simili sui singoli pacchetti. Per questo motivo, tecniche di analisi che si basano su di esse, come quelle presentate in questa Sezione, potrebbero non distinguere due pagine web differenti.

Lo studio invece utilizza un approccio alternativo analizzando la quantità di byte totale scambiati nell'intera connessione, questo è possibile perchè l'interazione con ogni pagina web nell'intero flusso è differente. Nello specifico allena un modello *k-nearest neighbors*[13] (k-NN) utilizzando come dataset la somma cumulativa dei payload dei primi 100 pacchetti di una connessione. Grazie a questa tecnica, lo studio, riesce a dimostrare che le pagine web possono essere facilmente riconoscibili tramite le impronte digitali. In particolare è evidenziato il punteggio F1 ottenuto del 91.8% che indica non solo un buon grado di accuratezza, ma anche un basso tasso di falsi positivi.

[11] https://www.whatsapp.com

[12] https://www.wechat.com/

[13] Una tecnica di apprendimento automatico supervisionata che permette di classificare tramite il raggruppamento di elementi simili.

Capitolo 8

Conclusioni

Questa tesi ha presentato un sistema di identificazione per scanner di vulnerabilità su di un traffico di rete. Le analisi di vulnerabilità prodotte dagli scanner costituiscono un problema in quanto il loro utilizzo permette di studiare un sistema in automatico e di individuarne vulnerabilità che possono essere utilizzate per violarlo. Strumenti di identificazioni che sfruttano analisi di tipo deep packet inspection (DPI) permetterebbero di riconoscere questi scanner, ma non possono essere utilizzati per via della graduale migrazione verso protocolli cifrati che rendono praticamente impossibile il loro utilizzo.

Per questo motivo l'obbiettivo di questo lavoro di tesi è stato quello di dimostrare la possibilità di poter individuare l'utilizzo di scanner di vulnerabilità su di un traffico di rete senza ricorrere ad analisi di tipo DPI, ovvero utilizzando tecniche di shallow packet inspection (SPI). Tali soluzioni analizzano l'intestazione dei pacchetti di rete senza però tentare di interpretarne il contenuto del payload. Sono quindi stati spiegati i concetti base delle reti, degli strumenti di analisi e degli scanner di rete.

Per raggiungere tale obbiettivo sono stati allenati dei classificatori basati su modelli di apprendimento automatico, nello specifico foreste casuali e reti neurali, in grado di analizzare il traffico tramite SPI. Sono quindi stati descritti i concetti base del funzionamento dei modelli di apprendimento automatici. La scelta è ricaduta su questi modelli per riuscire a evidenziare le differenze tra una tecnica più classica (foresta causale) rispetto a una più moderna che rappresenta la base del deep learning (rete neurale).

Successivamente è stata esposta una possibile soluzione in grado di raggiungere l'obbiettivo sfruttando due software scritti appositamente per tale soluzione e i risultati ottenuti dai modelli. Questi, oltre a riuscire a distinguere traffico legittimo da traffico malevolo, sono in grado di determinare quale scanner viene utilizzato all'interno di un insieme pre-determinato.

Statistiche (%)	*Foresta casuale*	*Rete neurale*
Precisione	89.0	81.5
Richiamo	98.5	95.0
F1	93.5	86.5

Tabella 8.1: Migliori statistiche ottenute dai due classificatori binari.

La sperimentazione condotta ha riportato buoni risultati come esposto nel Capitolo 6, in particolare si è osservato come il modello di foresta casuale abbia ottenuto generalmente risultati migliori rispetto alla rete neurale sia nella classificazione binaria che in quella multiclasse. I risultati ottenuti sono riassunti nella Tabella 8.1 per la prima e nella Tabella 8.2 per la seconda.

Statistiche (%)	*Foresta casuale*	*Rete neurale*
Precisione	75.0	59.5
Richiamo	94.3	83.0
F1	81.5	61.3

Tabella 8.2: Migliori statistiche ottenute dai due classificatori multiclasse.

Il modello di foresta casuale binario ha ottenuto risultati sufficientemente buoni da poter prevedere un eventuale sperimentazione in un ambiente reale con banda ridotta o da essere implementato come una parte di un sistema di individuazione e prevenzione delle intrusioni (IDS/IPS), per il momento non ancora in tempo reale. La rete neurale ha anch'essa buoni risultati, ma vista la possibilità di cambiare la sua architettura aggiungendo nuovi livelli nascosti e cambiare il numero dei nodi per ognuno di essi, potrebbe essere necessario sperimentare ulteriori configurazioni per aumentare ulteriormente la precisione ottenuta.

Il modello di rete neurale multiclasse ha dimostrato performance relativamente basse rispetto a quelle ottenute dalla foresta casuale, ciò è stato dovuto alla difficoltà di individuare la classe *sqlmap* che aveva pochi dati a disposizione. La problematica più evidente è stata la grande quantità di falsi positivi ottenuti per questa classe che ha peggiorato sensibilmente la precisione del modello e conseguentemente il punteggio F1. Il classificatore multiclasse della foresta casuale, nonostante i buoni risultati, potrebbe avere necessità di ulteriori accorgimenti, in quanto presenta, seppure in entità minore, le stesse problematiche della rete neurale.

Per superare la difficoltà appena descritta si potrebbero utilizzare modelli più avanzati di apprendimento automatico in grado di imparare meglio le particolarità delle diverse classi, in quanto la presenza dei pochi dati non ne permette una corretta assimilazione in una rete neurale completamente connessa con pochi livelli nascosti. L'utilizzo di questa nuova soluzione potrebbe essere una possibile estensione del seguente lavoro di tesi, sperimentando modelli di deep learning con più livelli nascosti e con modelli non totalmente connessi. Un ulteriore miglioramento sarebbe possibile utilizzando *reti neurali ricorrenti* (RNN) in grado di mantenere memoria degli stati precedenti dell'analisi, permettendo quindi di cogliere particolari legati all'evoluzione della scansione, come già tentato con successo da altri lavori presentati nel Capitolo 7.

A completare questo lavoro si potrebbe estendere l'analisi fatta sui pacchetti di rete anche al protocollo di trasporto *UDP*, per riuscire a riconoscere più servizi e ad aumentare la possibilità di individuare il traffico malevolo. I pacchetti UDP hanno una quantità minore di campi nell'intestazione rispetto al protocollo TCP e, conseguentemente, una minore quantità di statistiche disponibili. Inoltre non avendo segnalazioni sul canale trasmissivo che possono contribuire all'apprendimento rendono più complessa la loro individuazione.

Questa possibile espansione potrebbe essere un ottimo punto di partenza per aiutare la migrazione di servizi più sensibili basati su UDP, come il *protocollo SIP*[1], uno dei protocolli più utilizzati per fonia tramite Internet. Infatti la cifratura su questi servizi è ancora poco utilizzata per via della difficoltà di poter poi eseguire politiche di qualità del servizio e la natura stessa del protocollo di trasporto UDP.

[1]Session initiation protocol, è un protocollo di rete a livello applicativo in grado di gestire il trasporto di dati multimediali, come le chiamate voip (voice-over-ip).

Appendice A

Tabelle

C2S	*S2C*	*Unità*	*Descrizione Lunga*
1	15	-	Indirizzo IP del client/server
2	16	-	Porta TCP per il client/server
3	17	-	Numero totale di pacchetti osservati dal client/server
4	18	0/1	0 = nessun segmento RST è stato inviato
5	19	-	Numero di segmenti con ACK impostato a 1
6	20	-	Numero di segmenti con ACK impostato a 1 e senza dati
7	21	bytes	Numero di bytes inviati nel pyload
8	22	-	Numero di segmenti con payload
9	23	bytes	Numero di bytes trasmessi con payload (incluse le ritrasmissioni)
10	24	-	Numeri di segmenti ritrasmessi
11	25	bytes	Numero di bytes ritrasmessi
12	26	-	Numero di segmenti fuori sequenza osservati
13	27	-	Numero di segmenti con SYN impostato 1 (incluse le ritrasmissioni)
14	28	-	Nuemro di segmenti con FIN imporatato a 1 (incluse le ritrasmissioni)
29		ms	Tempo assoluto del primo pacchetto del flusso (epoch)
30		ms	Tempo assoluto dell'ultimo pacchetto del flusso (epoch)
31		ms	Durata del flusso dal primo all'ultimo pacchetto
32		ms	Tra il primo segmento del flusso e il primo con payload del client
33		ms	Tra il primo segmento del flusso e il primo con payload del server
34		ms	Tra il primo segmento del flusso e l'ultimo con payload del client
35		ms	Tra il primo segmento del flusso e l'ultimo con payload del server
36		ms	Tra il primo segmento del flusso e il primo ACK del client (senza SYN)
37		ms	Tra il primo segmento del flusso e il primo ACK del server (senza SYN)
38		0/1	1 = client ha IP interno, 0 = client ha IP esterno
39		0/1	1 = server ha IP interno, 0 = server ha IP esterno
40		0/1	1 = client IP è anonimizzato CryptoPAn
41		0/1	1 = server IP è anonimizzato CryptoPAn
42		-	Tipo di connessione identificato dal motore TCPL7
43		-	Tipo di protocollo P2P
44		-	Per flussi HTTP, identifica contenuti Web2.0
45	52	ms	RTT medio considerando il tempo tra l'invio di un segmento e l'ACK corrispondente
46	53	ms	RTT minimo osservato durante l'intera connessione
47	54	ms	RTT massimo osservato durante l'intera connessione
48	55	ms	Deviazione standard del RTT
49	56	-	Numero di RTT validi
50	57	-	Time To Live minimo
51	58	-	Time to Live massimo

Tabella A.1: Campi del formato Tstat.

C2S	*S2C*	*Unità*	*Descrizione Lunga*
59	64	-	Numero di segmenti con PSH impostato a 1
60	65	-	Numero di segmenti con URG impostato a 1
61	66	-	Numero di segmenti con ECE impostato a 1
62	67	-	Numero di segmenti con CWR impostato a 1
63	68	-	Numero di segmenti con NS impostato a 1

Tabella A.2: Campi aggiuntivi del formato Tstat esteso.

Codice attributo	*Min*	*Max*	*Media*	*Deviazione Standard*
c_packets	1	12512	419.04	1200.95
c_flag_R	0	17	0.03	0.21
c_flag_A	0	12511	418.04	1200.95
c_pureAck	0	11597	380.78	1125.05
c_data_byte_uniq	0	787755	12446.70	47270.81
c_data_pkts	0	1051	37.17	102.93
c_data_byte	0	787755	12535.34	47300.52
c_rexmit_pkts	0	669	16.11	66.43
c_rexmit_bytes	0	17355	46.24	479.68
c_out_of_seq_pkts	0	0	0	0
c_flag_S	0	3	1.00	0.07
c_flag_F	0	6	0.06	0.26
s_server_port	1	48619	884.47	1865.84
s_packets	1	19261	656.44	1929.72
s_flag_R	0	6	0.02	0.15
s_flag_A	1	19259	656.44	1929.72
s_pureAck	0	419	14.78	36.24
s_data_byte_uniq	0	$2.55 \cdot 10^7$	$8.59 \cdot 10^5$	$2.59 \cdot 10^6$
s_data_pkts	0	19187	640.58	1917.98
s_data_byte	0	$2.56 \cdot 10^7$	$8.60 \cdot 10^5$	$2.60 \cdot 10^6$
s_rexmit_pkts	0	128	1.47	5.74
s_rexmit_bytes	0	181500	755.87	6719.84
s_flag_S	0	6	1.01	0.23
s_flag_F	0	10	0.07	0.31
s_packets	1	19261	656.44	1929.72
s_flag_R	0	6	0.21	0.15
s_flag_A	1	19259	656.44	1929.72
s_pureAck	0	419	14.78	36.24
s_data_byte_uniq	0	$2.55 \cdot 10^7$	$8.59 \cdot 10^5$	$2.59 \cdot 10^6$
s_data_pkts	0	19187	640.58	1917.98
s_data_byte	0	$2.56 \cdot 10^7$	$8.60 \cdot 10^5$	$2.60 \cdot 10^6$
s_rexmit_pkts	0	128	1.47	5.74
s_rexmit_bytes	0	181500	755.87	6719.84
s_flag_S	0	6	1.01	0.23
s_flag_F	0	10	0.07	0.31
completation_time	0	363327.20	8449.80	31618.89
C_first_payload	0	288027.43	130.76	1702.32
S_first_payload	0	288027.43	130.76	1702.32
C_last_payload	0	363256.20	6819.04	29688.76
S_last_payload	0	363327.20	6839.38	28456.33
C_first_ack	0	288027.43	147.60	2005.28
S_first_ack	0	288027.43	147.60	2005.28
rtt_c_cnt	0	1051	37.54	102.71
rtt_s_cnt	0	19187	640.11	1917.48
c_flag_P	0	912	34.75	97.38
s_flag_P	0	2132	62.86	186.96

Tabella A.3: Statistiche delle metriche estratte dal dataset tramite Netstatpy.

Appendice B

Manuale utente

In questa Sezione verranno descritti i passaggi per l'installazione e l'utilizzo dei due software realizzati per questo studio: Netstatpy e Vulnscanpy.

B.1 Prerequisiti

Entrambi sono scritti in `Python` ed in particolare è necessaria come requisito minimo la sua versione 3.7. Per l'installazione dei moduli necessari al funzionamento dei diversi programmi si utilizzerà il gestore dei pacchetti python `pip3`.

B.1.1 Python 3.7

Per installare `Python 3.7` si può procedere nel seguente modo su sistemi operativi Windows:

1. Scaricare il binario presente sul sito https://www.python.org/downloads/
2. Installare l'eseguibile facendo attenzione ad aggiungere il percorso di installazione nelle variabili di sistema.

Invece per sistemi operativi basati su Linux che utilizzano il gestore dei pacchetti `apt`, come la distribuzione Debian, utilizzare il seguente comando:

```
$ apt-get install python3.7
```

B.1.2 Pip3

Il gestore dei pacchetti `pip3`, nei sistemi che hanno installato `python 3.7` tramite il download dell'eseguibile, è già compreso. Invece per i sistemi operativi Linux, va eseguito il seguente comando:

```
$ apt-get install python3-pip
```

B.1.3 Dipendenze

In entrambi i software è presente nella radice della cartella un file denominato `requirements.txt`, che contiene i diversi moduli python necessari al loro funzionamento. Questa operazione va eseguita dopo i passaggi per l'installazione dei due programmi.

Per entrambi è sufficiente avere a disposizione un terminale e spostarsi nella cartella del software da installare e lanciare il seguente comando:

```
$ pip install -r requirements.txt
```

Questa operazione va ripetuta in entrambe le cartelle dei due programmi.

B.1.4 Ambiente di analisi

Per creare un ambiente di analisi è necessario che entrambi gli strumenti siano presenti all'interno della stessa cartella e le operazioni che richiamano i due programmi vengano eseguite all'interno di essa.

Il nome della cartella può essere totalmente arbitrario, in queste Sezioni di Manuale verrà utilizzato il nome `envtest`.

B.2 Netstatpy

Netstatpy è un software di analisi del traffico di rete, scritto in `Python`, in grado di estrarre statistiche sulle intestazioni dei segmenti TCP, vedere Sezione 2.1.1 e Sezione 4.3.

B.2.1 Installazione

Per installare *netstatpy* è necessario possedere la cartella `netstatpy` allegata a questa tesi, contenente il software e successivamente bisogna seguire questi passaggi:

1. copiare la cartella nell'ambiente di analisi, come spiegato nella Sezione B.1.4, nell'esempio `envtest`;
2. installare le dipendenze come illustrato in B.1.3.

Per verificare l'installazione è possibile lanciare il seguente comando nell'ambiente di analisi:

```
$ python -m netstatpy -h
```

Questo comando restituirà, se il programma è correttamente installato, il manuale da linea di comando dove vengono elencate le varie opzioni con cui può essere eseguito e una breve spiegazione.

B.2.2 Utilizzo

Per eseguire il programma con le diverse opzioni bisogna utilizzare il seguente comando:

```
$ python -m netstatpy [-h] [-p PCAP_FILE] [-o [OUTPUT_FILE]] [-t
    [THREAD]] [-s STEP] [--live] IP_HOST
```

L'argomento posizionale relativo all'indirizzo IP dell'host è obbligatorio, mentre gli altri sono opzionali. Di seguito vengono descritte le diverse opzioni possibili:

`-h, --help`: mostra il messaggio d'aiuto e termina l'esecuzione.

`IP_HOST`: IP dell'host con cui viene fatta la cattura.

Questo viene utilizzato per filtrare i diversi pacchetti di rete che non possiedono l'intestazione IP (es. ARP) e di tenere solamente i pacchetti relativi all'indirizzo IP sull'interfaccia desiderata.

`--live`: analisi live.

Questa opzione abilita la cattura sull'interfaccia di rete principale e ne elabora i diversi pacchetti raccolti in tempo reale e attiva la modalità live.

`-p PCAP_FILE`: nella modalità live è usata per salvare la sessione, mentre in modalità normale è usata come sessione da analizzare.

Questa opzione ha due comportamenti distinti in base al contesto nel quale viene utilizzato, nel caso di un'analisi live questo rappresenta il percorso del file nel quale la sessione catturata verrà salvata. Invece in un'analisi offline rappresenta il percorso del file pcap da utilizzare per essa.

`-o [OUTPUT_FILE]`: salvare le statistiche su file. `OUTPUT_FILE` è opzionale.

Questa opzione permette di specificare che l'output dell'analisi vada salvato all'interno di un file sul filesystem, perchè altrimenti il comportamento di base è la stampa delle statistiche sul terminale. Con la sola opzione attivata viene automaticamente generato un file con un nome di default contenente la data e l'ora dell'analisi, altrimenti è possibile specificare il percorso del file nel quale salvare l'analisi.

`-t [THREAD]`: numero di thread usati, ignorato se analisi live. Se mancante, viene utilizzato un singolo thread. Se `[THREAD]` non specificato, il valore di default è 3.

Questa opzione permette di indicare se parallelizzare il calcolo dell'analisi. Nel caso di analisi live questo non è possibile, ma per analisi su pacchetti pcap si può far suddividere al programma, in maniera autonoma, i diversi pacchetti per velocizzare l'analisi. Il programma è stato testato correttamente fino a 12 thread.

`-s STEP`: step usati per l'analisi. Default: 5.

Grazie a questa opzione è possibile selezionare ogni quanti pacchetti di un determinato flusso è possibile estrarre una statistica. Si consideri che per ogni step verrà creato un dato nell'insieme di output e questo conterrà le statistiche sull'intero flusso fino all'ultimo pacchetto analizzato.

B.2.3 Configurazione

In Netstatpy è possibile modificare il formato di output con il quale i dati vengono restituiti, quello di default è il formato *tstat esteso* riportato dall'unione della Tabella A.1 e A.2.

Per modificare questa tipologia di uscita è possibile cambiare l'associazione presente nel file all'interno della cartella del programma `netstatpy/config/mapping_data.py`. In esso sono presenti alcuni dizionari `Python` nei quali avvengono le associazioni tra le nomenclature interne e quelle con le quali vengono estratte, in particolare la chiave rappresenta quella interna e il valore quella esterna. I dizionari sono suddivisi tra statistiche lato client, lato server e di flusso, per ognuno di essi ne sono presenti due, uno per compatibilità con il programma Tstat, vedere Sezione 2.2.4, e l'altro per estenderne i campi.

B.2.4 Importazione come libreria

È possibile anche importare Netstatpy all'interno di un altro programma `Python` e ottenere l'analisi sotto forma di `Dataframe`, la struttura base della libreria *Pandas*.

Per l'importazione è possibile utilizzare i seguenti comandi all'interno di uno script `Python` presente nell'ambiente di analisi (es. `envtest`):

```
from netstatpy import Netstatpy

stats = Netstatpy(IP_HOST, pcap=None, output_flag=True, output_file=None,
    live=False, training=True)
```

dove gli argomenti della funzione `Netstatpy` sono dei collegamenti con gli argomenti esposti nella Sezione precedente B.2.2, con l'aggiunta che l'argomento `training`, se impostato a `True`, permette di restituire l'analisi sotto forma di `Dataframe` come precedentemente citato.

B.3 Vulnscanpy

Vulnscanpy è un software, scritto in `Python`, creato per automatizzare l'addestramento e la valutazione delle performance di classificatori basati su apprendimento automatico. Esso, utilizzando un dataset in ingresso, è in grado di: preprocessarlo, selezionarne le feature più importanti, ricercare i valori migliori per gli iperparametri e produrre report per valutarne l'efficacia, tutte operazioni spiegate all'interno del Capitolo 3.

B.3.1 Installazione

Per installare *vulnscanpy* è necessario possedere la cartella `vulnscanpy` allegata a questa tesi, contenente il software, e successivamente bisogna seguire questi passaggi:

1. copiare la cartella nell'ambiente di analisi, come spiegato nella Sezione B.1.4, nell'esempio `envtest`;
2. installare le dipendenze come illustrato in B.1.3.

Per verificare l'installazione è possibile lanciare il seguente comando nell'ambiente di analisi:

```
$ python -m vulnscanpy -h
```

Questo comando restituirà, se il programma è correttamente installato, il manuale da linea di comando dove vengono elencate le varie opzioni con cui può essere eseguito e una breve spiegazione.

B.3.2 Utilizzo

Per eseguire il programma con le diverse opzioni bisogna utilizzare il seguente comando:

```
$ python -m vulnscanpy [-h] [-c CONFIG] [-l] [-M MODEL] [-b BINARY] [-m
    MULTICLASS]
```

Tutti i parametri di questo software sono opzionali, anche se ve ne sono due di cui almeno uno deve sempre essere presente (`-b BINARY` o `-m MULTICLASS`). In generale questi argomenti permettono di selezionare la configurazione da utilizzare per il flusso di esecuzione. Di seguito verranno illustrate le diverse opzioni:

`-h, --help`: mostra il messaggio d'aiuto ed esce.

`-b BINARY, --binary BINARY`: colonna usata per la classificazione binaria.

Almeno una tra questa e `-m MULTICLASS` deve essere specificata. Questa opzione serve ad indicare al software il nome della colonna contenente i target binari tra quelle del Dataframe utilizzato come input per l'analisi.

`-m MULTICLASS, --multiclass MULTICLASS` colonna utilizzata per la classificazione multi-classe.

Almeno una tra questa e `-b BINARY` deve essere specificata. Grazie a questa opzione è possibile configurare il nome della colonna contenente i target multiclasse presente nel Dataframe utilizzato come input per l'analisi.

`-l, --live`: permette di definire se il flusso di esecuzione è in uno stato di apprendimento o predittivo.

Questa opzione permette di impostare il flusso di esecuzione in modalità di apprendimento, se non impostata verranno caricati i modelli già allenati e utilizzati come classificatori saltando le fasi di training, ma comunque trasformando i dati per renderli compatibili con il modello.

Le prossime opzioni sono tuttora in fase di sviluppo o non del tutto implementate, dunque non ancora operative, ma vengono comunque descritte per fornire una visione più completa del progetto.

`-M MODEL, --model MODEL`: forza l'uso di un solo modello per l'analisi.

Di default il software carica tutti i modelli che sono stati definiti all'interno dei diversi file, questa opzione permette di ignorare tale caratteristica e utilizzare un modello in particolare.

`-c CONFIG, --config CONFIG`: configurazione per l'analisi.

Permette di caricare un file di configurazione in formato `JSON`, in grado di definire tutti i passaggi che il flusso di esecuzione deve compiere per allenare, valutare e utilizzare un modello.

B.3.3 Configurazione

Vulnscanpy è costituito da più componenti che possono essere modificati per costruire il flusso di esecuzione su misura in base all'analisi che si deve svolgere. Ne è un esempio la possibilità di cambiare l'intervallo dei valori utilizzati per gli iperparametri tra cui ricercare.

Ci sono principalmente tre aree in cui l'utente può intervenire per applicare le configurazioni:

- `vulnscanpy/lib/preprocessing.py`
- `vulnscanpy/lib/learning.py`
- `vulnscanpy/models/`

Il primo è un file nel quale possono essere definite le operazioni che vengono eseguite sul dataset come fase di preprocessamento, come spiegato nella Sezione 3.1.1.

Nel secondo file è presente una lista dei modelli disponibili che vengono caricati e/o allenati durante il flusso. Aggiungendo o rimuovendo i modelli a questa lista è possibile intervenire su quali modelli verranno utilizzati durante l'analisi.

Il terzo è un percorso nel quale sono contenute tutte le implementazioni dei modelli compatibili con il flusso di esecuzione, seguendo lo schema *"un file, un modello"*. Grazie a questa filosofia gli intervalli dei diversi iperparametri che si vogliono testare sono specificati all'interno dello stesso file nel quale il modello viene definito. Ogni modello ha un metodo chiamato `my_params()` all'interno del quale sono definiti gli iperparametri. La foresta casuale utilizzata in questa tesi, ad esempio, è dotata di questa funzione che permette di ritrovare i valori descritti nella Sezione 6.1. Il modello all'interno del filesystem si trova nell'ambiente di analisi al seguente percorso: `vulnscanpy/models/RF.py`, .

B.3.4 Importazione come libreria

Come per il software precedente, Vulnscanpy può essere importato all'interno di un altro programma `Python` per poter essere integrato o connesso ad altri flussi di esecuzione, ne è un esempio la possibilità di connettere Netstatpy e Vulnscanpy in sequenza per automatizzare interamente l'apprendimento dei classificatori.

Per l'importazione è possibile utilizzare i seguenti comandi all'interno di uno script `Python` presente nell'ambiente di analisi (es. `envtest`):

```
from vulnscanpy import Vulnscanpy

stats = Vulnscanpy(dataset, binary=None, multiclass=None, config_file=None,
    save=True, model=None, training=False)
```

dove gli argomenti della funzione `Vulnscanpy` sono dei collegamenti con gli argomenti esposti nella Sezione precedente B.3.2, con l'aggiunta che l'argomento `training` rappresenta l'opzione `-l, --live` e il dataset deve essere un *Dataframe* della libreria Pandas.

Appendice C

Manuale sviluppatore

C.1 Netstatpy

Netstatpy nasce come software facilmente personalizzabile, in questa Sezione verranno descritte la struttura del programma e la modalità per aggiungere nuove metriche.

C.1.1 Struttura

La struttura del software è composta da due cartelle principali, `lib` e `config`, e dal file `main.py` che rappresenta il punto di ingresso del programma.

lib

La cartella `lib` è il cuore del programma nel quale vengono definite le modalità di estrazione e produzione delle statistiche. In particolare, oltre a funzioni ausiliarie, è presente il file `statistic_functions.py`, nel quale si trovano le implementazioni delle funzioni che elaborano i pacchetti di rete e ne estraggono i metadati. Nella Sezione C.1.2 verrà utilizzato per aggiungere una nuova funzione.

config

All'interno della cartella `config` sono presenti due file: `func_2_stat.py` e `mapping_data.py`. Il primo si occupa di tradurre le metriche estratte dal software in campi interni da poter utilizzare all'interno del programma stesso; il secondo invece, come già anticipato nella Sezione 4.3.3, serve a creare un formato di uscita per il programma riordinando e selezionando i campi interni.

All'interno di entrambi questi file sono presenti delle sezioni dedicate alle statistiche lato client, lato server e di flusso. Intervenendo su di esse è quindi possibile, utilizzando i metadati estratti dalle funzioni precedentemente descritte, creare nuove statistiche e nuovi formati di output.

C.1.2 Aggiungere una nuova statistica

Per aggiungere una nuova metrica su Netstatpy è necessario descrivere la lavorazione del pacchetto di rete, quale statistica si vuole raccogliere e in che formato la si vuole esporre come output. Per illustrare meglio questo strumento verrà utilizzato un esempio per la creazione di una nuova metrica in grado di conteggiare il numero di segmenti con il flag ECE, vedere Sezione 2.1.1, presenti lato client in un flusso TCP.

Come prima operazione è necessario creare una funzione che manipoli il pacchetto di rete nel file `netstatpy/lib/statistic_functions.py`, per questioni di ottimizzazione si cerca di raggruppare conteggi simili sotto la stessa funzione. Per questo motivo nel seguente esempio sarebbe

possibile collegare al conteggio dei flag ECE anche il conteggio degli altri, come già eseguito dalla funzione `flags_stats(c_session)`. Questa funzione restituisce un dizionario `Python` contenente come chiave il flag e come valore il conteggio per il flusso.

Come seconda operazione è necessario dare una nomenclatura interna alla statistica che si sta implementando, questo è possibile all'interno del file `netstatpy/config/func_2_stat.py`, nel quale, utilizzando la sezione per le statistiche di singolo canale, è possibile aggiungere alla funzione `statistics_single_channel(c_session)` l'etichetta *flag_E* riconducendo il contenuto del dizionario con il corrispondente flag.

A questo punto è possibile inserire all'interno del file `netstatpy/config/mapping_data.py`, nella sezione dedicata alle statistiche lato client estese, la metrica appena creata, utilizzando una nuova etichetta che corrisponderà al nome del campo al di fuori del software, in questo caso *c_flag_E*.

Si noti che sarebbe stato possibile, grazie al doppio sistema di mappatura, combinare l'estrazione dei flag ECE con altri fattori, così da non limitarsi semplicemente a reindirizzare il contenuto del dizionario, ma creando a tutti gli effetti una statista nuova rielaborata a partire dall'intestazione del segmento.

C.2 Vulnscanpy

In questa Sezione verranno descritte le operazioni necessarie per aggiungere un nuovo modello di apprendimento automatico e la struttura del software.

C.2.1 Struttura

Vulnscanpy è composto da due cartelle `lib`, `models` e da un file `main.py` che funge da punto di inizializzazione del programma. Nella prima cartella sono presenti le funzioni che definisco il flusso di esecuzione di default e nella seconda sono invece definiti i modelli personalizzati da allenare e le loro specifiche.

lib

Nella cartella `lib` sono presenti tre file principali: `preprocessing.py`, `learning.py` e `pipeline.py`. Il primo si occupa di tutte le operazioni di preprocessamento comuni a tutti i modelli, descritte nella Sezione 3.1.1. Nel secondo sono invece caricati in memoria i diversi modelli con le impostazioni base. Infine il terzo invece si occupa di aggregare tutte le operazioni descritte nei precedenti, eseguirle e generarne poi i report.

models

Nella cartella `models` sono presenti i modelli per l'analisi, in particolare ve ne è uno base detto `Dummy.py`, un file contenente una classe omonima che si occupa di interfacciare la libreria che si occupa del machine learning con il flusso di esecuzione. Gli altri modelli sono tutte classi `Python` che estendono la classe *DummyModel* ereditando quindi i metodi base.

C.2.2 Aggiungere un nuovo modello

Per illustrare i passaggi necessari per aggiungere un nuovo classificatore al flusso di esecuzione, si utilizzerà come esempio l'aggiunta di un modello di rete neurale.

Come prima operazione è necessario aggiungere il file del modello nella cartella d'ambiente al percorso `vulnscanpy/models`, in questo caso il file `NN.py`. Al suo interno è necessario importare la classe *DummyModel* ed estenderla:

```
from vulnscanpy.models.Dummy import DummyModel
class NN(DummyModel):
    pass
```

Nella classe appena creata sarà necessario aggiungere due metodi obbligatori `__init__` e `my_params`. Il primo si deve occupare della raccolta dei parametri esterni al modello (es. numero di classi) e che permettano di configurarlo (es. numero di nodi nel livello di output basati sul numero di classi), il secondo deve invece costruire un dizionario `Python` contenente gli iperparametri che si vogliono testare del modello, rispettando il formato utilizzato da Scikit-learn nelle Pipeline[1].

Il nuovo modello va quindi aggiunto all'interno della lista presente nella funzione all'interno del file `vulnscanpy/lib/learning.py` chiamata `config_models(n_class, weight)` . Nella lista è possibile inserire l'inizializzazione del modello con i parametri esterni desiderati. Nell'esempio presentato è possibile indicare il numero di nodi da utilizzare per il livello di uscita e il peso per ogni classe,utilizzando la seguente linea di codice che inizializza il modello con i corretti parametri:

```
NN(module__output_dim=n_class, weight=weight)
```

A questo punto il modello è caricato e inserito nella coda di esecuzione quando il programma verrà lanciato.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Bibliografia

[1] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep Packet Inspection over Encrypted Traffic", ACM SIGCOMM Computer Communication Review, London (United Kingdom), August 17-21, 2015, pp. 213–226, DOI 10.1145/2785956.2787502

[2] M. Fujimoto, W. Matsuda, and T. Mitsunaga, "Detecting attacks leveraging vulnerabilities fixed in MS17-010 from Event Log", 2019 IEEE Conference on Application, Information and Network Security (AINS), Pulau Pinang (Malaysia), November 19-21, 2019, pp. 42–47, DOI 10.1109/AINS47559.2019.8968703

[3] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", IEEE Transactions on Communications, vol. 28, April 1980, pp. 425–432, DOI 10.1109/TCOM.1980.1094702

[4] R. Denenberg, "Data communications and OSI", Library Hi Tech, vol. 8, April 1990, pp. 15–32, DOI 10.1108/eb047805

[5] T. J. Socolofsky and C. J. Kale, "TCP/IP tutorial." RFC-1180, January 1991, DOI 10.17487/RFC1180

[6] J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach", Pearson, 2016, ISBN: 978-0-13-359414-0

[7] J. Postel, "Transmission Control Protocol." RFC-793, September 1981, DOI 10.17487/RFC793

[8] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry." RFC-6335, August 2011, DOI 10.17487/RFC6335

[9] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP." RFC-3168, September 2001, DOI 10.17487/RFC3168

[10] V. Jacobson, B. Braden, and D. Borman, "TCP Extensions for High Performance." RFC-1323, May 1992, DOI 10.17487/RFC1323

[11] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer." RFC-6298, June 2011, DOI 10.17487/RFC6298

[12] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control." RFC-5681, September 2009, DOI 10.17487/RFC5681

[13] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window." RFC-3390, October 2002, DOI 10.17487/RFC3390

[14] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." RFC-5280, May 2008, DOI 10.17487/RFC5280

[15] A. Freier, P. Karlton, and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0." RFC-6101, August 2011, DOI 10.17487/RFC6101

[16] S. A. Thomas, "SSL and TLS Essentials: Securing the Web", John Wiley & Sons, 2000, ISBN: 978-0-471-38354-3

[17] S. Turner and T. Polk, "Prohibiting Secure Sockets Layer (SSL) Version 2.0." RFC-6176, March 2011, DOI 10.17487/RFC6176

[18] T. Dierks and C. Allen, "The TLS Protocol Version 1.0." RFC-2246, January 1999, DOI 10.17487/RFC2246

[19] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC-8446, August 2018, DOI 10.17487/RFC8446

[20] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, "Hypertext Transfer Protocol – HTTP/1.0." RFC-1945, May 1996, DOI 10.17487/RFC1945

[21] T. Berners-Lee, R. T. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax." RFC-3986, January 2005, DOI 10.17487/RFC3986

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." RFC-2068, January 1997, DOI 10.17487/RFC2068

[23] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." RFC-2616, June 1999, DOI 10.17487/RFC2616

[24] E. Rescorla, "HTTP Over TLS." RFC-2818, May 2000, DOI 10.17487/RFC2818

[25] K. P. Murphy, "Machine learning: a probabilistic perspective", MIT Press, 2013, ISBN: 978-0-262-01802-9

[26] M. Nikulin and V. Voinov, "Chi-Squared Tests–II", Encyclopedia of Statistical Sciences, American Cancer Society, August 2006, DOI 10.1002/0471667196.ess3126

[27] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", Information Processing & Management, vol. 45, July 2009, pp. 427–437, DOI 10.1016/j.ipm.2009.03.002

[28] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An Introduction to Statistical Learning: With Applications in R", Springer Publishing Company, Incorporated, 2014, ISBN: 978-1-4614-7137-0

[29] C. M. Bishop, "Pattern Recognition and Machine Learning (Information Science and Statistics)", Springer-Verlag, 2006, ISBN: 978-0387-31073-2

[30] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning", Nature, vol. 521, May 2015, pp. 436–444, DOI 10.1038/nature14539

[31] Y. Ho and S. Wookey, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling", IEEE Access, vol. 8, December 2019, pp. 425–432, DOI 10.1109/access.2019.2962617

[32] F. Chollet, "Deep Learning with Python", Manning, November 2017, ISBN: 978-1-617-29443-3

[33] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp", Neural Networks: Tricks of the Trade, pp. 9–48, December 6-7, 1998, DOI 10.1007/3-540-49430-8_2

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", International Conference on Learning Representations 2015, San Diego (CA, USA), May 7-9, 2015. https://arxiv.org/abs/1412.6980

[35] R. Koch and G. D. Rodosek, "Command Evaluation in Encrypted Remote Sessions", 2010 Fourth International Conference on Network and System Security, Melbourne (Australia), September 1-3, 2010, pp. 299–305, DOI 10.1109/NSS.2010.62

[36] A. Balla, A. Stassopoulou, and M. D. Dikaiakos, "Real-time web crawler detection", 2011 18th International Conference on Telecommunications, Ayia Napa (Cyprus), May 8-11, 2011, pp. 428–432, DOI 10.1109/CTS.2011.5898963

[37] X. Yuan, C. Li, and X. Li, "Deepdefense: Identifying ddos attack via deep learning", 2017 IEEE International Conference on Smart Computing (SMARTCOMP), Hong Kong (China), May 29-31, 2017, pp. 1–8, DOI 10.1109/SMARTCOMP.2017.7946998

[38] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks", IEEE Access, vol. 6, August 2018, pp. 48231–48246, DOI 10.1109/ACCESS.2018.2863036

[39] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng, "Malware traffic classification using convolutional neural network for representation learning", 2017 International Conference on Information Networking (ICOIN), Da Nang (Vietnam), January 11-13, 2017, pp. 712–717, DOI 10.1109/ICOIN.2017.7899588

[40] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things", IEEE Access, vol. 5, September 2017, pp. 18042–18050, DOI 10.1109/ACCESS.2017.2747560

[41] B. Miller, L. Huang, and J. D. Joseph, A. D.and Tygar, "I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis", Privacy Enhancing Technologies, Amsterdam (Netherlands), July 16-18, 2014, pp. 143–163, DOI 10.1007/978-3-319-08506-7_8

[42] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service Usage Classification with Encrypted Internet Traffic in Mobile Messaging Apps", IEEE Transactions on Mobile Computing, vol. 15, January 2016, pp. 2851–2864, DOI 10.1109/TMC.2016.2516020

[43] M. Shen, Y. Liu, S. Chen, L. Zhu, and Y. Zhang, "Webpage Fingerprinting using Only Packet Length Information", ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai (CN), May 20-24, 2019, pp. 1–6, DOI 10.1109/ICC.2019.8761167