Alma Mater Studiorum · Università di Bologna
Campus di Cesena

SCUOLA DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Analisi e ricostruzione di flussi TCP con il software Caronte

Relatore
Prof. Franco Callegati

Presentata da
Emiliano Ciavatta

Correlatore
Dott. Andrea Melis
Dott. Davide Berardi

Sessione Speciale di Laurea
Anno Accademico 2019-2020

Sommario

Al giorno d'oggi la maggior parte delle applicazioni che utilizziamo sono connesse a Internet. Le aziende hanno iniziato a fornire i servizi su cloud, spostando la computazione e i dati dai dispositivi dei clienti a server raggiungibili attraverso Internet. Questo fenomeno ha causato un aumento esponenziale del traffico di rete. Esponendo però i servizi su Internet sono aumentate le minacce alla sicurezza che provengono dall'esterno e che passano attraverso la rete. Per questo motivo analizzare il traffico di rete è diventata un'attività fondamentale. I volumi di traffico generati però dalle applicazioni moderne sono elevati: servono quindi strumenti automatici che siano in grado di filtrare il traffico di rete per individuare e isolare le minacce. Esistono diversi strumenti automatici che sono in grado di rilevare e prevenire minacce di rete che funzionano analizzando i pacchetti catturati su interfacce di rete. Questi strumenti sono molto efficaci nell'individuare tentativi d'intrusione, ma non permettono di effettuare una post-analisi manuale delle minacce individuate per capire come gli utenti malevoli abbiano sfruttato le vulnerabilità e i problemi del software. La maggior parte di questi strumenti effettuano un'analisi a livello di rete, andando a ispezionare il contenuto dei pacchetti. Per capire come degli avversari sfruttano eventuali vulnerabilità del software molte volte però è necessario ricostruire il flusso logico delle conversazioni di rete. L'obiettivo di questo progetto di tesi è quindi quello di realizzare uno strumento che sia in grado d'individuare le minacce analizzando il traffico di rete e ricostruendo i protocolli utilizzati. Lo strumento sarà in grado di analizzare e ricostruire le connessioni TCP presenti in flussi di rete e tutti i protocolli basati su questo. Il processo di ricostruzione consiste nell'individuare tutti i pacchetti che formano una stessa connessione logica ed estrarre da questi il contenuto della conversazione effettuata.

Indice

Introduzione

Al giorno d'oggi la maggior parte delle applicazioni che utilizziamo sono connesse a Internet. Le aziende hanno iniziato a fornire i servizi su cloud, spostando la computazione e i dati dai dispositivi dei clienti a server distribuiti o centralizzati raggiungibili attraverso Internet.
Questo fenomeno ha portato a un aumento esponenziale del traffico di rete, perché la maggior parte dei servizi devono essere accessibili da remoto. Esponendo però i servizi su Internet sono aumentate le minacce alla sicurezza che provengono dall'esterno e che passano attraverso la rete.
Per questo motivo analizzare il traffico di rete è diventata un'attività fondamentale per individuare eventuali minacce, per correggere il software difettoso e per impedire l'accesso ai propri sistemi a utenti malintenzionati. I volumi di traffico generati però dalle applicazioni moderne sono elevati, servono quindi strumenti automatici che siano in grado di filtrare il traffico di rete per individuare e isolare le minacce.
Anche durante le competizioni di attacco/difesa nelle gare fra team in ambito di sicurezza informatica, è necessario analizzare il traffico per individuare le vulnerabilità sfruttate dagli avversari, per correggerle sui propri sistemi e replicarle sui sistemi avversari.
Esistono diversi strumenti automatici che sono in grado di rilevare e prevenire minacce di rete, che funzionano analizzando i pacchetti catturati su interfacce di rete. Questi strumenti sono molto efficaci nell'individuare tentativi d'intrusione, ma non permettono di effettuare una post-analisi manuale delle minacce individuate, per capire come gli utenti malevoli abbiano sfruttato le vulnerabilità e i problemi del software.
La maggior parte di questi strumenti effettuano un'analisi a livello di rete, andando a ispezionare il contenuto dei pacchetti. Per capire come degli

avversari sfruttano eventuali vulnerabilità del software molte volte però è necessario ricostruire il flusso logico delle conversazioni di rete, e non basta ispezionare il contenuto dei singoli pacchetti.
L'obiettivo di questo progetto di tesi è quindi quello di realizzare uno strumento che sia in grado d'individuare le minacce analizzando il traffico di rete e ricostruendo i protocolli utilizzati. Lo strumento sarà in grado di analizzare e ricostruire le connessioni TCP presenti in flussi di rete e tutti i protocolli basati su questo. Il processo di ricostruzione consiste nell'individuare tutti i pacchetti che formano una stessa connessione logica ed estrarre da questi il contenuto della conversazione effettuata.

Sommario dei capitoli

Nel **Capitolo 1** è riportata una breve introduzione agli Intrusion Detection System (IDS), gli strumenti utilizzati per analizzare il traffico di rete. Verrà descritto come funzionano e quali sono gli scopi di questi strumenti. Saranno riportati inoltre limitazioni e difetti.
Il **Capitolo 2** consiste in una panoramica delle Capture The Flag (CTF), le competizioni di sicurezza informatica, che saranno utilizzate come oggetto di studio per cui realizzare lo strumento per effettuare l'analisi di rete.
Nel **Capitolo 3** è descritto il *TCP Reassembly*, la tecnica che sarà poi implementata nello strumento per ricostruire le connessioni TCP a partire da un flusso di pacchetti di rete.
Nel **Capitolo 4** è riportata l'architettura e l'implementazione di Caronte, il progetto di questa tesi. Verranno descritte tutte le fasi per la realizzazione dello strumento, a partire dall'analisi dei requisiti fino al dispiegamento.
Il **Capitolo 5** propone alcune funzionalità più avanzate che possono essere adottate e implementate nello strumento, per rendere il processo di analisi più efficiente ed efficace. La prima proposta descritta riguarda una nuova strategia sperimentale di indicizzazione delle connessioni estratte: l'indicizzazione per differenza. La seconda proposta riguarda il tracciamento dei dispositivi attraverso alcune opzioni presenti nel protocollo TCP.

Capitolo 1

Breve introduzione agli IDS

Oggigiorno quasi tutti i dispositivi utilizzati nella quotidianità sono collegati a Internet. Quasi la totalità delle applicazioni e dei servizi che utilizziamo sono basati su cloud. La maggior parte di questi utilizzano il cloud non solo per la memorizzazione di dati dell'utente, ma anche per eseguire fisicamente le applicazioni.

Con l'avvento dell'Internet of Things molti oggetti e strumenti presenti nelle abitazioni e nelle aziende, che prima erano controllati meccanicamente, sono stati collegati a Internet per essere automatizzati e controllati da remoto. Il numero di oggetti "intelligenti" che eseguono al loro interno un software a cui è possibile interfacciarsi da remoto è aumentato esponenzialmente negli ultimi anni.

Questi fenomeni hanno causato un aumento esponenziale del traffico di rete e inevitabilmente un aumento di minacce alla sicurezza. Rendere le applicazioni e i servizi accessibili da Internet per questioni di comodità e usabilità significa esporre su Internet le interfacce con cui interagire con il software anche a utenti malintenzionati. La maggior parte delle minacce alla sicurezza proviene quindi dall'esterno attraversando la rete, e non più solo dall'interno, come succedeva decenni fa.

Per questo negli ultimi anni gli studi sui sistemi di rilevamento delle intrusioni hanno ricevuto molte attenzioni nel campo delle reti e del trasporto in Internet e nel campo della sicurezza informatica. Questo breve capitolo tratterà quindi gli *intrusion detection system* (IDS) descrivendo come fun-

zionano e come sono strutturati, e riportando le motivazioni sul perché siano fondamentali.

1.1 Cosa sono gli IDS

Un *intrusion detection system* (IDS) è un dispositivo o un'applicazione software che monitora il traffico di rete rivolto verso un'applicazione critica per rilevare eventuali minacce. Lo scopo di un IDS è quello di individuare attività dannose, come un abuso di sistemi informatici o violazioni della privacy e riportarlo agli amministratori o raccogliere gli avvisi e gli eventi in un database.
A differenza di un *intrusion prevenction system* (IPS) un IDS non blocca le intrusioni. Un'intrusione è un tentativo di compromettere la CIA, acronimo di Confidenzialità, Integrità e disponibilità (Availability). Il riconoscimento di intrusioni è il processo che monitora il traffico di rete e lo analizza per individuare eventi che tentano di superare i meccanismi di sicurezza di un computer o di una rete.
Il compito di un IDS è quindi quello di analizzare le comunicazioni di rete e identificare tramite euristiche e pattern gli attacchi più comuni che vengono effettuati sull'infrastruttura. L'analisi e il controllo vengono solitamente effettuati a livello di rete nello stack ISO/OSI. La tecnica utilizzata è nominata *deep packet inspection* (DPI) e consiste nell'ispezionare tutti i pacchetti di rete che transitano su un'interfaccia.
Esistono vari livelli di analisi, a partire da quelli meno approfonditi effettuati a livello di rete, controllando ad esempio gli header dei pacchetti IP. La maggior parte delle volte non ci si ferma a livello di rete, ma si effettua l'analisi anche a livello di trasporto, ispezionando ad esempio gli header dei segmenti dei protocolli TCP e UDP. Un IDS solitamente è in grado di riassemblare le connessioni TCP per controllare le conversazioni di rete con delle *signature*, ovvero sequenze di byte considerate malevole. Alcune tecniche prevedono anche l'ispezione e l'analisi a livello di applicazione, permettendo l'estrazione e il controllo dei metadati delle interazioni dei protocolli più comuni.
La tecnica di riassemblare le connessioni TCP è spiegata dettagliatamente nel capitolo 3. Lo strumento che si intende sviluppare, oggetto di questa tesi,

può essere utilizzato anche nel contesto delle competizioni *Capture The Flag* di tipo attacco/difesa per analizzare il traffico di rete a livello di trasporto e applicazione, per individuare vulnerabilità ed exploit. Il contesto del caso di studio è spiegato dettagliatamente nel capitolo 2.

1.2 Classificazione degli IDS

Tipologia di IDS

[1] classifica gli *intrusion prevenction system* in quattro differenti categorie:

- **Network-based intrusion prevention system (NIPS)**: monitorano il traffico dell'intera rete, che può essere utilizzata per le comunicazioni di più dispositivi, per individuare minacce e intrusioni attraverso l'analisi dei protocolli di rete;

- **Wireless intrusion prevention system (WIPS)**: monitorano il traffico di rete che viene scambiato soltanto da interfacce e protocolli wireless;

- **Network behavior analysis (NBA)**: esaminano il traffico di rete per identificare le minacce che generano flussi di traffico insoliti, come attacchi DDoS (Distributed Denial of Service), alcune forme di malware e violazioni delle policy;

- **Host-based intrusion prevention system (HIPS)**: strumenti che devono essere installati sui singoli host che monitorano gli eventi e analizzano le attività sospette che vengono effettuate su quel determinato dispositivo.

Metodi di rilevamento

Oltre alla classificazione in base a dove il processo di rilevamento avviene è possibile classificare gli IDS in base al metodo di rilevamento utilizzato. [2] classifica i metodi di rilevamento in tre macro categorie:

- **Signature-based detection**. Una signature è un pattern o una stringa che corrisponde a un attacco o una minaccia noti. Il *signature detection* è il processo per confrontare i pattern con gli eventi catturati per riconoscere possibili intrusioni. Le signature devono essere preconfigurate e sono statiche, ovvero non cambiano se non vengono aggiornate manualmente. Siccome le signature sono costruite sulla base di attacchi rilevati e sulle vulnerabilità di sistema scoperte, il *signature detection* è anche noto come rilevamento basato sulla conoscenza (*Knowledgebased Detection*).

  Il vantaggio di questo sistema è che costituisce il metodo più semplice ed efficace per rilevare gli attacchi noti. Lo svantaggio è che risulta inefficace per rilevare attacchi sconosciuti, attacchi di evasione e varianti dello stesso attacco. Inoltre, è difficile e oneroso in termini di tempo mantenere le signature e i pattern aggiornati.

- **Anomaly-based detection**. Un'anomalia è una deviazione di un comportamento conosciuto, mentre un profilo rappresenta il normale o atteso comportamento che deriva dal monitoraggio di attività legittime, connessioni di rete, dispositivi e utenti in un certo intervallo di tempo. I profili possono essere sia statici sia dinamici, e sviluppati per diversi attributi, ad esempio i tentativi di login falliti, la percentuale di utilizzo del processore, il numero di e-mail inviate. L'*anomaly detection* compara i profili registrati con gli eventi osservati per riconoscere gli attacchi più significativi. Per questo motivo questa tecnica è anche chiamata rilevamento in base al comportamento (*Behaviour-based Detection*).

  Il vantaggio di questo sistema è che può rilevare efficacemente nuovi attacchi e vulnerabilità non viste prima. È meno dipendente dai protocolli e dal sistema operativo, e facilita la rilevazione di abusi di privilegi. Come svantaggio c'è la difficoltà a creare i profili e tenerli aggiornati, e la difficoltà a inviare gli avvisi in tempo quando vengono rilevati comportamenti scorretti.

- **Stateful protocol analysis detection**. *Stateful* significa che viene considerato e tracciato lo stato dei protocolli. I pacchetti quindi vengono analizzati nell'insieme, e non solo singolarmente. Questo metodo identifica le anomalie degli stati del protocollo, confrontando gli eventi osservati con profili predeterminati di definizioni generalmente accettate di attività benigne. I modelli di protocollo di rete considerati benigni si basano solitamente sugli standard dei protocolli internazionali.

  Questa tipologia di IDS è in grado di individuare le sequenze di operazioni e di comandi scorretti attraverso il tracciamento dello stato dei protocolli. Ha lo svantaggio però che l'operazione di tracciamento e analisi è costosa computazionalmente, e non permette di ispezionare gli attacchi che sembrano comportamenti benigni del protocollo.

1.3 Limitazioni degli IDS

Gli *intrusion detection system* sono degli strumenti potenti che hanno però delle limitazioni. Di seguito è riportato un elenco di casistiche in cui un IDS può non funzionare.

- I pacchetti cifrati non possono essere analizzati perché è impossibile estrarre il contenuto del pacchetto senza conoscere la chiave. Nella maggior parte dei protocolli moderni che utilizzano crittografia simmetrica la chiave non è statica ma è generata da entrambe le parti tramite un algoritmo di scambio di chiavi come Diffie-Hellman, descritto in [3].

- I pacchetti danneggiati lungo il percorso o generati da software con bug o problemi possono essere riconosciuti come tentativi di intrusione e considerati come pacchetti malevoli. I pacchetti corrotti possono generare un elevato numero di falsi positivi, rendendo difficile l'individuazione delle minacce reali.

- Molti attacchi sfruttano problemi legati alla logica del software o alla sua implementazione e non vulnerabilità note comuni a tutti i protocol-

li. Negli IDS basati su signature è impossibile rilevare questa tipologia di attacchi se non si hanno regole per gli specifici software.

- Gli IDS sono strumenti critici che ricevono in input il traffico non filtrato proveniente dalla rete. Eventuali vulnerabilità presenti negli IDS possono essere sfruttate attraverso l'invio di pacchetti invalidi o costruiti in maniera strategica per ingannare l'IDS o addirittura interrompere l'esecuzione.

- Gli IDS basati su signature devono essere costantemente aggiornati per far fronte anche alle nuove minacce. Un attaccante può sfruttare il lasso di tempo prima che le signature di nuove vulnerabilità vengano scritte e caricate sul software.

Capitolo 2

Caso di studio: le CTF

Una Capture the Flag (abbreviato in CTF) è una competizione di sicurezza informatica, dove team o singoli cercano vulnerabilità in sistemi e software messi a disposizione dagli organizzatori della competizione, al fine di sfruttarle e di collezionare le varie flag nascoste sul sistema bersaglio. Oltre a trovare e sfruttare vulnerabilità molte challenge consistono nel risolvere puzzle logici o capire come funziona un sistema e abusarne. Il gioco è ispirato e prende il nome da Rubabandiera, che in inglese è chiamato appunto Capture the Flag.

Le CTF di sicurezza informatica servono come esercizio educativo per offrire ai partecipanti esperienza nell'individuare vulnerabilità nel codice o in applicazioni esistenti e nel proteggere una macchina o un sistema. Durante una CTF si impara a condurre e reagire ad attacchi informatici con lo scopo di poter portare le conoscenze acquisite nel mondo reale per individuare codice o applicazioni non sicuri e correggerli, e mettere in sicurezza sistemi informatici.

Le conoscenze acquisite durante le CTF possono essere sfruttate per individuare vulnerabilità in programmi e siti web anche popolari. Molte aziende offrono programmi di bug bounty per permettere a professionisti di individuare le vulnerabilità nei loro sistemi, ricompensandoli in denaro quando vengono individuati bug che possono essere pericolosi per la sicurezza degli utenti.

Ci sono vari tipi di Capture the Flag. I più famosi sono Jeopardy e Attack/Defense. Seppure questi siano i formati più popolari, ci sono rari casi di

CTF con formato misto, per esempio Jeopardy + Attack/Defense, o che non rispettano alcun formato particolare.
Le CTF di tipo Jeopardy sono probabilmente i più popolari, dato che si adattano bene a competizioni online. In questo formato gli organizzatori mettono a disposizione dei giocatori diverse challenge, ognuna delle quali offre una singola flag. Quando un giocatore trova la flag, a seconda della difficoltà della challenge, guadagna dei punti. Il giocatore con il numero più alto di punti vince la CTF. Questo formato è generalmente a squadre. Le challenge vengono divise e assegnate tra i membri della squadra per velocizzare la loro risoluzione e la conquista delle flag, dato che in genere le competizioni hanno un limite di tempo.
Le CTF di tipo Attack/Defense verranno descritte e analizzate più approfonditamente nella prossima sezione, perché saranno oggetto del caso di studio di questa tesi.

2.1 Attack/Defense

Nel formato Attack/Defense i team che partecipano alla competizione non devono attaccare l'infrastruttura degli organizzatori, bensì tutti i team avversari. In questa tipologia di CTF quindi i giocatori hanno un duplice ruolo: cercare vulnerabilità nei sistemi avversari e sfruttarle per rubare le flag, e difendere i propri servizi e la propria infrastruttura per prevenire che gli avversari rubino le flag contenute nei propri sistemi.
Nelle Attack/Defense gli organizzatori forniscono ai giocatori, organizzati solitamente in squadre, una o più macchine virtuali preconfigurate con dei servizi vulnerabili. Nelle competizioni fisiche in cui tutti i team si ritrovano nello stesso luogo solitamente è l'organizzatore che installa le macchine virtuali e configura l'infrastruttura di rete. Nelle competizioni virtuali è compito di ciascuna squadra installare le macchine virtuali sulla propria infrastruttura, che può essere composta da macchine fisiche o cloud.
Le macchine vulnerabili devono far parte di una stessa rete per poter essere accessibili da tutte le squadre che partecipano alla competizione. Devono quindi far parte di una rete di gioco, che solitamente è privata per non rendere le macchine vulnerabili accessibili da Internet. In una competizione fisica

la rete di gioco può essere formata banalmente attraverso una Local Area Network (LAN), collegando macchine vulnerabili attraverso degli switch di rete. In una competizione effettuata in remoto ciascun team deve connettere la propria infrastruttura alla rete di gioco utilizzando una rete virtuale privata (VPN).

Punteggio

Ogni competizione ha una durata prefissata, stabilita a priori. Il tempo di gioco è suddiviso in round, intervalli di tempo regolari stabiliti dall'organizzatore. Esiste un servizio messo a disposizione dall'organizzatore che scandisce il tempo, il game master, consultabile da ciascun team. In ogni round ciascun partecipante può accumulare dei punti, che saranno sommati man mano che la competizione avanza e i round scorrono. La somma dei punti ottenuti in ciascun round determina il punteggio finale di ogni squadra.
In ogni round il game master contatta tutti i servizi di ogni team e inserisce una flag diversa. Ogni servizio corrisponde ad una challenge. Tutti i partecipanti hanno gli stessi servizi, con le stesse API e gli stessi protocolli di interazione. È compito di ogni squadra mantenere i servizi attivi, perché ad ogni round il game master controlla utilizzando il protocollo di ogni servizio se le flag che ha inserito sono presenti e sono valide.
Il punteggio viene calcolato per ciascun team nel seguente modo:

- sommando i **punti di attacco**, ovvero i punti che si ottengono quando si rubano le flag degli avversari. Per massimizzare il punteggio occorre rubare e consegnare le flag di ogni round, di ogni team, e per ogni servizio o challenge disponibile. Le flag rubate devono poi essere consegnate utilizzando un protocollo stabilito dall'organizzatore al game master, che assegna i punti al team.

  Per poter rubare le flag è necessario individuare una vulnerabilità nei servizi e scrivere un exploit per sfruttarla. Ogni servizio o challenge può contenere una o più vulnerabilità, a discrezione degli organizzatori. Solitamente le flag presenti nei database di ogni servizio sono accessibili soltanto a utenti autenticati, le cui credenziali possiede soltanto il game master. Per rubare le flag dal database è necessario

che i team individuino vulnerabilità che permettano loro di superare l'autenticazione.

- sottraendo i **punti di difesa**, ovvero i punti che si perdono quando gli avversari riescono a esfiltrare le flag dai propri servizi. La macchina vulnerabile di ogni team deve essere esposta sulla rete di gioco, essendo quindi esposta a tutti gli altri avversari. Se gli altri partecipanti della competizione individuano delle vulnerabilità possono sfruttarle per rubare le flag, facendo diminuire i punti al team a cui sono state rubate.

  Ciascun team, quando individua una vulnerabilità per fini d'attacco, deve anche sistemare i bug sui propri servizi per impedire che gli avversari possano sfruttarli per esfiltrare le flag. Questa operazione, chiamata *patching*, serve per impedire un accesso non autorizzato ai propri servizi da parte degli avversari, consentendo comunque al game master di verificare se le flag da lui inserite sono presenti e accessibili.

- sommando i **punti di SLA**, ovvero i punti di *service level agreement*. Questi punti sono necessari per premiare i team che mantengono attivi i servizi. Se questi punti non fossero aggiunti, i partecipanti spegnerebbero la macchina vulnerabile o renderebbero inaccessibili i servizi per impedire che gli avversari possano rubare le flag, azzerando i punti di difesa persi. I punti di SLA costringono quindi i partecipanti a tenere le macchine vulnerabili e i servizi attivi e *patchare* le vulnerabilità quando le individuano.

  I punti di SLA vengono calcolati dal game master. Durante i round il game master effettua delle richieste a sorpresa a ogni servizio dei vari team: se il servizio controllato è attivo e fornisce la flag corretta inserita nel round precedente dal master vengono assegnati i punti di SLA, altrimenti no. Il master si camuffa tra le richieste degli avversari per impedire che i team lo riconoscano facilmente per autorizzare soltanto le sue richieste e bloccare le altre. Il game master effettua quindi le richieste a intervalli di tempo irregolari per impedire che le sue richieste vengano riconosciute.

La squadra che al termine della competizione ottiene più punti vince.

Rete di gioco

Per poter partecipare a una competizione attacco/difesa ciascun team deve connettere la macchina vulnerabile alla rete di gioco, in modo tale che possa essere accessibile dal game master e dai team avversari. Anche i membri del team devono poter accedere alla rete del gioco per poter lanciare attacchi contro gli avversari e per consegnare al game master le flag che vengono esfiltrate.
Nelle CTF "offline" la rete di gioco è una rete LAN e ogni team ha un router che connette i pc dei membri del team e la loro macchina vulnerabile all'infrastruttura della competizione. Nelle CTF "virtuali", effettuate in remoto, ciascun team collega la propria infrastruttura alla rete di gioco attraverso una VPN, che deve essere correttamente configurata sul router di ciascun partecipante.
Gli organizzatori solitamente forniscono a ciascun team una sottorete che fa parte della rete di gioco, per poter assegnare alla macchina vulnerabile e alle altre risorse di ciascun team degli indirizzi univoci. Ciascun team deve assegnare alla propria macchina vulnerabile un indirizzo statico nella propria sottorete deciso dagli organizzatori. Gli indirizzi delle macchine vulnerabili degli avversari sono quindi fissi e conosciuti a tutti.
Gli organizzatori installano nell'infrastruttura di gioco uno o più router che connettono le sottoreti dei team e i servizi relativi alla competizione, come il game master. I router degli organizzatori sono collegati direttamente ai router di ciascun team, che fanno da ponte con l'infrastruttura di gioco. In questo modo la topologia della rete si semplifica e i pc e gli strumenti di ciascun team non sono esposti direttamente sulla rete di gioco. Solitamente ciascun team configura infatti il NAT sul proprio router per tradurre gli indirizzi della propria sottorete per nascondere il mittente e per renderlo inaccessibile dalla rete di gioco.
I router dell'infrastruttura di gioco oltre ad avere la funzione di connettere le sottoreti dei partecipanti hanno anche la funzione di riscrivere gli indirizzi dei pacchetti che inoltrano. Questa operazione, realizzata attraverso l'uso di

un NAT, viene effettuata per nascondere il mittente negli attacchi rivolti alle macchine vulnerabili. Se non ci fosse un NAT che riscrivesse gli indirizzi, ciascun team potrebbe utilizzare un firewall per bloccare i pacchetti provenienti dalle sottoreti degli avversari e permettere soltanto i pacchetti inviati dal game master, rendendo ingiocabile la competizione. Con l'utilizzo di un NAT sui router di gioco i team ricevono gli attacchi da uno stesso indirizzo e sono quindi impossibilitati a filtrare i pacchetti e a distinguere le richieste effettuate alla macchina vulnerabile se sono legittime o malevole.

Esempio di una rete di gioco

Nella figura 2.1 è riportato lo schema della rete di gioco utilizzato durante la competizione FAUST CTF 2019[1].
Nella figura sono mostrate soltanto le sottoreti di due team, il team 1 e il team 2, ma in una competizione sono presenti centinaia di team. Ad ogni team è assegnata una sottorete con netmask 24, con indirizzo `10.66.X.0/24`, dove `X` è l'id del team. Ciascun partecipante deve installare il router a indirizzo `10.66.X.1` e la macchina vulnerabile (vulnbox in figura) a indirizzo `10.66.X.2`.
L'organizzatore assegna a ciascun team un gateway per connettersi alla rete di gioco, disponibile all'indirizzo `10.65.X.1`. Tutti i pacchetti ricevuti dalla rete di gioco hanno come quindi indirizzo IP mittente l'indirizzo del gateway assegnato a ciascun team.
All'interno della rete di gioco sono presenti anche i servizi dell'organizzazione, nascosti ai partecipanti, come il game server. È presente, infine, un servizio accessibile da tutti che permette ai partecipanti di consegnare le flag (`submission.faustctf.net` in figura).

Strumenti utili

Per sostenere una competizione attacco/difesa, oltre ad avere le competenze e le conoscenze sui vari ambiti della sicurezza informatica (come crittografia, reverse engineering, web security, pwning) servono anche i giusti strumenti,

[1] https://2019.faustctf.net/information/setup/

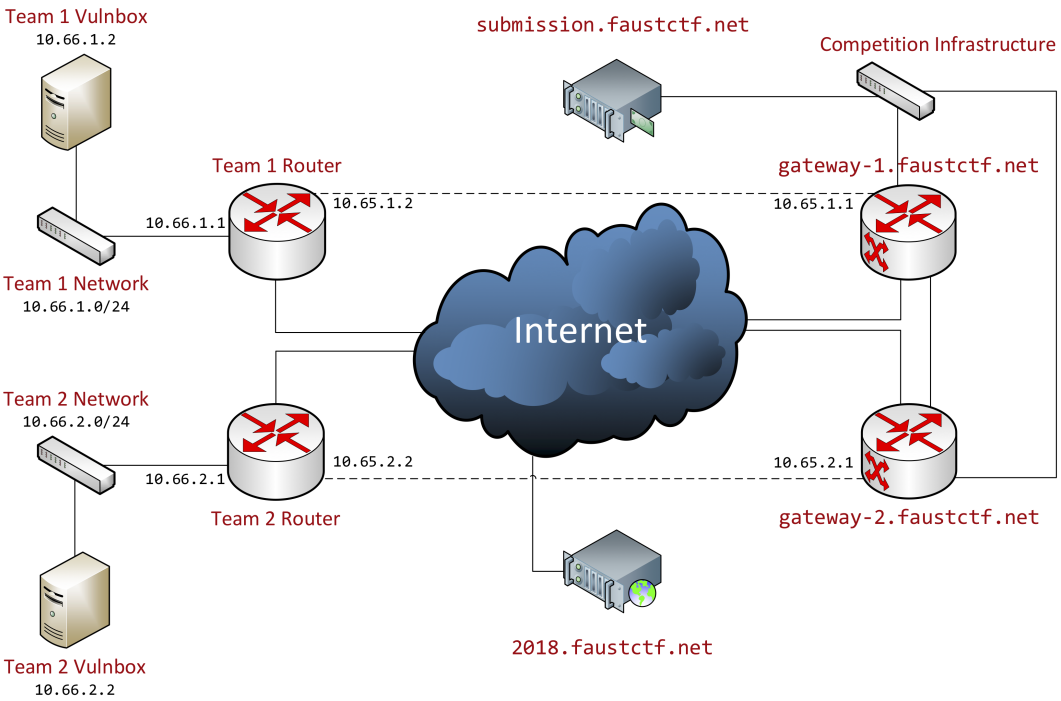

Figura 2.1: Esempio dell'architettura di rete utilizzata durante la competizione FAUST CTF 2019.

che possono fare la differenza. Gli strumenti si dividono solitamente in due categorie: gli strumenti di attacco, e quelli di difesa.
Lo strumento di attacco più rilevante è il *flag submitter/exploiter*. Il programma ha due principali funzionalità:

- lanciare gli exploit scritti dai membri del team su tutte le macchine vulnerabili dei team avversari presenti nella rete di gioco, per esfiltrare flag. Siccome le flag si rigenerano, gli exploit devono essere eseguiti ad ogni round;

- consegnare le flag esfiltrate al servizio di *submission*, che attribuisce i punti al team in base alle flag consegnate.

Lo strumento di difesa essenziale da avere durante una competizione attacco/difesa è l'analizzatore di traffico. Tutti gli attacchi rivolti verso la propria macchina vulnerabile passano per la rete; diventa quindi un'attività fondamentale analizzare il traffico di rete per individuare gli attacchi e capire come gli avversari sfruttino le vulnerabilità presenti sulla macchina vulnerabile per esfiltrare la flag.
La prossima sezione tratterà interamente questo argomento.

2.2 Analisi del traffico

Il flusso di pacchetti scambiato tra la macchina vulnerabile e la rete di gioco può essere intercettato e visualizzato in tempo reale, o registrato in file PCAP per poter essere analizzato successivamente. Lo strumento più utilizzato per registrare il traffico di rete è `tcpdump`, una utility open source da utilizzare tramite interfaccia a riga di comando che serve per catturare il traffico da interfaccia di rete e filtrarlo.
Quasi la totalità dei servizi vulnerabili messi a disposizione come challenge durante una competizione di tipo attacco/difesa utilizzano protocolli basati su TCP per interagire. Ciascun servizio può usare un protocollo applicativo diverso, anche in base alla categoria della challenge. I più frequenti sono HTTP, FTP, IMAP, SMTP o protocolli personalizzati basati su TCP.

Importanza di analizzare il traffico

Durante una competizione è importante analizzare il traffico principalmente per due motivi:

- **Ricostruire gli exploit**. Come già detto in precedenza ai servizi vulnerabili arrivano sia richieste legittime sia richieste malevole che hanno l'obiettivo di esfiltrare le flag. Le richieste malevole sfruttano una vulnerabilità o un problema nel protocollo per alterare il comportamento di una determinata operazione. Se tramite l'analisi del traffico si individua una richiesta malevola è possibile provare a replicare la sequenza di messaggi inviati dall'attaccante su altri avversari per ottenere lo stesso effetto. Questa tecnica è denominata **attack replay**.

- **Individuare le vulnerabilità**. Analizzare il traffico è anche fondamentale per individuare le vulnerabilità che vengono sfruttate dagli avversari per esfiltrare le flag. Visualizzando la sequenza di messaggi o di byte scambiati da attaccante e macchina vulnerabile è più semplice individuare i bug nei servizi e capire come l'avversario ha sfruttato le vulnerabilità per trarne un profitto.

Difficoltà nell'analisi

Il protocollo TCP permette a due host di instaurare una connessione, formata da due stream unidirezionali e indipendenti accoppiati. All'interno di una connessione vengono scambiati numerosi pacchetti di rete, che aumentano in base alla quantità di dati da trasportare e alla durata della connessione. Durante una competizione i servizi vulnerabili possono essere acceduti concorrentemente da più client, ed essendo solitamente molti i partecipanti ad una gara, ci possono essere picchi di centinaia o migliaia di connessioni attive contemporaneamente.
Il traffico generato dai bot degli avversari che tentano di esfiltrare le flag contenute nei servizi vulnerabili è elevato, e il volume di dati scambiato è consistente. Questo significa che la quantità di pacchetti scambiata tra macchina vulnerabile e rete di gioco è elevata, e le attività di analisi sono

da effettuare in condizioni difficili. L'utilizzo di Wireshark o altri strumenti che visualizzato il traffico di rete con una granularità a livello di pacchetto di rete è quindi inefficace.

Analizzando i singoli pacchetti, inoltre, non si ha contesto e si hanno soltanto informazioni parziali. All'interno del payload di un segmento TCP c'è soltanto una piccola parte della conversazione di una connessione TCP. Analizzando ed estraendo informazioni dai singoli pacchetti come fanno strumenti come Wireshark non si riesce a ricostruire i messaggi delle richieste che vengono effettuate ai servizi vulnerabili. Per poter analizzare il traffico a livello di connessione e non più di pacchetto occorre ricostruire le connessioni attraverso un procedimento denominato *TCP reassembly*, che verrà descritto nel capitolo 3.

Ricostruire le conversazioni permette di fare analisi a un livello più alto per ispezionare i messaggi scambiati tra client legittimo o malevolo e macchina vulnerabile. Permette quindi di fare analisi a livello di protocollo applicativo, e non più a livello di rete. In questo modo è possibile individuare dei pattern all'interno delle conversazioni utilizzando le espressioni regolari. Attraverso questa tecnica, ad esempio, è possibile individuare ed etichettare le connessioni in cui vengono esfiltrate le flag, perché solitamente le flag seguono regole ben precise.

Un'altra difficoltà nell'analisi è quella che le conversazioni vengano cifrate e non sia quindi permesso leggere e analizzare il contenuto delle connessioni. Un modo efficace per risolvere questo problema, che non sempre funziona, è disabilitare la cifratura nei servizi o catturare il traffico prima di cifrare i messaggi e dopo averli decifrati. Ad esempio, per un servizio che fa uso del protocollo HTTPS, se non è possibile disabilitare TLS e utilizzare la modalità HTTP in chiaro, è possibile installare un reverse proxy come `nginx`[2] per implementare la cifratura esternamente. In questo modo si può catturare il traffico tra l'endpoint del servizio e il reverse proxy per vedere i messaggi scambiati in chiaro.

[2]https://www.nginx.com/

Problematiche di strumenti esistenti

Esistono numerosi strumenti per analizzare il traffico di rete, ma la maggior parte di questi sono inadatti durante una competizione di tipo attacco/difesa. Alcuni hanno delle funzionalità utili che possono fare al caso ma che sono inefficienti e dispersive se applicate al contesto delle CTF. In questa breve sezione verranno presi in considerazione gli strumenti più adatti e verranno elencate le eventuali mancanze o problematiche.

Wireshark

Wireshark[3] è un analizzatore di protocolli di rete più diffuso e utilizzato. È uno strumento open source che permette di catturare il traffico da un'interfaccia di rete e di visualizzare in tempo reale i pacchetti che vengono scambiati.
Esiste sia l'applicazione desktop che permette di visualizzare la lista dei pacchetti analizzati attraverso un'interfaccia grafica, sia la versione a riga di comando che mostra i pacchetti scambiati a terminale, denominata `tshark`[4].
Wireshark è un ottimo strumento per analizzare la rete a livello di pacchetti, perché è efficiente, funzionale, e permette di decodificare i pacchetti di quasi la totalità dei protocolli in qualsiasi formato. Non ricostruisce automaticamente però i flussi TCP, non permette quindi di visualizzare la lista delle connessioni contenute in una registrazione di rete. Anche la ricerca di pattern complessi all'interno del contenuto delle connessioni non è presente. Inoltre, Wireshark è uno strumento da avviare e utilizzare sulla macchina locale, e l'analisi non può essere condivisa da più membri del team.

Suricata e Snort

Suricata e Snort sono due *intrusion detection system* (IDS) e *intrusion prevention system* (IPS). Sono entrambi open source e sono due degli strumenti più diffusi nell'ambito dell'analisi di rete in tempo reale e della prevenzione di minacce.

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

Nella modalità IPS i due software individuano pacchetti considerati malevoli e li scartano, impedendo che raggiungano quindi la destinazione. Nella modalità IDS ogni volta che lo strumento rileva una minaccia viene generato un avviso o viene intrapresa un'azione. Entrambi i software hanno un linguaggio per definire le regole che definiscono i filtri e i pattern con cui selezionare i pacchetti malevoli.
Entrambi gli strumenti supportano i protocolli più diffusi, tra cui TCP. Le connessioni TCP vengono quindi ricostruite durante l'analisi e viene effettuato il *pattern matching* in tempo reale per individuare determinate stringhe attraverso espressioni regolari.
Questi software però sono entrambi strumenti di backend, e non permettono né la visualizzazione dei flussi che sono stati ricostruiti, né la visualizzazione del contenuto delle connessioni.

CapAnalysis

CapAnalysis[5] è uno strumento open source che serve per analizzare grossi volumi di traffico di rete acquisiti in file PCAP. Tramite un'interfaccia web è possibile visualizzare i risultati dell'analisi e filtrare i flussi che sono stati catturati. Effettua *deep packet inspection* e supporta un vasto numero di protocolli.
CapAnalysis, inoltre, riassembla le connessioni TCP e ne estrae i metadati. È in grado di identificare quindi il numero di byte persi, per ogni direzione, e il totale di byte scambiati rimuovendo dal conteggio i pacchetti ritrasmessi.
Questo strumento ha però due grandi mancanze: non permette di visualizzare il contenuto delle conversazioni ricostruite, che è essenziale durante una competizione di tipo attacco/difesa, e non permette di identificare e ricercare le connessioni utilizzando dei pattern o delle regole.

[5]https://www.capanalysis.net/ca/

Capitolo 3

TCP Reassembly

Il protocollo TCP è il più utilizzato tra i protocolli di rete. Gli autori di [4] stimano che più del 90% del traffico di rete è TCP, e quasi la totalità dei servizi vulnerabili presenti in una competizione attacco/difesa si basa su TCP.

Il protocollo TCP permette di instaurare una connessione punto a punto affidabile tra due host collegati da una rete che può essere instabile. Con instabile si intende che i pacchetti di rete spediti da un host possono essere persi o possono giungere a destinazione in un ordine diverso dall'ordine in cui sono stati spediti. Per ovviare a questi problemi il protocollo TCP implementa dei meccanismi di ritrasmissione di pacchetti e controlli di sequenza per garantire che i messaggi scambiati all'interno di una connessione TCP arrivino a destinazione nello stesso ordine in cui sono stati spediti.

Quando si registra il traffico di rete si possono quindi incontrare pacchetti TCP duplicati o fuori ordine. A partire da una sequenza di segmenti TCP trasmessi via rete è possibile ricostruire la conversazione originale ed estrarre i messaggi scambiati tra due host attraverso un procedimento che viene definito *TCP Reassembly*. In questo breve capitolo verrà descritta la tecnica utilizzata per estrarre da un flusso di pacchetti di rete le connessioni TCP presenti.

Motivazioni della ricostruzione

Ricostruire le connessioni TCP non serve solo per poter visualizzare il contenuto dei messaggi scambiati, ma anche per altri scopi. Si possono ad esempio estrarre dei metadati utili, come la durata di una connessione o quanti byte sono stati scambiati all'interno di una connessione.

Anche per poter effettuare *pattern matching*, sia con espressioni regolari, sia con stringhe statiche, è necessario ricostruire le connessioni. Ciascun segmento TCP, infatti, contiene soltanto una piccola porzione del contenuto dei messaggi scambiati in una conversazione, quindi il *pattern matching* non può essere effettuato a livello di pacchetto, perché una occorrenza può trovarsi a cavallo tra due segmenti. Effettuare *pattern matching* a livello di pacchetto non può funzionare perché i segmenti possono essere ripetuti o non in ordine; effettuare quindi *pattern matching* sui segmenti man mano che arrivano darebbe risultati sbagliati.

Due stream differenti

Il protocollo TCP permette di instaurare una connessione punto a punto tra due host full-duplex. Questo significa che i dati possono essere trasmessi in entrambe le direzioni anche contemporaneamente. TCP implementa questa modalità creando due canali unidirezionali e indipendenti ogni volta che viene aperta una nuova connessione.

In ciascun canale i dati fluiscono in un solo verso, e i versi dei due canali sono opposti per permettere ad entrambi gli host di spedire i dati. Essendo indipendenti non ci sono meccanismi di sincronizzazione tra i due canali: dati due messaggi trasmessi nella stessa connessione ma in due canali differenti è impossibile stabilire da protocollo quale messaggio è stato trasmesso prima. Esiste però un modo approssimato per farlo, che è quello di andare a vedere il timestamp del pacchetto di rete.

Dato che i due stream all'interno di una connessione sono distinti e indipendenti, anche durante il processo di ricostruzioni i due flussi vengono estratti separatamente.

Tecniche di ricostruzione

Esistono diverse tecniche per estrarre le connessioni TCP e ricostruirle da un flusso di pacchetti di rete. Oltre a tecniche implementate via software, esistono tecniche anche implementate su hardware. Un esempio hardware-based è [5], che implementa il riassemblaggio dei segmenti sui router situati nelle dorsali di rete, che devono processare grandi quantità di pacchetti. Un altro esempio è [6] che implementa la ricostruzione degli stream TCP utilizzando hardware economico come FPGA, garantendo alte prestazioni e un basso utilizzo di memoria, oppure [7], che implementa il riassemblaggio dei segmenti TCP su switch di rete con velocità fino a un gigabit al secondo. Estrarre le connessioni dal traffico di rete può essere un'operazione delicata nei contesti in cui si utilizzi questa tecnica per rilevare o bloccare minacce. Se l'algoritmo utilizzato per riassemblare i segmenti è progettato o implementato male possono essere effettuati diversi attacchi per confondere la ricostruzione e inviare contenuti malevoli senza che questi possano essere identificati. In [8] è riportato un algoritmo resistente a questo tipo di attacchi.

3.1 Algoritmo di ricostruzione

In questa sezione è descritto un esempio di algoritmo che può essere utilizzato per l'estrazione delle connessioni TCP a partire da un flusso di segmenti. La procedura presentata dall'algoritmo deve essere eseguita ogni volta che un nuovo segmento viene catturato e deve essere processato. L'algoritmo fa quindi riferimento solamente al segmento corrente.

1. **Si ignorano i segmenti TCP vuoti**. I segmenti che non hanno le flag SYN, FIN o RST attive (e che quindi non sono utili a determinare se una connessione sta per essere creata o distrutta), e che non hanno contenuto nel campo Payload del segmento, vengono scartati e la procedura termina, in attesa di processare un nuovo segmento.

   I segmenti senza flag di controllo e senza Payload servono solitamente per confermare la ricezione del pacchetto alla controparte della connessione, ma non sono utili ai fini della ricostruzione.

2. **Si genera la chiave che identifica la connessione a cui il segmento appartiene**. La chiave è formata dalla quadrupla (indirizzo sorgente, indirizzo destinazione, porta sorgente, porta destinazione). Le informazioni per generare la chiave sono presenti all'interno del pacchetto.

   Questa tupla identifica globalmente la connessione, perché nello stesso istante non è possibile che siano attive più connessioni che abbiano le chiavi identiche. A meno di configurazioni di rete particolari si ha la certezza che i segmenti ricevuti a distanza ravvicinata che generano la stessa chiave appartengono alla stessa connessione.

   La tupla generata, inoltre, considera anche il verso della connessione. Due segmenti che fanno parte della stessa connessione ma che hanno verso opposto generano due chiavi diverse. Il processo di assemblamento quindi tratta i due stream della stessa connessione in modo indipendente.

3. **Si individua la connessione corrispondente alla chiave generata**. Il processo di ricostruzione utilizza un dizionario globale che associa per ogni chiave di connessione la connessione corrispondente. Se all'interno del dizionario non è presente una connessione con la chiave generata viene istanziata una nuova struttura dati che fa riferimento alla connessione corrente e viene inserita nel dizionario. Se il dizionario contiene la chiave generata dalla connessione corrente si ottiene il riferimento alla struttura dati della connessione corrente.

4. **Si confronta il numero di sequenza del segmento con quello della ricostruzione**. Nella struttura dati di ciascuna connessione è presente una variabile che serve per memorizzare il successivo numero di sequenza valido. Questo contatore serve per controllare se i segmenti che arrivano sono in ordine e non sono duplicati. All'inizio la variabile è impostata a `invalidSequence`.

   - Se il contatore del successivo numero di sequenza da accettare è `invalidSequence` e il segmento ha la flag SYN attiva il segmento

è arrivato in ordine perché si tratta del primo pacchetto che inizializza la connessione. Si estrae quindi il Payload del segmento e lo si aggiunge al buffer che contiene il contenuto della connessione. Si imposta inoltre il contatore del successivo numero di sequenza al numero di sequenza del segmento + la lunghezza del Payload del segmento + 1.

- Se il contatore del successivo numero di sequenza da accettare è `invalidSequence` e il segmento non ha la flag SYN attiva si tratta di un pacchetto che non è arrivato in ordine e che va accantonato finché non si trova il segmento che inizializza la connessione.

  I segmenti accantonati vengono mantenuti in una *doubly linked list*, ovvero una lista i cui nodi hanno il riferimento sia al nodo precedente, sia a quello successivo. I segmenti vengono memorizzati all'interno della lista in ordine rispetto al numero di sequenza che li accompagna. Il fatto che si utilizzi una lista doppiamente collegata è per una questione di ottimizzazione, perché è più efficiente scorrere la lista per individuare i due nodi tra cui inserire il nuovo segmento. È più facile inoltre modificare i puntatori all'elemento precedente e all'elemento successivo per inserire in mezzo il nodo del nuovo segmento. In figura 3.1 è riportato uno schema di esempio.

  In questo caso non si incrementa il contatore che indica la successiva sequenza del segmento da accettare, perché il pacchetto arrivato era fuori ordine e finché non si processa il segmento col numero di sequenza giusto non si può avanzare.

- Se la differenza tra il numero di sequenza del segmento corrente e il contatore del successivo numero di sequenza atteso è maggiore di zero si sta processando un segmento fuori ordine che deve essere accantonato allo stesso modo del punto precedente, oppure sono stati persi dei segmenti durante la trasmissione. Anche in questo caso il contatore del numero di sequenza atteso non deve essere incrementato. In figura 3.2 è mostrato uno schema relativo

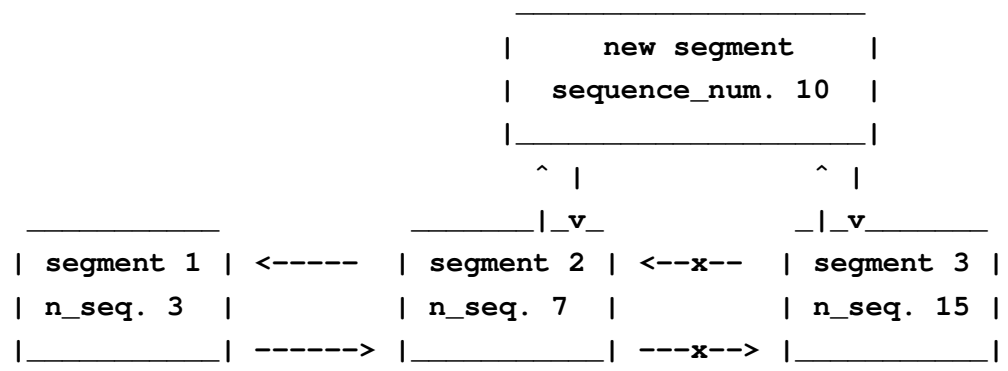

Figura 3.1: Schema che mostra la struttura dati *doubly linked list* utilizzata per memorizzare i segmenti fuori ordine. Quando un nuovo segmento deve essere aggiunto alla lista deve essere inserito tra due segmenti che hanno rispettivamente numero di sequenza inferiore e superiore a quello del segmento da aggiungere.

a questo caso.

- Se la differenza tra il numero di sequenza del segmento corrente e il contatore del successivo numero di sequenza atteso è minore o uguale a zero si sta processando un segmento contiguo ai segmenti precedentemente memorizzati. Il contenuto del Payload del segmento deve quindi essere accodato al buffer della connessione.

  Il segmento corrente il cui contenuto del Payload deve essere aggiunto al buffer può però essere un segmento duplicato, e parte del suo contenuto può già essere presente nel buffer. Per stabilire quale fetta del Payload deve essere accodato al buffer occorre calcolare la differenza tra il numero di sequenza del segmento corrente e il contatore del successivo numero di sequenza atteso.

  - Se il risultato della differenza è uguale o inferiore a zero tutti i byte contenuti nel Payload del segmento devono essere accodati al buffer, perché non si tratta di byte duplicati. Lo schema riportato in figura 3.3 mostra questo caso.

  - Se la lunghezza del Payload è inferiore al risultato della differenza nessun byte deve essere aggiunto al buffer, perché nel

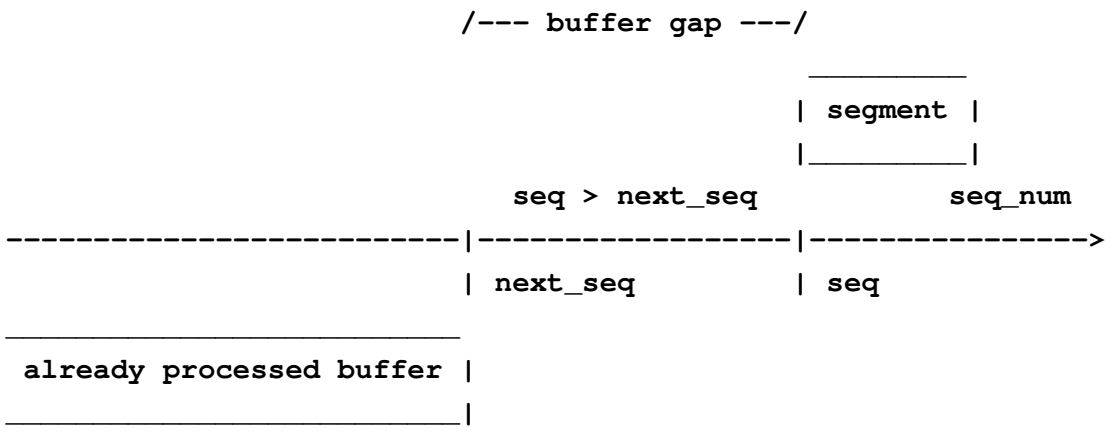

Figura 3.2: Schema che mostra l'arrivo di un segmento fuori ordine, che ha un numero di sequenza maggiore al successivo numero di sequenza atteso. Il segmento deve quindi essere accantonato.

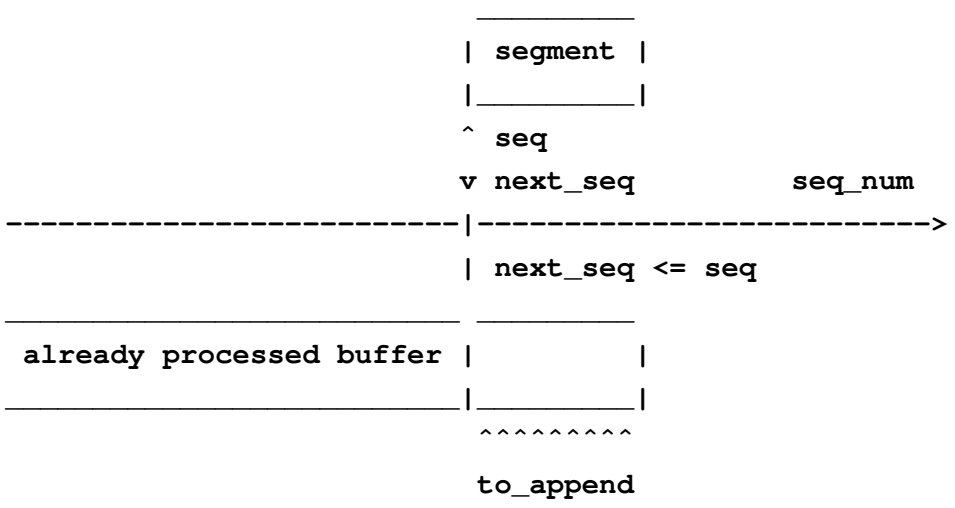

Figura 3.3: Schema che rappresenta l'arrivo di un segmento con numero di sequenza uguale o maggiore al numero di sequenza atteso.

Payload sono contenuti soltanto byte duplicati già presenti nel buffer. In figura 3.4 è mostrato questo caso.

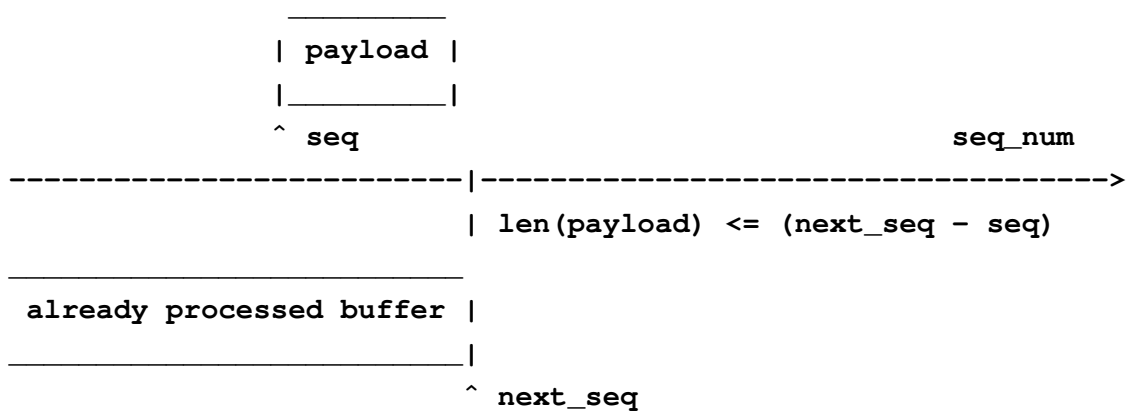

Figura 3.4: Schema che mostra l'arrivo di un segmento già trasmesso il cui contenuto è già stato inserito nel buffer di ricezione che contiene il contenuto dello stream già processato.

– Altrimenti si accoda al buffer soltanto la fetta del Payload del segmento che inizia ad indice pari al risultato della differenza. La fetta del Payload del segmento che va da indice zero fino ad indice pari al risultato della differenza è già presente nel buffer che contiene il contenuto della connessione ricostruito. In figura 3.5 è riportato lo schema che mostra il caso corrente.

Infine, occorre aggiungere al contatore del successivo numero di sequenza atteso con il numero di byte inseriti in coda al buffer della connessione.

5. **Se nella lista dei segmenti accantonati ci sono segmenti con numero di sequenza contiguo al contatore del successivo numero di sequenza** occorre rimuovere i segmenti contigui dalla lista e aggiungere il contenuto del Payload nel buffer della connessione. Dalla lista devono essere rimossi tutti i segmenti contigui utilizzando il seguente algoritmo iterativo:

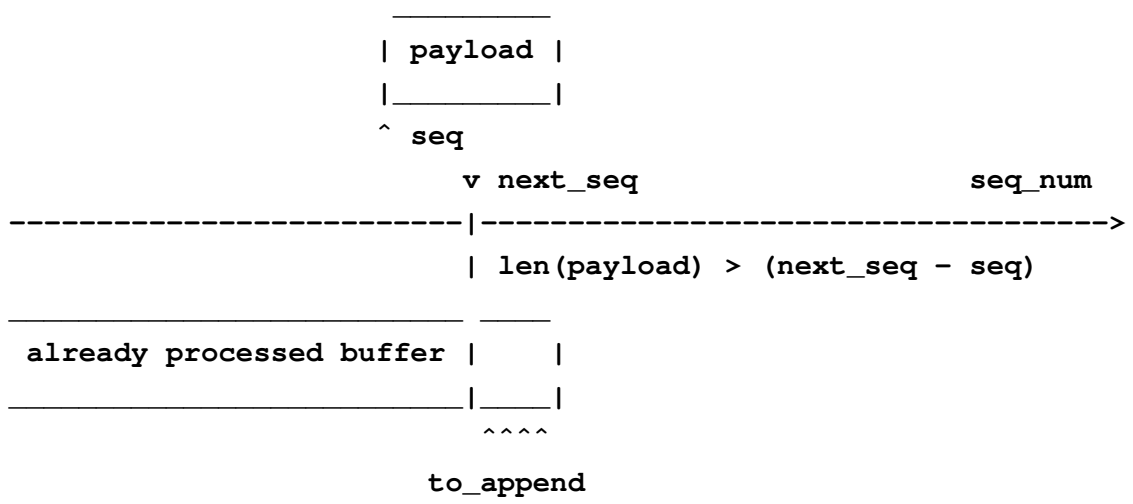

Figura 3.5: Schema che mostra che soltanto parte del contenuto del segmento deve essere deve essere aggiunto al buffer del contenuto già processato.

- se si individua un segmento contiguo questo deve essere rimosso dalla lista e il contenuto del suo Payload deve essere accodato al buffer della connessione;

- se la lista dei segmenti accantonati è vuota o il successivo segmento non è contiguo questo procedimento viene interrotto.

6. **Se il segmento corrente ha la flag FIN o RST attiva** e viene il segmento non viene accodato alla lista dei segmenti accantonati, oppure se dalla lista dei segmenti accantonati l'ultimo segmento che si estrae ha la flag FIN o RST attiva, si liberano le risorse e si può considerare la connessione corrente come terminata. La connessione può quindi essere rimossa dal dizionario globale che associa ad ogni chiave la relativa struttura dati della connessione. In questo modo se in futuro un segmento genera la stessa chiave viene creata una nuova connessione.

Pseudocodice dell'algoritmo

Di seguito è riportato lo pseudocodice dell'algoritmo descritto nella sezione precedente.

```
connections_dict = new(Dict)

func AssembleSegment(segment) {

// Punto 1: ignora i pacchetti TCP vuoti

if not_syn(segment) and not_fin(segment) and not_rst(segment) and
    len(payload) == 0:
    return


// Punto 2: viene generata la chiave di identificazione della connessione

key = <source_address(segment), destination_address(segment),
       source_port(segment), destination_port(segment)>


// Punto 3: si ottiene o si crea l'oggetto relativo alla connessione

conn = nil
if connections_dict[key] != nil:
    conn = connections_dict[key]
elif not_syn(segment) and len(payload) == 0:
    return // ignora i segmenti vuoti in connessioni chiuse
else:
    conn = new(Connection)
    conn.nextSequence = invalidSequence
    conn.buffer = make(ByteArray)
    conn.segmentsAside = make(DoublyLinkedList)


// Punto 4: confronta il numero di sequenza del segmento con il
//          contatore del successivo numero di sequenza atteso
```

```
if conn.nextSequence == invalidSequence:
    if is_syn(segment):
        // inserisce il contenuto del segmento nel buffer
        conn.buffer += payload
        conn.nextSequence = sequence(segment) + len(payload) + 1
    else:
        // inserisce nella doubly-linked list il segmento fuori ordine
        // da accantonare, utilizzando come metrica di ordinamento il
        // numero di sequenza
        ordered_insert(conn.segmentsAside, segment,
                       lambda s1, s2: sequence(s1) <= sequence(s2))
elif difference(conn.nextSequence, sequence(segment)) > 0:
    // c'è un vuoto tra il segmento attuale e i segmenti già aggiunti
    // al buffer: occorre accantonare il segmento
    ordered_insert(conn.segmentsAside, segment,
                   lambda s1, s2: sequence(s1) <= sequence(s2))
else:
    // il segmento è contiguo all'ultimo segmento aggiunto al buffer
    if conn.nextSequence <= sequence(segment):
        conn.buffer += payload
        conn.nextSequence += len(payload)
    elif len(payload) <= difference(conn.nextSequence, sequence(segment)):
        pass
    else:
        span = difference(conn.nextSequence, sequence(segment))
        conn.buffer += payload[span:]
        conn.nextSequence += len(payload) - span


// Punto 5: si aggiungono al buffer tutti i segmenti che sono stati
//          precedentemente accantonati che sono contigui all'ultimo
//          segmento inserito nel buffer

last_segment = segment
```

```
while conn.segmentsAside.first != nil and
    difference(conn.nextSequence, sequence(conn.segmentsAside.first)) <= 0:
    last_segment = conn.segmentsAside.first
    remove_first(conn.segmentsAside)

    if conn.nextSequence <= sequence(last_segment):
        conn.buffer += last_segment.payload
        conn.nextSequence += len(last_segment.payload)
    elif len(last_segment.payload) <= difference(
        conn.nextSequence, sequence(last_segment.payload)):
        pass
    else:
        span = difference(conn.nextSequence, sequence(last_segment))
        conn.buffer += last_segment.payload[span:]
        conn.nextSequence += len(last_segment.payload) - span


// Punto 6: controlla se l'ultimo segmento processato è un segmento
//          di chiusura; in tal caso segnala che la connessione è
//          stata estratta completamente

if is_fin(last_segment) or is_rst(last_segment):
    connections_dict[key] = nil
    complete(conn)
}
```

Capitolo 4

Architettura e implementazione di Caronte

Come già descritto nel capitolo precedente, la necessità di realizzare questo strumento nasce dal fatto che durante le competizioni di tipo attacco/difesa è fondamentale analizzare il traffico di rete per individuare le vulnerabilità sfruttate dagli avversari. Quel che mi ha spinto a realizzare questo progetto è il fatto che esistono innumerevoli strumenti per analizzare il traffico di rete, ma nessuno di questi è in grado di adattarsi in modo completo al contesto delle competizioni Capture The Flag. In questo capitolo andrò a descrivere il processo di realizzazione di Caronte, partendo dalla descrizione dei requisiti che ho raccolto per la definizione degli obiettivi da raggiungere fino alla validazione e alla distribuzione.

4.1 Definizione dei requisiti

Caronte non è il primo strumento di analisi di rete che ho realizzato. In passato ho scritto altri script e strumenti per analizzare il traffico e per ricostruire le connessioni, per poter visualizzare quanto scambiato da client e server in un modo a più alto livello rispetto alla semplice visualizzazione di pacchetti. Un progetto degno di nota che ho realizzato prima di Caronte è `pyranoic`[1], uno strumento a riga di comando che ho sviluppato per la

[1] https://github.com/eciavatta/pyranoic

CyberChallenge[2], il programma di formazione nazionale sulla cybersicurezza che coinvolge le maggiori università italiane.
Ho deciso di realizzare Caronte per correggere gli errori progettuali di `pyranoic` e per inserire come requisiti aspetti che non avevo considerato quando ho iniziato a elaborare il vecchio progetto. Molte limitazioni di `pyranoic` sono apparse evidenti durante l'utilizzo, altri suggerimenti e consigli di funzionalità mi sono arrivati dai miei compagni di team.
Di seguito descriverò quindi i requisiti e gli obiettivi che mi sono posto per la realizzazione di Caronte, correlati con le motivazioni che mi hanno spinto ad effettuare tali scelte.

Requisiti funzionali

- **Lettura e cattura dei PCAP**. Lo strumento deve analizzare il traffico di rete; deve quindi essere in grado di leggere i pacchetti che vengono trasmetti da e verso un'interfaccia di rete. Esistono due modalità per caricare il traffico di rete all'interno dello strumento:

  - modalità online, attraverso la cattura del traffico di un'interfaccia. I pacchetti trasmessi vengono immediatamente catturati e inviati allo strumento per essere processati. Ha come vantaggio il fatto di non avere ritardo: i pacchetti vengono processati dallo strumento immediatamente. Ha come svantaggio il fatto di dover spostare l'applicazione in una macchina che inoltra i pacchetti da destinare ai servizi da analizzare, aggiungendo ritardi di processamento e problemi legati alla sicurezza.

  - modalità offline, attraverso la lettura di un file PCAP, ovvero di un file che contiene un insieme di pacchetti precedentemente registrati. Ha come vantaggio il fatto di poter spostare l'analizzatore in una macchina a piacimento, situata anche in un'altra rete rispetto a quella in cui sono contenuti i servizi. In questo modo è possibile dedicare una macchina diversa con risorse dedicate. Lo

[2]https://cyberchallenge.it/

svantaggio principale è quello di non riuscire a raggiungere un'analisi in tempo reale: i file PCAP vengono creati ed inviati ad intervalli di tempo regolari.

- **Solo traffico TCP**. Nelle competizioni di tipo attacco/difesa la maggior parte dei servizi, se non tutti, utilizzano il protocollo TCP per comunicare. Il protocollo applicativo più utilizzato è HTTP, ma capitano spesso altri tipi di protocolli come FTP, MQTT o protocolli personalizzati. Raramente si hanno a disposizione servizi che non si basano su TCP. Questi protocolli, come UDP, restano esclusi dall'ambito di questo progetto.

  Qualsiasi conversazione che si basa sul protocollo TCP può essere ricostruita: i singoli pacchetti possono essere riassemblati e riordinati per estrarre il contenuto della connessione. Lo scopo dello strumento è quindi quello di estrarre le connessioni per poter effettuare un'analisi ad un livello più alto rispetto ad un'analisi a livello di pacchetto. L'entità minima visualizzabile su Caronte è quindi la connessione.

  Occorre tenere in considerazione che il protocollo TCP non prevede un'unica connessione bidirezionale, bensì utilizza due connessioni monodirezionali separate. È compito del riassemblatore quindi individuare le distinte connessioni che appartengono ad un'unica conversazione e unirle o collegarle. D'ora in avanti col termine connessione ci si riferisce ad entrambi i versi di una conversazione TCP.

- **Regole per l'individuazione di connessioni**. È necessario un meccanismo per individuare in un insieme di connessioni quelle che soddisfano un determinato criterio. Il criterio può essere scelto sia in base alle proprietà della connessione sia in base al contenuto della connessione. Viene definita regola l'insieme dei criteri per selezionare le connessioni con le proprietà ricercate.

  Le proprietà delle connessioni sono informazioni che devono essere generate durante la fase di riassemblamento. Le principali proprietà dal-

le quali deve essere possibile definire un criterio di selezione sono le seguenti:

  - *Indirizzo sorgente*: indirizzo IP del client
  - *Indirizzo destinazione*: indirizzo IP del server
  - *Porta sorgente*: numero di porta del client
  - *Porta destinazione*: numero di porta del servizio
  - *Data inizio*: timestamp relativo al primo pacchetto SYN
  - *Data fine*: timestamp relativo all'ultimo pacchetto FIN o RST
  - *Durata connessione*: differenza tra data di fine e inizio
  - *Dimensione connessione*: quantità di byte scambiata

  La ricerca all'interno del contenuto della connessione deve essere possibile in due modalità: per stringa, ricercando un'occorrenza esatta di testo o di binario, o per espressione regolare.

  Un esempio di regola è quella per individuare le connessioni che esfiltrano le flag: occorre impostare come indirizzo sorgente l'indirizzo della macchina vulnerabile e scegliere come metodo di ricerca quello per espressione regolare, specificando il formato della flag. Impostando invece come indirizzo di destinazione l'indirizzo della macchina vulnerabile e lasciando la ricerca per contenuto invariata, si tracciano le connessioni dei master che inseriscono le flag all'interno dei servizi.

- **API e interfaccia web**. Deve essere possibile interagire con lo strumento da remoto. Deve essere inoltre possibile interagire con lo strumento tramite script e programmi automatizzati per ottenere le informazioni processate dallo strumento tramite opportune API. Tutte le funzionalità offerte tramite API devono poter essere accedute anche tramite interfaccia web.

Si è scelto di inserire l'interfaccia web come requisito perché è la modalità più immediata per le persone che fanno parte di un team durante una competizione, per ottenere tutte le informazioni che necessitano sul traffico di rete. L'interfaccia web permette di interagire con lo strumento da remoto senza dover installare alcun tipo di client e di utilizzare più istanze con visualizzazioni diverse.

Requisiti non funzionali

- **Capacità di processamento maggiore a traffico generato**. Sia nella modalità online (con cattura del traffico), sia nella modalità offline (con lettura di PCAP), è necessario che lo strumento riesca a processare più pacchetti di quelli che riceve in input. In altre parole, è necessario che la velocità con cui lo strumento ricostruisce le connessioni ed estrae i metadati sia superiore alla velocità di produzione dei pacchetti di rete.

- **Reattività, efficienza, e resistenza agli errori**. Lo strumento deve essere utilizzato anche in contesti competitivi, è necessario quindi che sia rapido, reattivo, e che non sia un impedimento e che non sottragga del tempo inutilmente agli utenti che lo utilizzano. Deve essere inoltre resistente agli errori per fare in modo che questi non possano compromettere l'esperienza utente o corrompere le funzionalità dello strumento.

Requisiti implementativi

- **Gestione efficiente della memoria**. Il volume di dati generato durante una competizione di attacco/difesa è elevato. Per poter processare tale quantità di informazioni è necessario utilizzare un linguaggio con un'efficiente gestione della memoria e che non impieghi la maggior parte del tempo di processamento nella creazione e nella distruzione di oggetti in memoria. Solitamente la memoria a disposizione dello strumento è condivisa con altri processi presenti all'interno della macchina, è bene quindi usarla in modo efficiente.

- **Facilità di aggiunta di nuovi protocolli**. Esistono infiniti protocolli che utilizzano TCP per instaurare una comunicazione tra due macchine. Ciascun protocollo utilizza una propria sintassi e una propria semantica, e da ciascun protocollo possono essere estratte informazioni in modo strutturato. Per il protocollo HTTP, ad esempio, è possibile estrarre informazioni quali l'URL della richiesta che viene effettuata, il codice di risposta del server, i campi presenti nell'header. Queste informazioni strutturate possono essere utilizzate per definire regole più dettagliate e più potenti.

  Al primo rilascio dello strumento soltanto pochi moduli per estrarre informazioni da protocolli verranno implementati; il progetto deve quindi essere organizzato per poter permettere di aggiungere facilmente nuovi moduli per estrarre informazioni da nuovi protocolli.

- **Installazione facilitata e automatizzata**. Nelle competizioni di attacco/difesa ciascun team crea la propria infrastruttura utilizzando solitamente strumenti per automatizzare le procedure di installazione e configurazione come `Ansible`[3]. Caronte deve quindi poter essere configurato e installato facilmente utilizzando questi strumenti. Siccome molti parametri di configurazione variano da competizione a competizione, è necessario che questi possano essere modificati dopo l'installazione dello strumento. Deve quindi essere possibile modificare la configurazione del programma dinamicamente da remoto e non solo tramite file di configurazione presenti all'interno della macchina che esegue il processo.

Requisiti utente

- **Condivisione fra più utenti**. Lo strumento deve poter essere utilizzato da più utenti contemporaneamente. Tutte le funzionalità dello strumento devono essere accessibili da remoto, ovvero da una macchina diversa da quella in cui è attivo il processo dello strumento. Deve essere

[3] https://github.com/ansible/ansible

quindi possibile caricare file PCAP da remoto, modificare le impostazioni, visualizzare i risultati dell'analisi, essere notificati quando delle connessioni corrispondono a determinate regole.

4.2 Scelte progettuali

Partendo dai requisiti definiti nella sezione precedente verranno riportate le scelte di progettazione che sono state seguite per la realizzazione di Caronte. Come già descritto parte di queste scelte derivano da errori e limitazioni di software già esistenti.

- **Scartare pacchetti non TCP**. Da requisiti lo strumento deve essere in grado di processare e analizzare soltanto pacchetti TCP. Tutti i pacchetti di altri protocolli possono quindi essere scartati. È necessario quindi anteporre un filtro prima della ricostruzione delle connessioni per selezionare soltanto pacchetti TCP.

- **Architettura REST**. Lo strumento deve poter essere controllato da remoto e devono essere esposte le API per poter automatizzare le operazioni attraverso script. Anche l'interfaccia web, ovvero il frontend, deve comunicare con il backend tramite un'interfaccia. Si è quindi deciso di realizzare un'unica API utilizzando l'architettura REST, per permettere sia al frontend sia a programmi esterni di comunicare con il backend utilizzando un'interfaccia standard.

- **Database non relazionale**. Le connessioni dopo essere state ricostruite devono essere salvate in una struttura dati per essere indicizzate e per poter essere cercate e ottenute immediatamente. La struttura dati da utilizzare deve essere ottimizzata in scrittura, perché la quantità di informazioni generata durante il processo di estrazione e ricostruzione delle connessioni è elevata. Deve poter essere ottimizzato anche in lettura, perché deve essere possibile ottenere in tempi veloci tutte le connessioni che rispettano determinati criteri.

Nel database occorre memorizzare, oltre alle informazioni in comune fra tutti i tipi di connessione, anche altri metadati che variano da protocollo a protocollo. I metadati di ciascuna connessione sono diversi e non seguono uno schema ben definito. Occorre quindi che la struttura dati abbia uno schema elastico per poter inserire record con dati strutturati con uno schema non definito a priori.

Si è scelto quindi un database management system di tipo non relazionale, perché è necessaria una struttura dati di tipo *schemaless* e perché ciascun record è indipendente e non ha relazioni con gli altri record.

- **Regole processate dopo ricostruzione**. Le regole possono essere definite dagli utenti in qualsiasi momento. Appena una nuova connessione viene ricostruita si controllano tutte le regole definite nei confronti della connessione, per verificare se una o più di esse innescano qualche reazione. Si è deciso di controllare le regole immediatamente dopo la ricostruzione per lanciare avvisi o notifiche nel caso di corrispondenza, e per salvare nel database per ciascuna connessione processata le regole innescate.

  Salvando le informazioni sulle regole corrisposte nei metadati di ciascuna connessione diventa quindi possibile ricercare nel database tutte le connessioni che soddisfano determinate regole in breve tempo, senza dover controllare le regole rispetto a ogni connessione presente nel database. In questo modo si risparmiano grandi quantità di tempo e di risorse. Un problema che sorge utilizzando questa modalità consiste nel fatto che quando si aggiunge una nuova regola questa viene confrontata solamente con le nuove connessioni che verranno processate, ma non con quelle già presenti nel database.

  Una soluzione a questo problema potrebbe essere processare le regole retroattivamente estraendo anche le connessioni presenti nel database. Un'altra soluzione, che sarà quella poi adottata, è quella di inserire un meccanismo per poter ricercare manualmente nel database delle connessioni estratte quelle che contengono una determinata stringa testuale o espressione regolare.

- **Servizi individuati da numero porta**. Durante una competizione un servizio rappresenta uno o più processi in esecuzione sulla macchina vulnerabile ed espone un'interfaccia per comunicare con l'esterno tramite un protocollo. Siccome Caronte gestisce solamente protocolli basati su TCP, ogni servizio è associato ad una o più porte TCP. L'associazione porta-servizio deve poter essere effettuata dopo l'installazione dello strumento e in qualsiasi momento durante la competizione, perché i servizi possono essere aggiunti, rimossi e modificati nel tempo.

  L'individuazione dei servizi serve per diversi motivi: per filtrare tutte le connessioni di uno stesso servizio, per applicare regole soltanto alle connessioni relative ad un determinato servizio, per visualizzare le statistiche di un singolo servizio. Potrebbe essere anche utile escludere e non processare un servizio perché produce un volume di dati troppo elevato o perché non è interessante da analizzare.

- **Aggregazione delle statistiche**. Le statistiche non sono solo uno strumento per osservare l'andamento generale del programma, ma rivelano informazioni sullo stato dei servizi. Un aumento di connessioni verso un determinato servizio può ad esempio significare l'inizio di un attacco, l'aumento del numero di flag esfiltrate, la scoperta da parte dell'avversario di una nuova vulnerabilità. Le statistiche sono quindi importanti, ed è necessario che siano aggiornate.

  Calcolare le statistiche ogni volta che vengono richieste è un'operazione dispendiosa, perché occorre aggregare informazioni per ciascuna connessione presente in un intervallo di tempo. Occorre quindi mantenere una struttura dati con le ultime statistiche e aggiornarle periodicamente quando nuove connessioni vengono processate. Anche le statistiche dovrebbero poter essere accedute tramite API, per permettere a software di monitoring come `Prometheus`[4] di ottenere informazioni per essere poi visualizzate.

[4]https://github.com/prometheus/prometheus

4.3 Componenti principali del sistema

Lo strumento è composto da due macro moduli: il backend, che contiene la logica per processare i pacchetti e ricostruire le connessioni, la logica dell'analisi e di reportistica, ed espone le API per interagire con lo strumento, e il frontend, che fornisce l'interfaccia grafica per interagire con lo strumento.
In questa fase di progetto viene tenuto in considerazione soltanto il modulo del backend, e verranno descritti i componenti individuati per implementare la logica.

Importazione PCAP e gestione sessioni

È il componente che si occupa di leggere il traffico di rete e avviare la procedura per la ricostruzione delle connessioni. Ha più funzionalità, descritte di seguito.

- **Effettuare il parsing dei pacchetti**. Questa operazione serve per leggere un file PCAP da disco ed estrarre i pacchetti contenuti in esso. Il componente è anche predisposto per la cattura in tempo reale dei pacchetti da un'interfaccia di rete, ma questa funzionalità non verrà implementata inizialmente.

- **Filtrare i pacchetti TCP**. Come specificato nelle scelte progettuali che sono state compiute, lo strumento processa solo traffico TCP. Tutti i pacchetti di altri protocolli vengono quindi scartati.

- **Aggiornare le statistiche**. Lo strumento tiene traccia di quanti pacchetti sono stati processati, quanti pacchetti sono stati scartati e quanti sono invalidi. Vengono memorizzate anche le statistiche per servizio, ovvero quanti pacchetti vengono processati per ogni porta di destinazione TCP.

- **Inviare i pacchetti TCP al componente che ricostruisce le connessioni**. Ogni pacchetto viene associato al timestamp relativo alla sua ricezione e inviato al componente che estrae le connessioni a partire dai

pacchetti grezzi. È importante associare a ciascun pacchetto il timestamp di ricezione perché in questo modo è possibile ricostruire l'ordine con cui due host si scambiano i messaggi, come descritto nel Capitolo 3. Inoltre, attraverso il timestamp è possibile stabilire la durata di una connessione.

- **Gestire le sessioni**. Ogni volta che un file PCAP deve essere processato viene generata una nuova sessione. Ogni sessione è identificata da una chiave univoca che rappresenta il digest risultato della funzione sha256 del file PCAP. Questo per impedire che un PCAP venga analizzato due volte, andando a confondere il componente che estrae le connessioni.

  Una sessione viene creata automaticamente quando viene impartito il comando per processare un nuovo PCAP. Una sessione può essere cancellata se non sono ancora stati processati tutti i pacchetti relativi a quella determinata sessione. Possono essere ottenute inoltre tutte le sessioni in corso che sono state completate. Nel diagramma in figura 4.1 è mostrata la struttura dati che rappresenta una sessione.

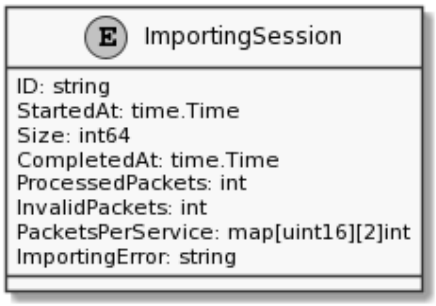

Figura 4.1: Struttura dati che rappresenta una sessione.

- **Download dei PCAP processati**. A volte può essere comodo analizzare con altri strumenti i file PCAP, per controllare eventuali anomalie od ottenere dettagli aggiuntivi. Questo componente permette di ottenere i file PCAP precedentemente processati dallo strumento, che vengono mantenuti sul filesystem indicizzati per hash.

PCAP			
Gestione e analisi dei file PCAP			
METODO	URL	DESCRIZIONE	PARAMETRI
POST	`/api/pcap/upload`	effettua l'upload di un PCAP e inizia a processarlo	**file**: il PCAP da caricare **flush_all**: chiude tutte le connessioni che non sono state terminate
	`/api/pcap/file`	processa un PCAP presente sul filesystem dello strumento	**file**: il percorso del PCAP relativo al filesystem locale da processare **flush_all**: chiude tutte le connessioni che non sono state terminate **delete_original_file**: rimuove il PCAP dal filesystem dopo essere stato processato

Tabella 4.1: API relative alla risorsa **pcap**

Nella tabella 4.1 sono riportate le API per le due operazioni di caricamento PCAP, attraverso l'upload remoto e specificando il percorso di un file locale. Nella tabella 4.2 sono riportate le API per la gestione delle sessioni e per il download dei PCAP.

Gestione regole

È il componente che si occupa dell'inserimento, aggiornamento e ottenimento delle regole, oltre che di verificare se una connessione corrisponde a una delle regole presenti nel database di regole. Di seguito sono descritte in dettaglio

SESSIONS			
Gestione delle sessioni			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/sessions`	ottiene la lista di tutte le sessioni, in fase di processamento e già completate	
	`/api/sessions/{id}`	ottiene la sessione specificata	**id**: l'id della sessione da ottenere
	`/api/sessions/{id}/download`	scarica il file PCAP associato alla sessione	**id**: l'id della sessione da cui scaricare il PCAP
DELETE	`/api/sessions/{id}`	sospende una sessione se è in fase di processamento	**id**: l'id della sessione da sospendere

Tabella 4.2: API relative alla risorsa **sessions**.

tutte le funzionalità di questo componente.

- **Ottenere regole esistenti**. Questa operazione serve per ottenere la lista delle regole presenti nel database al fine di essere visualizzate. È anche possibile ottenere una singola regola fornendo il suo id, un codice identificativo univoco per ogni regola.

- **Inserire nuove regole**. Dopo che una nuova regola viene aggiunta il componente ricompila il database delle regole per includere le nuove specifiche di *matching*. Nella figura 4.2 è mostrato il diagramma della struttura dati che rappresenta una regola. Ciascuna regola ha delle proprietà di base per l'identificazione e la visualizzazione; inoltre è associata ad un filtro e ad un insieme di pattern.

  Il filtro serve per selezionare le connessioni a cui applicare la regola. Le proprietà del filtro corrispondono alle proprietà delle connessioni, ovvero alle informazioni che vengono generate in fase di ricostruzione e assemblamento.

Una regola deve contenere anche uno o più pattern. Un pattern serve per specificare i criteri con cui una connessione deve essere corrisposta. L'unica tipologia di pattern implementato è il pattern di ricerca, che serve per individuare attraverso l'utilizzo di espressioni regolari se all'interno di una connessione sono presenti una o più occorrenze. È possibile specificare facoltativamente una direzione per restringere la ricerca in solo una delle due direzioni della connessione TCP.

Le regole prima di essere inserite vengono verificate, per identificare duplicati o problemi di compatibilità. I pattern delle regole aggiunte vengono inoltre compilati in una struttura dati comune a tutti i pattern di tutte le regole, per ottimizzare la ricerca.

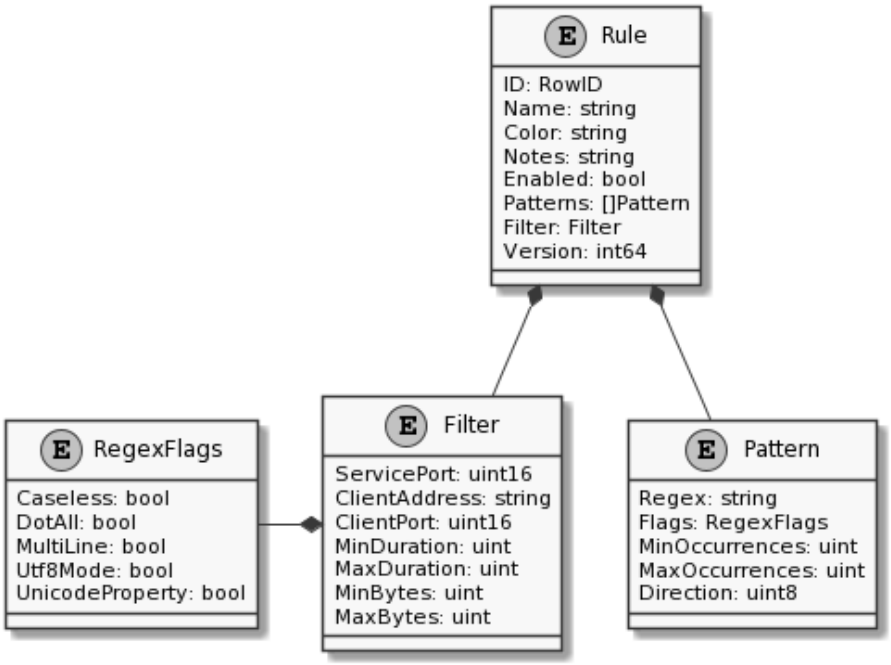

Figura 4.2: Struttura dati che rappresenta una regola.

- **Aggiornare regole esistenti**. L'operazione di aggiornamento è delicata, perché le regole vengono verificate solamente quando una nuova connessione è stata ricostruita. Modificare il filtro o uno o più pattern di una regola significherebbe rieseguire la procedura di *matching* su

tutte le connessioni già processate e presenti nel database, ma essendo questa una procedura costosa in termini di risorse viene evitata. La scelta che è stata effettuata in questo caso è quella di permettere di poter disabilitare una regola errata o non più utile; in tal caso l'utente potrà crearne una nuova in sostituzione di quella obsoleta o errata.

L'aggiornamento di una regola permette quindi di abilitarla o disabilitarla, ma anche di aggiornare le proprietà di pura visualizzazione come il nome, il colore (visualizzato sul frontend) e le note.

- **Verificare la corrispondenza delle regole rispetto ad una connessione**. Oltre alla gestione delle regole, questo componente ha il compito di verificare quali regole fanno *match* quando sono confrontate con una connessione. La prima operazione che viene effettuata è quella di verificare se le informazioni della connessione corrispondono alle proprietà dei filtri di ciascuna regola. Dopo che una regola viene ricostruita dal componente che gestisce le connessioni, tutti i pattern presenti nel database dei pattern (compilato da questo componente) vengono processati attraverso un motore di espressioni regolari sul contenuto della connessione, alla ricerca di occorrenze. Il passo successivo nel processo di verifica di corrispondenza delle regole è quindi quello di individuare a quali regole appartengono i pattern sui quali è stato fatto *match*.

  Se una regola ha i filtri compatibili con la connessione che si sta analizzando, e tutti i pattern relativi alla regola riportano un numero di occorrenze compatibile con le specifiche del pattern, la connessione viene marcata e viene associata alle regole che sono state individuate.

L'insieme delle regole inserite dagli utenti viene mantenuto in memoria per una questione di efficienza, e memorizzato sul database per rendere le regole persistenti dopo un eventuale riavvio dello strumento. Le regole, quindi, devono essere sincronizzate tra memoria e database. Ogni volta che una regola viene aggiunta o aggiornata, le modifiche vengono riflesse sia in memoria sia sul database. All'avvio dello strumento le regole presenti sul database vengono caricate in memoria: in questo modo quando un utente o un altro

RULES			
Inserire, rimuovere e aggiornare regole			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/rules`	ottiene la lista di `Rule` definite nel sistema	
	`/api/rules/{id}`	ottiene la `Rule` specificata	**id**: l'id della regola da ottenere
POST	`/api/rules`	inserisce una nuova regola al database delle regole	la struttura `Rule` definita in figura 4.2
PUT	`/api/rules/{id}`	modifica una regola esistente	la struttura `Rule` definita in figura 4.2

Tabella 4.3: API relative alla risorsa **rules**.

componente richiede la lista delle regole inserite è possibile ottenere l'insieme delle regole in memoria senza effettuare query sul database.

Gestione servizi

Come descritto nell'analisi del problema, la prima modalità di classificazione di una connessione è per servizio. Ogni servizio è identificato da un numero di porta, che è il numero di porta di destinazione relativo al flusso della connessione che inizia dal client ed è rivolto al server in cui è in esecuzione il servizio. Il servizio, quindi, è un'entità utile solo a scopi di visualizzazione, ed è descritto dal diagramma in figura 4.3.

Le operazioni che si possono effettuare sulla risorsa `services` sono riportate nella tabella 4.4, e sono l'ottenimento di tutti i servizi esistenti e l'inserimento o la modifica di un servizio.

Anche i servizi, come le regole, vengono mantenute sia in memoria sia sul database. Ogni volta che un servizio viene modificato viene quindi attuata una sincronizzazione tra le due parti.

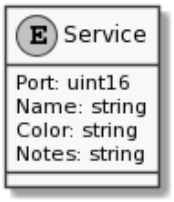

Figura 4.3: Struttura dati che rappresenta un servizio.

SERVICES			
Inserire o modificare i servizi			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/services`	ottiene la lista di `Service` presenti	
PUT	`/api/services`	inserisce un nuovo servizio e ne modifica uno esistente	la struttura `Service` definita in figura 4.3

Tabella 4.4: API relative alla risorsa **services**.

Ricostruzione e gestione connessioni

È il componente più importante, che si occupa di estrarre le connessioni TCP bidirezionali dal flusso di pacchetti di rete ricevuto in entrata. Di seguito sono descritte in dettaglio l'insieme delle funzionalità di questo componente.

- **Associare ed unire le connessioni unidirezionali**. Il componente che ricostruisce le connessioni a partire dai pacchetti di rete produce dei flussi di byte unidirezionali, perché come dalle specifiche in TCP sono presenti due flussi unidirezionali indipendenti. Lo strumento deve essere in grado di analizzare e visualizzare le connessioni come insieme dei due stream che ne fanno parte, è quindi necessario un meccanismo per individuare ed associare i due flussi distinti.

  L'algoritmo di accorpamento utilizzato è il seguente:

  1. si inizializza un dizionario vuoto che ha come tipo di chiave gli identificativi di stream e come tipo di valore gli stream stessi

2. viene generata la chiave di identificazione dello stream da associare, a partire dalla quadrupla <indirizzo sorgente, indirizzo destinazione, porta sorgente, porta destinazione>

3. viene generata la chiave di identificazione inversa dello stream da associare (ovvero dello stream nell'altra direzione), a partire dalla quadrupla <indirizzo destinazione, indirizzo sorgente, porta destinazione, porta sorgente>

4. se nel dizionario è contenuto uno stream con chiave invertita, lo stream individuato viene rimosso dal dizionario e associato allo stream da cui è stata generata la chiave

5. altrimenti lo stream da associare viene inserito nel dizionario utilizzando la chiave della giusta direzione

Il concetto di stream viene gestito da un componente separato, il cui scopo principale è quello di inserire i byte dei segmenti TCP man mano che questi vengono processati. Il componente che si occupa di gestire le connessioni deve essere notificato non appena entrambi gli stream di una connessione vengono chiusi. Non appena questo succede, vengono generati i metadati dai due stream e la connessione viene salvata su database.

In particolare viene calcolato il tempo di inizio come minimo fra i timestamp del primo segmento di ciascuno dei due stream, e il tempo di fine come massimo fra i timestamp dell'ultimo segmento di ciascuno dei due stream. Viene inoltre richiesto al componente che gestisce le regole di ottenere la lista di regole per le quali è stato fatto *match*.

In ultimo vengono aggiornati nel database i due stream associati alla connessione e memorizzati in un luogo differente, andando a scrivere il codice identificativo della connessione. La chiave di connessione è un hash calcolato in funzione del tempo di inizio e della quadrupla descritta sopra che identifica lo stream che ha richiesto per primo di instaurare la connessione.

- **Aggiornare le statistiche**. Ogni volta che una nuova connessione viene processata e completata vengono aggiornate le statistiche. Lo strumento mantiene statistiche temporali: le statistiche generate vengono aggregate in intervalli di tempo regolari per poter calcolare rapidamente le statistiche fra due istanti di tempo e per poter visualizzare le statistiche attraverso grafici.

- **Filtrare e ottenere le connessioni**. Le connessioni dopo essere state ricostruite vengono memorizzate e indicizzate nel database. Per ottenere la lista di connessioni che soddisfano determinati criteri è necessario applicare un filtro, le cui proprietà sono descritte nel diagramma riportato nella figura 4.4.

  Il componente permette anche di richiedere una singola connessione a partire dal suo ID. Ad ogni connessione sono associate diverse proprietà estratte durante la fase di estrazione dai segmenti TCP, che sono riportate nel diagramma in figura 4.6.

Figura 4.4: Struttura dati che rappresenta un filtro di ricerca per connessioni.

Nella tabella 4.5 sono riportate le operazioni che possono essere richieste al componente delle connessioni.

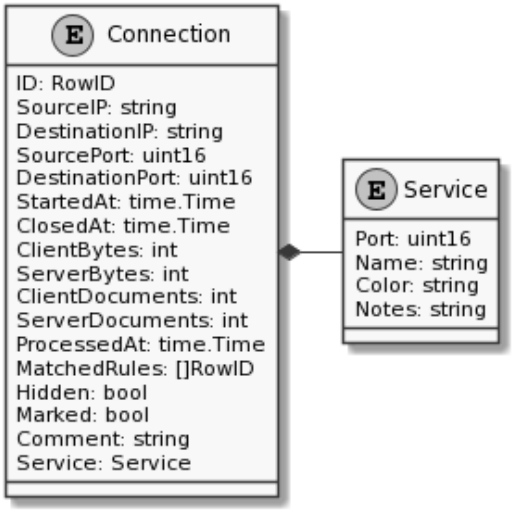

Figura 4.5: Struttura dati che rappresenta una connessione.

CONNECTIONS			
Per ottenere le connessioni già processate utilizzando dei criteri di selezione			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/connections`	ottiene una lista di `Connection` a partire da un criterio di ricerca	la struttura `ConnectionsFilter`, rappresentata in figura 4.4, che specifica i criteri di selezione
	`/api/connections/{id}`	ottiene la `Connection` specificata	**id**: l'id della connessione da ottenere
POST	`/api/connections/{id}` `/{action}`	esegue un'azione sulla connessione selezionata	**id**: l'id della connessione su cui effettuare l'azione **action**: la tipologia di azione da effettuare. Le operazioni disponibili sono *hide*, *show*, *mark*, *unmark*, *comment*

Tabella 4.5: API relative alla risorsa **connections**.

Aggregazione segmenti e gestione stream

Uno stream rappresenta un flusso unidirezionale di byte. Una connessione è composta da due stream differenti. Questo componente si occupa di ricostruire uno stream a partire dai segmenti TCP che vengono processati durante la fase di assemblaggio. Di seguito sono riportati i principali compiti di questo componente.

- **Aggregare i segmenti**. I segmenti ricevuti da questo componente sono in ordine e non presentano duplicati, l'operazione che viene effettuata è quindi quella di estrarre il contenuto man mano che i segmenti arrivano e accumularlo in un buffer.

  Siccome uno stream può essere potenzialmente infinito, è necessario suddividere il buffer che mantiene il flusso di byte in più chunk quando deve essere memorizzato. La suddivisione in chunk viene effettuata per due motivi:

  - il database ha dei limiti di dimensione, e ciascun record deve rispettare una dimensione massima prestabilita

  - supponendo che non ci siano limiti di dimensione e sul database sia presente un record di uno stream di grandi dimensioni, quando viene effettuata una query per richiedere il contenuto dello stream la query impiega molto tempo e utilizza molte risorse. Suddividendo quindi il contenuto in chunk è possibile inserire un meccanismo di paginazione, e trasferire soltanto i chunk necessari ai fini di analisi e visualizzazione

- **Memorizzare i chunk sul database**. Quando un chunk raggiunge la dimensione massima prestabilita oppure lo stream viene chiuso, è necessario salvare il chunk sul database e notificare il componente che gestisce le connessioni che lo stream per la direzione indicata dallo stream è stato chiuso.

  Ogni chunk di uno stream è rappresentato dalla struttura dati riportata in figura 4.6. L'id dello stream viene generato a partire dall'hash che

identifica lo stream ed è concatenato all'id incrementale del chunk, anche presente nel campo `DocumentIndex`. Il campo `ConnectionID` contiene il riferimento alla connessione a cui appartiene lo stream; all'inizio il campo è vuoto, e viene aggiornato soltanto quando due stream vengono riconosciuti come facente parte di un'unica connessione. Il campo `Payload` contiene il buffer in cui sono stati aggregati i byte dello stream. Il campo `BlocksIndexes` è un array di interi in cui ciascun elemento è un indice in riferimento al campo `Payload` e indica dove inizia ciascun segmento. Il campo `BlocksTimestamps` è un array di timestamp della stessa lunghezza di `BlocksIndexes` e mantiene i timestamp di arrivo di ciascun segmento dello stream. Il campo `BlocksLoss` è un array di valori booleani e serve per indicare se i segmenti sono frutto di una ritrasmissione, quindi se sono stati originalmente persi.

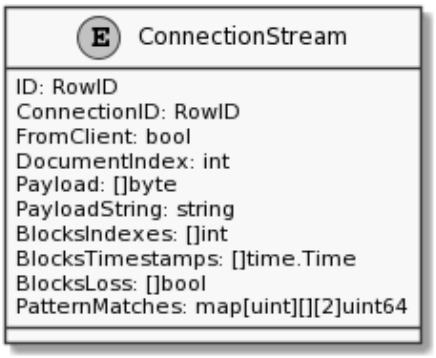

Figura 4.6: Struttura dati che rappresenta uno stream di una connessione.

- **Effettuare il pattern matching**. Come descritto nella sezione del componente che gestisce le regole, i pattern non sono altro che espressioni regolari. Tutti i pattern inseriti dagli utenti vengono compilati in un unico automa che permette di effettuare il *matching* di più espressioni regolari su uno stream di byte. Il funzionamento e le dinamiche di questo processo sono descritti in dettaglio nella sezione 4.4.

  Dai segmenti TCP man mano che vengono processati viene quindi estratto il contenuto e inviato al processore di pattern matching, che

richiama una callback non appena individua uno dei pattern presenti nel database. I pattern vengono quindi accumulati localmente. Per ciascun pattern individuato viene memorizzato il suo id e l'indice di inizio e di fine dell'occorrenza all'interno del buffer che contiene i byte dello stream. I pattern individuati vengono infine salvati su database nel campo `PatternMatches` di ogni chunk.

- **Ricostruire il flusso bidirezionale**. Ai fini di analisi e visualizzazione è importante considerare una connessione come un unico stream bidirezionale invece che due stream unidirezionali distinti. È possibile effettuare una ricostruzione a partire dai due stream distinti utilizzando un algoritmo banale. I due stream distinti possono essere visti come array di segmenti; in ciascun array i segmenti sono già ordinati seguendo i numeri di sequenza TCP. Per generare una conversazione unica è sufficiente estrarre dalla testa dei due array il segmento con timestamp inferiore, fino a consumare tutti i segmenti nei due array.

  Ciascun segmento all'interno della conversazione ricostruita viene inserito in una struttura dati di tipo `Message`, mostrata nel diagramma in figura 4.7. Il campo `FromClient` indica se il messaggio è stato inviato dall'agente che ha iniziato la connessione. Il campo `Content` è la rappresentazione testuale del contenuto del segmento. I campi `Index`, `Timestamp`, `IsRetransmitted` e `RegexMatches` hanno lo stesso significato di quello descritto precedentemente. Questo componente si occupa anche di estrarre ulteriori metadati dal protocollo di alto livello su cui si basa la conversazione. Il campo `Metadata` è una struttura personalizzabile dai `parser` di ciascun protocollo implementato. Ciascun `parser` implementa un metodo `TryParse` che riceve in input il contenuto di un messaggio e restituisce, se il protocollo è corretto, l'oggetto `Metadata` che contiene le informazioni estratte.

- **Scaricare e ottenere il flusso di una connessione**. Il componente permette di scaricare il contenuto della conversazione separando per caratteri di nuova linea i messaggi, oppure scaricare soltanto i messaggi

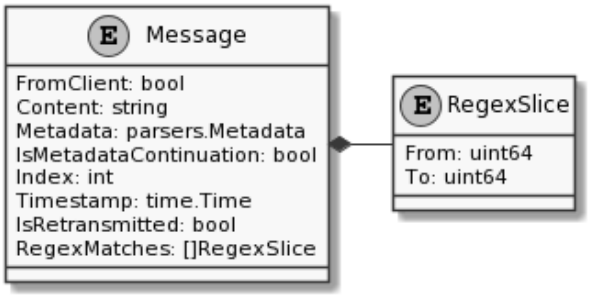

Figura 4.7: Struttura dati che rappresenta un messaggio di una conversazione.

di una delle due direzioni della connessione. È infine possibile scegliere il formato dell'encoding dei messaggi, ad esempio *hex* o *base64*.

Nella tabella 4.6 sono riportate le operazioni che possono essere richieste a questo componente.

Generazione statistiche

È il componente che si occupa di produrre statistiche sulle connessioni e sulle altre informazioni presenti sul database. Come descritto nella sezione del componente che gestisce le connessioni, le statistiche prodotte dal processo di estrazione delle connessioni vengono aggregate in intervalli di tempo regolari. Ogni record della struttura dati che mantiene le statistiche rappresenta un intervallo di tempo, e ha un timestamp di inizio e uno di fine. Tutte le connessioni che vengono inizializzate in quell'intervallo di tempo vengono aggregate nel record selezionato.
Le metriche principali implementate sono le seguenti:

- **ConnectionsPerService**: rappresenta il numero di connessioni effettuate raggruppate per servizio

- **ClientBytesPerService**: è il numero di byte trasmessi dall'agente che ha richiesto di stabilire la connessione, raggruppati per servizio

- **ServerBytesPerService**: è il numero di byte ricevuti dall'agente che ha richiesto di stabilire la connessione, raggruppati per servizio

STREAMS			
Per ottenere i messaggi di una conversazione di una connessione			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/streams/{id}`	ottiene una lista di `Message`, ovvero una lista di messaggi scambiati nella conversazione specificata	**id**: l'id della connessione di cui si vogliono ottenere i messaggi; **format**: l'encoding con cui devono essere restituiti i messaggi
GET	`/api/streams/{id}` `/download`	scarica i messaggi contenuti in una connessione	**id**: l'id della connessione di cui si vogliono scaricare i messaggi; **format**: l'encoding con cui devono essere scaricati i messaggi; **type**: la modalità con cui devono essere scaricati i messaggi (separati da newline, soltanto di un host senza separazione tra i messaggi, ecc..)

Tabella 4.6: API relative alla risorsa **streams**.

- **ClientBytesPerService**: è il numero di byte trasmessi complessivamente all'interno della connessione, raggruppati per servizio

- **DurationPerService**: rappresenta il numero di secondi totale in cui tutte le connessioni di un certo servizio sono rimaste aperte

- **MatchedRules**: indica il numero connessioni che sono state corrisposte per ogni regola presente nel database

È possibile ottenere le statistiche in due modalità:

- raggruppate per intervalli di tempo, esattamente come sono salvate su database. Il risultato dell'operazione è un array di `StatisticRecord`, in cui ciascun elemento rappresenta le statistiche per un singolo intervallo di tempo

- in forma aggregata, sommando i totali. Il risultato dell'operazione è un unico `StatisticRecord` con i valori delle statistiche sommati

Queste due operazioni sono riportate nella tabella 4.7, che mostra le API relative alla risorsa `statistics`.
È possibile creare una vista delle statistiche, andando a selezionare soltanto le metriche a cui si è interessati, così come è possibile selezionare l'intervallo di tempo. Nel diagramma riportato in figura 4.8 è mostrato l'oggetto `StatisticsFilter` utilizzato per applicare il filtro.

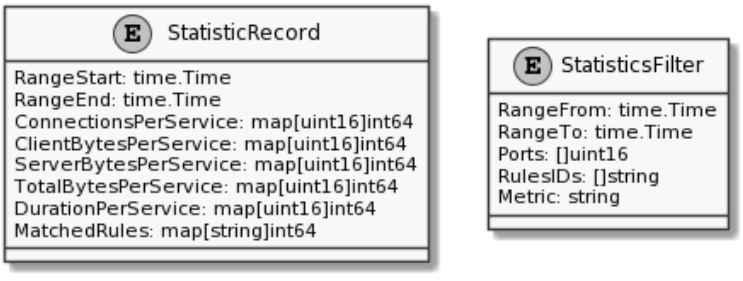

Figura 4.8: Struttura dati che rappresenta le statistiche relative ad un intervallo di tempo.

STATISTICS			
Ottiene le statistiche delle connessioni per intervalli di tempo regolari			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/statistics`	ottiene la lista di `StatisticsRecord`, ovvero le statistiche suddivise in intervalli di tempo regolari	la struttura `StatisticsFilter`, che permette di selezionare l'intervallo di tempo voluto e i servizi da filtrare
GET	`/api/statistics /totals`	ottiene un unico `StatisticsRecord` con le statistiche aggregate	la struttura `StatisticsFilter`, che permette di selezionare l'intervallo di tempo voluto e i servizi da filtrare

Tabella 4.7: API relative alla risorsa **statistics**.

Ricerca di connessioni

Lo strumento permette di ricercare un pattern all'interno del contenuto delle connessioni in due modalità:

- in modalità *online*, ovvero effettuando la ricerca quando le connessioni vengono processate, non appena gli stream vengono ricostruiti. Questa tipologia di ricerca, già descritta nelle sezioni precedenti, è implementata attraverso il sistema delle regole;

- in modalità *offline*, ovvero eseguendo la query di ricerca sulle connessioni già processate e presenti sul database. Il componente di ricerca si occupa di questo tipo di ricerca.

Per la ricerca *offline* sono permessi due tipi di ricerca:

- per espressione regolare, molto lenta perché la regex deve essere provata su tutte le connessioni presenti nel database in modo sequenziale e senza ottimizzazioni. È possibile ricercare sia le connessioni in cui ci siano

delle occorrenze risultato di una ricerca per un'espressione regolare, sia le connessioni in cui non si verifichino occorrenze;

- per termine, più rapida perché attraverso l'indicizzazione per testo è possibile individuare tutte le connessioni in cui è presente un termine in modo efficiente. La ricerca per termine permette di cercare tutte le connessioni che contengono un insieme di termini, che non contengono un insieme di termini, o che contengono una frase esatta.

Siccome la ricerca *offline* può essere lunga e costosa, si è deciso di effettuare *caching* per memorizzare i risultati delle ricerche effettuate. In questo modo è possibile visualizzare i risultati di ricerche passate senza dover rieffettuare le query di ricerca.

Ogni ricerca è rappresentata dalla struttura dati `PerformedSearch`, anche riportata nel diagramma nella figura 4.9. Ad ogni ricerca è associato un id univoco autogenerato. Nel campo `PerformedSearch` della struttura vengono memorizzati i parametri utilizzati per la ricerca. `AffectedConnections` contiene tutti gli identificativi delle connessioni che soddisfano i criteri di ricerca, mentre `AffectedConnectionsCount` indica il numero di connessioni trovate. Nei campi `StartedAt` e `FinishedAt` è specificato il timestamp di inizio e di fine, utile per calcolare quanto ha impiegato una ricerca ad essere effettuata. Le ricerche possono essere aggiornate, ovvero la ricerca può essere estesa anche sulle nuove connessioni che sono state inserite nel database dopo l'ultimo aggiornamento della ricerca. Il campo `UpdatedAt` indica il timestamp dell'ultimo aggiornamento. Infine, il campo `Timeout` indica il timeout utilizzato per effettuare la ricerca: se l'operazione non è riuscita a concludersi entro la durata definita in `Timeout`, la ricerca è incompleta e sono restituiti soltanto dei risultati parziali. È stato inserito il campo `Timeout` per non occupare troppe risorse quando una ricerca è molto lenta e non si conclude in tempi ragionevoli.

Nella tabella 4.8 sono riportate le API che possono essere utilizzate per effettuare ricerche di connessioni.

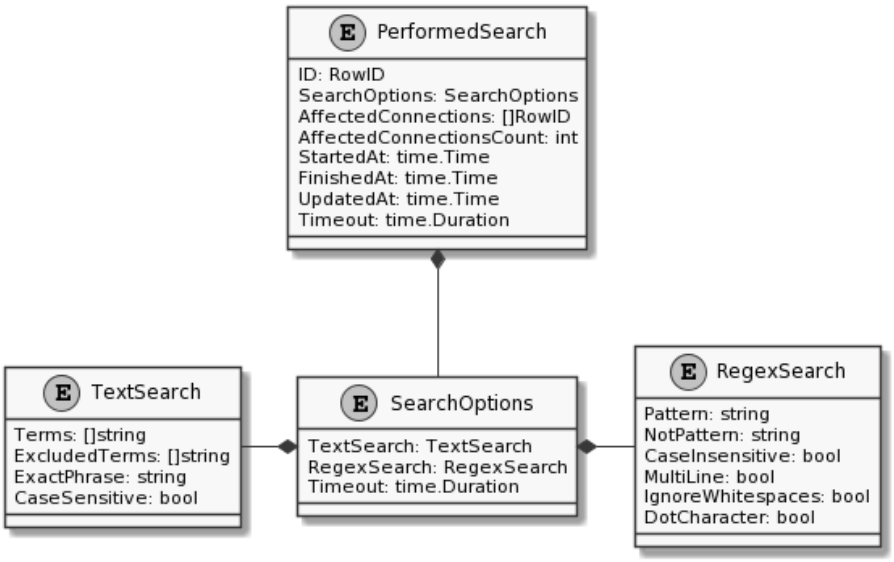

Figura 4.9: Struttura dati che rappresenta una ricerca di connessioni.

SEARCHES			
Cercare le connessioni il cui contenuto corrisponde ad un determinato pattern			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/searches`	ottiene la lista di `PerformedSearch`, ovvero le ricerche precedentemente fatte	
POST	`/api/searches /perform`	effettua una nuova ricerca	la struttura `SearchOptions` definita in figura 4.9

Tabella 4.8: API relative alla risorsa **searches**.

Gestione delle notifiche

Lo strumento permette ad agenti esterni di essere notificati non appena un evento viene scatenato. L'architettura delle notifiche è stata realizzata utilizzando il pattern *publish/subscribe*. Tutti i componenti del sistema possono richiedere al controllore delle notifiche di inviare un messaggio sotto forma di evento a tutti gli agenti che hanno effettuato la sottoscrizione. L'evento viene pubblicato a tutti, è compito quindi degli agenti che hanno effettuato la sottoscrizione scegliere se eseguire un'azione quando viene ricevuto un evento.

Il sistema di gestione delle notifiche è realizzato attraverso le websocket. Quando un nuovo client si vuole sottoscrivere alle notifiche effettua una richiesta utilizzando il protocollo delle websocket e rimane in ascolto degli eventi. Per interrompere la sottoscrizione il client può chiudere banalmente la connessione per non essere più notificato.

Ad ogni evento è associato un nome suddiviso in namespace, in cui la radice corrisponde al componente che ha generato l'evento.

Monitoraggio delle risorse

È il componente che si occupa di monitorare le risorse utilizzate dallo strumento, principalmente l'utilizzo della CPU, della RAM e del disco. Quando vengono superate soglie critiche lo strumento genera degli eventi di allerta.

Il componente permette anche di esportare le statistiche delle risorse utilizzate per essere visualizzate dal frontend e da altri strumenti di monitoraggio. Le due operazioni disponibili, per ottenere le informazioni sullo stato del processo e del sistema operativo, sono riportate nella tabella 4.9 relativa alle API della risorsa `resources`.

RESOURCES			
Ottiene le statistiche di utilizzo delle risorse			
METODO	URL	DESCRIZIONE	PARAMETRI
GET	`/api/resources /system`	ottiene le `SystemStats`, ovvero le statistiche di utilizzo delle risorse del sistema, come utilizzo di CPU, RAM e disco	
GET	`/api/resources /process`	ottiene le statistiche relative al processo	

Tabella 4.9: API relative alla risorsa **resources**.

4.4 Linguaggio, strumenti e librerie utilizzate

Go (Golang)

È il linguaggio utilizzato per l'implementare lo strumento. Go[5] è un linguaggio compilato e tipato staticamente, progettato e realizzato da Google. È sintatticamente simile al C, ma si differenzia dal C in quanto la gestione della memoria è automatica.
È stato scelto di usare Go perché ha una sintassi semplice e concisa. I sorgenti di programmi scritti in Go sono sempre molto simili: il linguaggio ha pochi costrutti e ci sono poche regole o convenzioni da seguire. Molte regole in Go sono intrinseche nel linguaggio, come ad esempio la *naming convention*: i nomi dei metodi e dei campi da esportare iniziano con una lettera maiuscola, altrimenti sono privati. Questo permette al linguaggio di assumere un unico stile e permette a sviluppatori diversi di scrivere un codice omogeneo senza preoccuparsi di uniformare lo stile.
Go ha il vantaggio di avere una standard library semplice e completa, e seppur sia un linguaggio relativamente nuovo ha un ecosistema di librerie molto vasto. Il fatto che la sintassi sia semplice e che i costrutti del linguaggio

[5]https://golang.org/

siano pochi, porta a scrivere un codice meno complicato e implicitamente più sicuro. Anche il fatto che sia fortemente tipato e che la gestione della memoria sia automatica implica maggiori vantaggi in termini di sicurezza.
Un altro enorme vantaggio di Go è quello di avere costrutti built-in per la scrittura di programmi concorrenti. Il linguaggio permette di implementare operazioni parallele o asincrone senza che ci sia bisogno di ricorrere a funzioni di libreria. Il costrutto principale che il linguaggio fornisce per operazioni concorrenti sono le *goroutine*, una tipologia di processi molto leggeri e gestiti interamente dal linguaggio. Go permette di lanciare funzioni in nuove *goroutine* diverse da quelle correnti, e utilizza il sistema dei *channels* per la sincronizzazione fra *goroutine* e per lo scambio di messaggi.
Il principale svantaggio di Go è che data la semplicità del linguaggio e la presenza di soli costrutti essenziali molte operazioni banali in altri linguaggi si complicano esageratamente. La mancanza di generici, ad esempio, costringe a scrivere più funzioni che effettuano la stessa operazione su array con tipi diversi. Una banale operazione di somma degli elementi di un array deve essere riscritta per tutti i tipi numerici e non può essere generalizzata. Questo porta ad una maggiore scrittura di codice duplicato.

GoPacket

È la libreria utilizzata per assemblare i pacchetti di rete, ricostruire i segmenti TCP ed estrarre gli stream delle connessioni. GoPacket[6] è scritta e mantenuta da Google, e permette di effettuare numerose operazioni su file PCAP. È stata scelta perché è efficiente, minimale e affidabile. Tra le librerie di analisi ed estrazione di informazioni dai pacchetti di rete è una di quelle che ottiene prestazioni migliori.
GoPacket è in grado sia di catturare il traffico direttamente da un'interfaccia di rete sia di leggere ed effettuare il *parsing* dei pacchetti contenuti in file PCAP. I pacchetti estratti possono essere processati in modo concorrente attraverso i costrutti del linguaggio. Filtrando solamente i pacchetti del protocollo TCP è possibile poi procedere con l'estrazione degli stream delle connessioni.

[6] https://github.com/google/gopacket

Anche il processo di ricostruzione può essere svolto concorrentemente. La libreria permette di istanziare una struttura di tipo `StreamPool` che mantiene in memoria i metadati e le informazioni utili per riassemblare i pacchetti ed estrarre gli stream. A ciascun `StreamPool` possono essere assegnati più `Assembler`, oggetti che permettono di riassemblare i singoli pacchetti man mano che arrivano.

Ogni volta che viene individuato un nuovo stream la libreria chiama una callback, definita dallo strumento nel componente delle connessioni. GoPacket tratta gli stream come flussi unidirezionali, ma una connessione è composta da due stream unidirezionali con verso opposto. È compito del componente delle connessioni, come anche descritto precedentemente, individuare ed associare due stream che fanno parte della stessa connessione.

Ogni stream individuato definisce due callback, definite dallo strumento nel componente degli stream: `Reassembled` e `ReassemblyComplete`. `Reassembled` viene invocata quando uno o più segmenti sono stati processati e devono essere aggiunti in coda allo stream. I segmenti ricevuti con la callback sono già in sequenza: la libreria si occupa quindi di riordinare i segmenti nel caso in cui non si ricevessero in ordine. GoPacket ha inoltre il compito di rimuovere i segmenti duplicati e i segmenti malformati. `ReassemblyComplete` viene invocata quando uno stream viene chiuso o terminato per errore, per indicare all'utente che il processo di estrazione è concluso e per lo stream corrente non arriveranno più segmenti.

Hyperscan

Lo strumento permette di ricercare dei pattern di stringhe all'interno del contenuto delle connessioni, man mano che queste vengono processate. I pattern vengono definiti attraverso il sistema delle regole, come descritto nella sezione del componente delle regole. I pattern sono definiti tramite espressioni regolari. Lo strumento permette di inserire un numero illimitato di regole. Il modo più semplice di controllare quali pattern, dato un insieme di pattern, corrispondono se confrontati con una connessione, è controllarli singolarmente sequenzialmente. Ogni volta che viene aggiunto un nuovo pattern al database dei pattern però l'operazione diventa sempre più lenta.

Hyperscan[7] è una libreria che permette di effettuare il *matching* di espressioni regolari multiple ad alte prestazioni. Utilizza la sintassi per le espressioni regolari definita nella libreria PCRE, utilizzata da quasi tutte le implementazioni. Hyperscan utilizza tecniche di automi ibridi per consentire il *matching* simultaneo di grandi numeri (fino a decine di migliaia) di espressioni regolari, e permette di cercare la corrispondenza di espressioni regolari anche tra stream di dati.
Hyperscan impiega due tecniche fondamentali per un pattern matching efficiente. Innanzitutto, sfrutta la scomposizione dei grafi che traduce il *matching* di espressioni regolari in una serie di corrispondenze di stringhe e automi finiti. A differenza delle soluzioni esistenti, la corrispondenza delle stringhe diventa parte della corrispondenza delle espressioni regolari, eliminando le operazioni duplicate. I componenti decomposti delle espressioni regolari aumentano anche la possibilità di un rapido DFA *matching* poiché tendono ad essere più piccoli dei pattern originali. In secondo luogo, Hyperscan accelera il *matching* sia di stringhe che di automi finiti utilizzando operazioni SIMD, che portano a un sostanziale miglioramento del throughput.
Il *matching* delle espressioni regolari è una funzionalità chiave delle applicazioni di sicurezza di rete. Spesso diventa però il collo di bottiglia delle prestazioni poiché implica una scansione pesante in termini di performance di ogni byte del payload del pacchetto. Con la tendenza verso l'aumento della larghezza di banda della rete e un ampio set di regole di modelli complessi, i requisiti di prestazioni diventano sempre più esigenti. Come documentato in [9], Hyperscan migliora le prestazioni notevolmente, e per questo è tipicamente utilizzato in applicazioni di tipo DPI (Deep Packet Inspection).
Il progetto utilizza la libreria Hyperscan utilizzando il binding delle API scritto in Go[8]. Come anche riportato nella descrizione del componente dello stream, è stata utilizzata la libreria in modalità streaming.

Gin Web Framework

Gin è un web framework scritto in Go. È molto utilizzato date le sue elevate performance e la struttura delle API minimale che ne rende facile l'utilizzo.

[7] https://www.hyperscan.io/
[8] https://github.com/flier/gohs

È veloce, perché utilizza un instradamento basato su *radix tree* che non richiede un utilizzo elevato della memoria. Supporta la creazione di middleware e ne fornisce diversi già pronti per l'utilizzo, come il middleware per l'autenticazione e quello per la compressione delle richieste. Gestisce in maniera efficiente gli errori, e impedisce al programma di interrompersi quando se ne verifica uno. Permette di validare richieste JSON per verificare ad esempio se sono presenti campi obbligatori. Infine, supporta la creazione di gruppi di *route* per organizzare le API più facilmente, permettendo di raggruppare quelle con radice in comune e che utilizzano gli stessi middleware.
Gin è stato scelto per gestire e implementare l'architettura REST dello strumento, data la sua versatilità, minimalità ed efficienza nel gestire le richieste. Per verificare se le richieste JSON siano valide, ovvero contengano tutti i campi obbligatori e tutti i vincoli di tipo e di semantica siano rispettati, è stato utilizzato `go-playground/validator`[9], che si integra facilmente con Gin. Per implementare la comunicazione attraverso websocket per la gestione delle notifiche è stata utilizzata la libreria `gorilla/websocket`[10], integrabile anch'essa con Gin.

MongoDB

Come definito nell'analisi del problema, è stato scelto di utilizzare un database non relazionale, bilanciato in lettura e scrittura e senza vincoli di schema. La scelta è ricaduta su MongoDB, il DBMS NoSQL orientato ai documenti. MongoDB permette di memorizzare record all'interno di collezioni in un formato compatibile con il JSON. L'integrazione dei dati con il modello del dominio dello strumento è stata quindi facilitata, dato che ciascun oggetto o struttura con lui lavora lo strumento è serializzabile facilmente in formato JSON.
MongoDB è stato scelto per diversi motivi, elencati di seguito.

- Non richiede di definire uno schema. I metadati estratti dalle connessioni non seguono uno schema predefinito, ma possono variare da

[9] https://github.com/go-playground/validator/
[10] https://github.com/gorilla/websocket

protocollo a protocollo. Avere un database senza schema è quindi un requisito importante.

- È facile da installare. Esistono anche immagini preconfigurate per utilizzare MongoDB attraverso i container. Le operazioni di configurazione sono essenziali e non impegnative.

- Non sono richiesti particolari requisiti tecnici e di sistema, e può scalare facilmente se si aggiungono più istanze per raggiungere performance migliori e per aumentare le capacità del database.

- Il linguaggio di query di MongoDB è molto evoluto e include un vasto numero di espressioni che possono essere utilizzate ad esempio in una query di selezione.

- Le performance di MongoDB sono nettamente superiori se comparate con qualsiasi altro database relazionale.

MongoDB ha anche degli svantaggi, ma che non influiscono sulle necessità di questo progetto e non ne inficiano i benefici. Di seguito sono elencati i principali svantaggi.

- Non sono permesse operazioni di *join*, perché non possono essere create relazioni tra i dati. Ciascun componente del sistema è stato modellato però per avere le proprie strutture dati in modo tale che non abbiano relazioni con le strutture di altri componenti. Ogni componente può quindi avere la propria collezione in cui inserisce la propria tipologia di documenti.

  Permette però di avere uno schema dei documenti variabile e complesso. Nei casi in cui è risultato necessario in un documento fare riferimento a un documento di un'altra collezione si è proceduto con la duplicazione dei dati. Questa soluzione è stata adottata quando i dati da duplicare erano pochi e non c'era necessità di sincronizzazione fra le collezioni.

- Utilizza molta memoria per la memorizzazione di dati. Lo strumento però solitamente deve essere utilizzato per un tempo limitato, come quello di una competizione o di una sessione di analisi. Se lo strumento viene utilizzato su cloud la necessità di spendere per ottenere più risorse per un intervallo di tempo limitato non è eccessivamente rilevante sui costi.

- C'è un limite sulla dimensione dei documenti, che è di 16 MB. In relazione alle necessità dello strumento però il problema non si pone, dato che gli unici documenti che possono essere di dimensioni elevata sono gli stream che sono però già suddivisi in chunk da progetto. L'unico vincolo, che non influisce però sulle prestazioni e sull'usabilità dello strumento, è quindi quello di impostare come dimensione massima dei chunk quella di un documento di MongoDB.

- Non c'è supporto alle transazioni. Le operazioni critiche sono però poche, e un comportamento transazionale può essere implementato nella logica applicativa. Per la maggior parte delle operazioni, e per quelle più eseguite, non serve la presenza di transazioni. Le connessioni e gli stream sono univoci e non possono più essere modificati una volta inseriti nel database.

Numerose applicazioni e progetti di analisi del traffico di rete utilizzano MongoDB come database per i log e per le informazioni estratte. Uno di questi, descritto in [10], è *Analysis farm*, una piattaforma scalabile che analizza i log di rete. Per sopperire all'enorme quantità di dati che viene generata quando si registrano le attività di rete *Analysis farm* usa un cluster di più istanze di MongoDB per memorizzare i dati e per permettere un'analisi efficiente.

4.5 Validazione e dispiegamento

Prima di iniziare con lo sviluppo del progetto ho realizzato un prototipo per la ricostruzione delle connessioni TCP a partire da un flusso di rete. Il processo di estrazione delle connessioni è il fulcro dello strumento e comprende le

operazioni più complicate. La fase di prototipazione è servita per chiarire alcuni dubbi sul processo di estrazione e per scegliere le librerie più efficienti ed efficaci adatte al caso.
Terminata la fase di prototipazione ho iniziato a definire le interfacce e le strutture dati dello strumento, definendo fin da subito le API necessarie e modellando il dominio seguendo l'architettura REST. Lo strumento è stato poi implementato seguendo un approccio primitivo di *test-driven development*, definendo i test prima di implementare le interfacce precedentemente definite.

Test unitari e d'integrazione

Come framework di test si è utilizzato `testify`[11], un wrapper che fornisce funzionalità di test avanzate realizzate sopra il meccanismo standard di test del linguaggio Go. Il framework permette la validazione della maggior parte dei tipi primitivi e delle strutture della libreria standard di Go, e permette due modalità di verifica: *assert* per assicurarsi che una certa condizione sia rispettata, senza fare fallire i test; *require* per impedire al test di procedere quando una condizione non è rispettata.
Attraverso il framework sono stati realizzati sia i test unitari, utili per verificare il corretto funzionamento di ciascuna funzione e procedimento implementato, sia i test d'integrazione, per verificare che le funzionalità dello strumento producessero il risultato atteso.

Difficoltà di validazione

Sono emerse due principali difficoltà durante le attività di testing:

- i test di integrazione per la verifica del processo di estrazione delle connessioni da un flusso di rete sono particolarmente complessi da attuare. Lo strumento si interfaccia direttamente con il database per memorizzare lo stato e il risultato delle operazioni, quindi si è rivelato necessario effettuare delle operazioni di *mocking* del database per intercettare le operazioni di lettura e scrittura. Attraverso un database *mocked* è stato

[11] https://github.com/stretchr/testify

quindi possibile verificare che le operazioni di scrittura fossero corrette, e fornire risultati strumentalizzati durante le operazioni di lettura.

- molte funzionalità dello strumento sono realizzate attraverso procedimenti asincroni ed è stato fatto largo uso di costrutti concorrenti come le Goroutine per implementare il parallelismo. Durante la fase di testing si è quindi rivelato complesso costruire test per rilevare corse critiche e intercettare problemi di deadlock.

  Parte dei problemi dovuti alla concorrenza sono stati risolti utilizzando il `Go Race Detector`, uno strumento integrato nella toolchain di Go per il rilevamento delle corse critiche. Il compilatore strumentalizza tutti gli accessi alla memoria con del codice che registra quando e come è stato effettuato l'accesso alla memoria, mentre la libreria runtime controlla gli accessi non sincronizzati alle variabili condivise. Quando viene rilevato un accesso concorrente viene stampato un avviso che mostra la porzione di codice problematica.

Continuous Integration e Continuous Delivery

Durante tutte le fasi di realizzazione dello strumento sono ampiamente state utilizzate tecniche di Continuous Integration e Continuous Delivery per automatizzare le procedure di testing e per costruire gli artefatti ad ogni milestone del progetto.
È stato utilizzato il servizio offerto per i progetti open source di Travis CI per eseguire i test ad ogni evento di push del repository e ad ogni pull request creata. I test effettuati su cloud vengono eseguiti utilizzando Docker per creare facilmente l'ambiente di testing necessario per l'esecuzione dei test. Attraverso Docker è stato infatti possibile creare e distruggere istanze di MongoDB, il database utilizzato dallo strumento, in maniera rapida e minimizzando il consumo di risorse dovute al processo di installazione.
Ogni volta che viene rilasciata una versione stabile (si intende stabile il branch main) viene automaticamente costruita l'immagine Docker dello strumento e caricata su DockerHub, il servizio che permette di condividere facil-

mente immagini Docker. L'immagine ufficiale dello strumento è disponibile all'indirizzo `https://hub.docker.com/r/eciavatta/caronte`.

Docker è stato utilizzato per permettere facilmente agli utilizzatori di installare lo strumento senza dover scaricare manualmente e installare i componenti necessari per compilare ed eseguire lo strumento. Attraverso la utility `docker-compose` è infatti possibile avviare un'istanza dello strumento in pochi secondi in un ambiente sicuro e isolato dalla macchina host e senza che ci sia bisogno di installare il database.

4.6 Interfaccia utente

Il frontend è stato realizzato con React, un framework JavaScript per costruire interfacce utente caratterizzate dal fatto di essere dichiarative, efficienti e flessibili. React permette di comporre UI complesse definendo una struttura gerarchica formata da componenti, ovvero elementi della pagina che definiscono un comportamento proprio e che possono essere riutilizzati.

Il frontend comunica con il backend attraverso richieste AJAX alle API esposte dallo strumento, seguendo un'architettura REST. Le richieste vengono effettuate attraverso l'API di JavaScript `Fetch`[12], che permette di realizzare richieste asincrone all'endpoint delle API dello strumento.

Quando un'istanza del frontend viene avviata viene aperta una connessione tramite websocket al backend per poter ricevere gli eventi che vengono generati dallo strumento. L'interfaccia grafica è reattiva e si aggiorna in automatico ogni volta che nuovi dati sono disponibili o delle modifiche sono state effettuate. L'applicazione web è quindi utilizzabile da più utenti in contemporanea, e le azioni effettuate da un utente sono propagate in modo automatico a tutti.

L'interfaccia grafica implementa tutte le funzionalità presenti nello strumento disponibili attraverso API. Di seguito sono riportati alcuni screenshot dello strumento con annessa una breve spiegazione di ogni funzionalità presentata in ogni schermata.

[12] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

La figura 4.10 mostra la schermata principale dello strumento. La finestra è suddivisa in un layout con pannelli ridimensionabili. La parte superiore dell'interfaccia è composta da due pannelli dinamici, mentre nella parte inferiore è presente soltanto un unico pannello fisso.

Il pannello superiore sinistro mostrerà sempre la tabella di connessioni estratte dal traffico di rete fornito. Ciascuna riga rappresenta una connessione, e per ciascuna connessione sono mostrate informazioni quali servizio a cui fa riferimento, indirizzo e porta sorgente, indirizzo e porta destinazione, timestamp, durata e dimensioni. Le connessioni possono essere contrassegnate, per poterle ritrovare facilmente, e può essere associato a loro un commento. Le connessioni vengono mostrate utilizzando come ordinamento il timestamp in cui sono state inizializzate, in ordine decrescente. In cima alla tabella si troveranno quindi le connessioni effettuate di recente. Non è presente paginazione per scorrere fra le connessioni, ma queste vengono caricate dinamicamente man mano che si scorre la tabella.

Figura 4.10: Schermata principale dello strumento.

Anche in figura 4.11 è riportata la schermata principale dello strumento, in cui però sono mostrate altre funzionalità del pannello di dettaglio posizionato sulla destra, in cui viene mostrato il contenuto di ciascuna connessione. Ciascun segmento della connessione viene mostrato in un riquadro, etichettato con *client* se il segmento è stato inviato dall'host che ha inizializzato la connessione, o *server* se il segmento è stato inviato dall'host in ascolto.

Lo strumento effettua il parsing del contenuto dei protocolli più frequenti, come HTTP, formattando il contenuto in modo adeguato. La schermata mostra che è possibile cambiare il formato di encoding di visualizzazione dei messaggi e scaricare il contenuto dell'intera conversazione.

Nella parte inferiore della schermata è invece presente la timeline, che mostra le statistiche aggregate per minuto relative in base alla metrica selezionata (di default è mostrato il numero di connessioni per ciascun servizio). Nella timeline è presente una finestra scorrevole che può essere spostata e ridimensionata per visualizzare le connessioni in un certo intervallo di tempo.

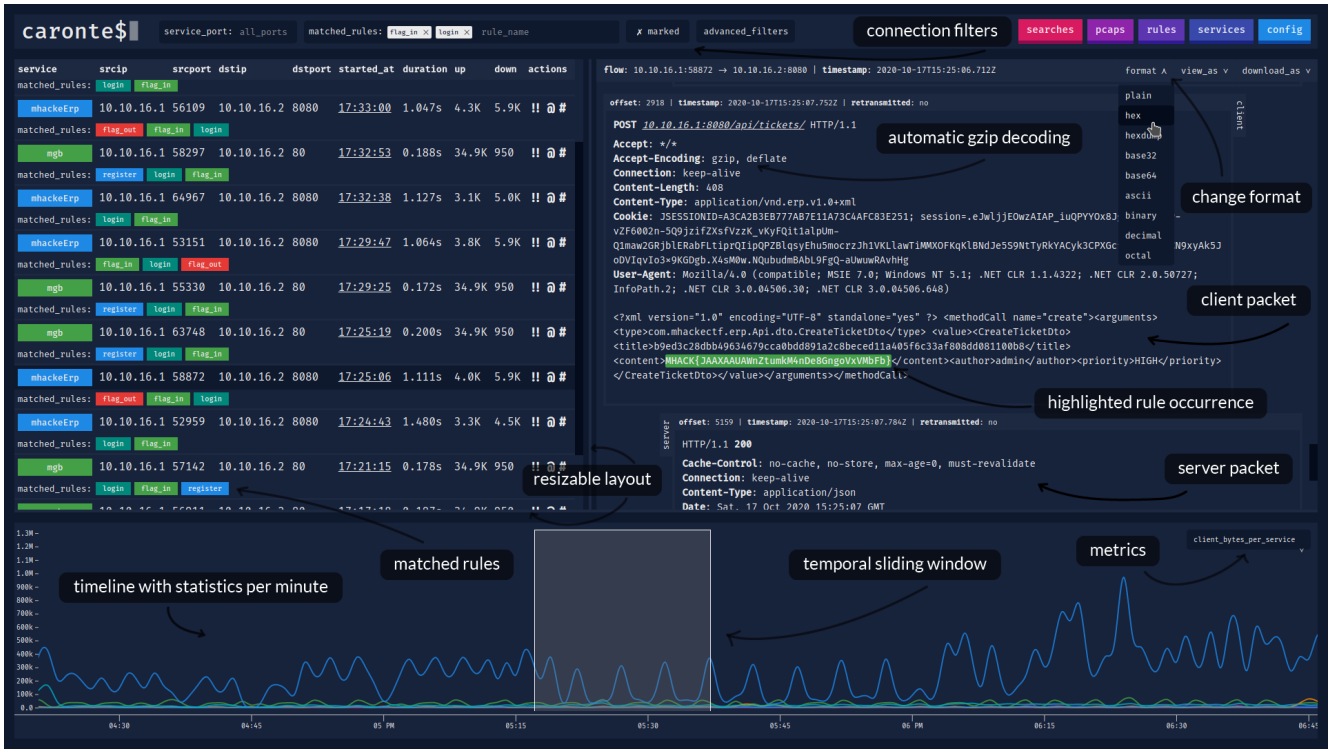

Figura 4.11: Schermata principale dello strumento.

La schermata in figura 4.12 mostra due pannelli. A sinistra è presente il pannello di configurazione delle regole, a destra quello dei servizi.

Nel pannello delle regole è possibile inserire nuove regole o modificare quelle esistenti. Per ciascuna regola è possibile inserire uno o più pattern, ovvero espressioni regolari che verranno cercate non appena verranno estratte nuove connessioni. Le regole, come spiegato nelle sezioni precedenti, non funzionano retroattivamente sulle connessioni già ricostruite e memorizzate nel database.

Non è permesso modificare i pattern o i filtri di una regola creata, ma è possibile cambiarne il nome, il colore e le note associate ad essa. Esistono due regole predefinite denominate `flag_in` e `flag_out`, che etichettano rispettivamente le connessioni in cui il master di gioco inserisce le flag nei servizi e le connessioni in cui vengono richieste legittimamente o esfiltrate le flag.

Nel pannello dei servizi è possibile associare ad ogni porta destinazione delle connessioni estratte un'etichetta per poter essere filtrate e individuate velocemente.

Figura 4.12: A sinistra il pannello delle regole; a destra quello dei servizi.

Nella schermata in figura 4.13 sono mostrati a sinistra il pannello di ricerca, a destra il pannello delle sessioni e dei PCAP.
Nel pannello di ricerca è possibile effettuare ricerche nel database delle connessioni estratte. Sono permesse due modalità di ricerca: per termine/frase esatta, o per espressione regolare. La ricerca per termini è consigliata perché sfrutta l'indicizzazione del database, mentre la ricerca per espressione regolare è molto più lenta perché ogni connessione presente nel database deve essere *matchata*. Le ricerche vengono salvate nel database e possono essere visualizzate nuovamente in modo rapido.
Il pannello delle sessioni mostra tutti i PCAP che sono stati processati. Per ciascun PCAP sono associate le statistiche di processamento. I PCAP possono anche essere scaricati. Nella parte inferiore sono presenti due moduli che permettono di processare PCAP caricandoli dal browser o indicandone il percorso sul filesystem in cui è avviato lo strumento.

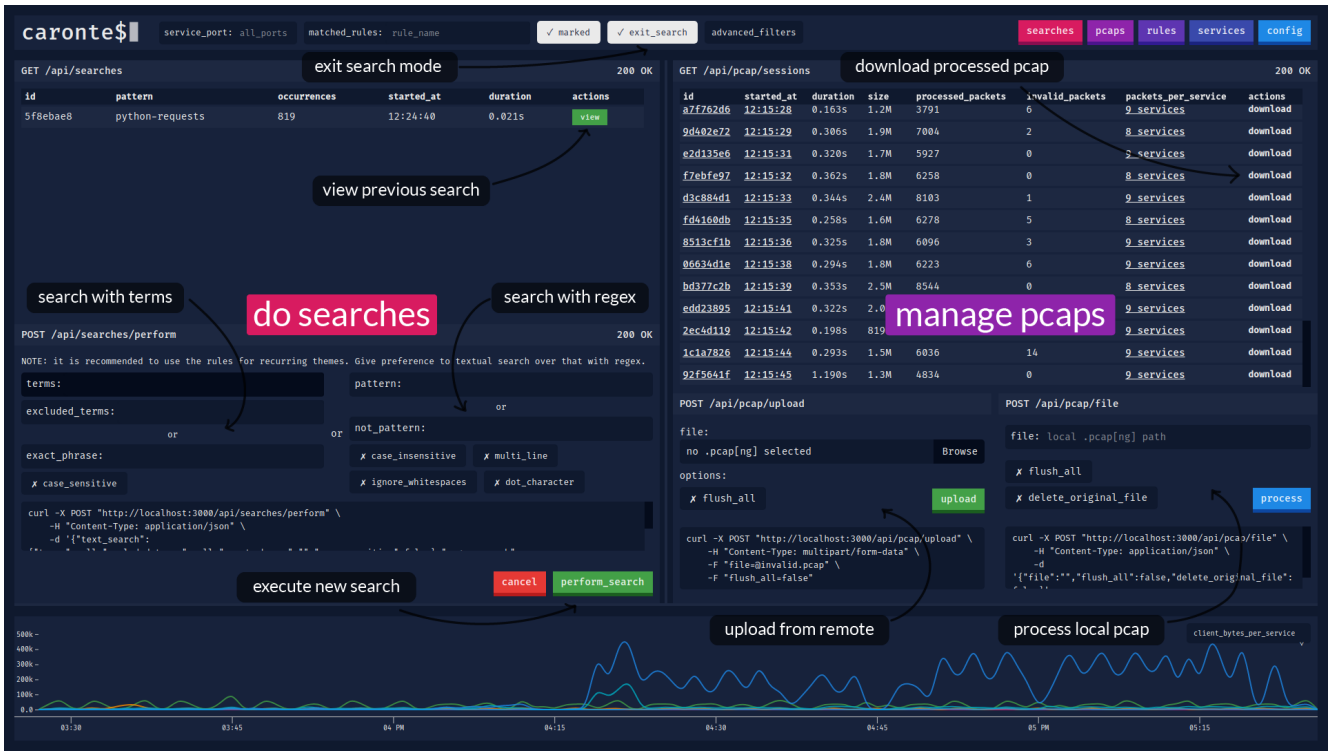

Figura 4.13: A sinistra il pannello di ricerca; a destra quello delle sessioni.

Capitolo 5

Aspetti innovativi

Nel capitolo precedente è stato descritto il percorso seguito per la realizzazione di Caronte, a partire dalla fase di raccolta di requisiti fino all'implementazione e al rilascio. È stato quindi descritto il procedimento e l'architettura dello stato attuale dello strumento nella sua versione più stabile, rilasciata con licenza GPL-3.0 su Github[1].
L'intenzione di questo capitolo è invece descrivere gli aspetti più innovativi e di ricerca che sono stati individuati e che non sono ancora stati integrati nelle versioni stabili dello strumento. Alcune funzionalità e idee qui descritte potrebbero anche non essere implementate perché non efficaci o non abbastanza rilevanti. In questo capitolo verranno discusse sia proposte che potrebbero sostituire componenti o dinamiche esistenti, sia proposte che integrano funzionalità nello strumento. Le proposte verranno suddivise per sezioni.

5.1 Indicizzazione per differenza

Caronte è stato realizzato con uno scopo ben preciso: analizzare il traffico di rete durante una competizione di tipo attacco/difesa, anche se lo strumento si adatta a qualsiasi altro tipo di situazione. In una competizione di tipo attacco/difesa ci sono pochi servizi da difendere (solitamente meno di dieci), ma ciascuno di questi riceve un traffico elevato.

[1] https://github.com/eciavatta/caronte

La maggior parte dei servizi ha un protocollo di comunicazione semplice e strutturato. I messaggi di richiesta e di risposta tra diversi messaggi dello stesso servizio sono molto simili, e in molti protocolli contengono una parte statica che si ripete. Le richieste del protocollo HTTP, ad esempio, utilizzano una struttura fissa: due richieste ad una stessa risorsa sono molto simili. Solitamente l'unica differenza è l'User-Agent del client che ha effettuato la richiesta, che però anch'esso si ripete: due richieste alla stessa risorsa dallo stesso client producono due richieste HTTP identiche.
Anche i contenuti statici serviti da un servizio web sono sempre gli stessi. Il codice CSS o JavaScript di una pagina non cambia tra una richiesta e l'altra. In una pagina HTML che mostra dei dati in formato tabellare soltanto il contenuto delle celle della tabella cambia tra una richiesta e l'altra, mentre la struttura della pagina rimane invariata.
Memorizzare ciascun messaggio di ciascuna connessione nel database come è implementato nella versione attuale dello strumento implica che nel database la maggior parte dei messaggi siano duplicati, o comunque molto simili. Questo comporta uno spreco di spazio di archiviazione e una maggiore necessità di banda per trasferire i messaggi sul o dal database. Inoltre, la presenza di molti dati duplicati all'interno del database rallenta la ricerca: in una ricerca per espressione regolare occorre testare la regex per ogni record del database. Se molti record sono duplicati si effettua inutilmente una ricerca più volte, consumando più tempo.
L'idea è quindi quella di trovare un meccanismo per memorizzare nel database soltanto messaggi che non siano già stati memorizzati precedentemente, collegando i messaggi identici o simili fra di loro. Di seguito verranno descritte le proposte per cercare una soluzione a questo problema.

Tecniche esistenti di compressione e deduplica

Semplice compressione dei record

Il modo più semplice per utilizzare meno spazio di archiviazione è quello di utilizzare un algoritmo di compressione per ridurre la dimensione dei record. Gli algoritmi più utilizzati e più adatti per questo scopo sono LZ77 e LZ78, descritti rispettivamente in [11] e [12]. Sono entrambi stati ideati da Abraham

Lempel e Jacob Ziv e sono due algoritmi *loseless*, ossia senza perdita di informazioni. Esistono anche altre versioni di algoritmi di cifratura della stessa famiglia più ottimizzati per scopi più specifici.
Questa famiglia di algoritmi rientra nella classe dei codificatori a dizionario, che operano ricercando corrispondenze tra il testo da comprimere e un insieme di stringhe contenute in una struttura dati: il dizionario. Quando il codificatore trova una corrispondenza, sostituisce la porzione di stringa duplicata con il riferimento della stringa nel dizionario.
Utilizzare un algoritmo della famiglia LZ per comprimere il contenuto di ogni singola connessione, però, non porta ulteriori vantaggi se non quello di ridurre lo spazio su disco utilizzato. Indipendentemente da quanto sia efficace l'algoritmo di compressione utilizzato, e da quanto all'interno del contenuto di una connessione ci siano stringhe ripetute, questo metodo non risolve il problema dei record duplicati. Per effettuare la ricerca per stringa o per espressioni regolari, quindi, è necessario comunque verificare tutti i record, moltiplicando la ricerca per i record duplicati.

Deduplica dei dati

La deduplica è una tecnica per eliminare le copie duplicate dei dati ripetuti. A differenza della compressione dei documenti, questa tecnica è trasversale tra i record. Il processo di deduplica suddivide un documento da memorizzare in chunk, ovvero blocchi contigui di byte. Per ciascun chunk viene calcolato un hash utilizzando una funzione crittografica. Se nel database è già presente un chunk con lo stesso hash viene inserito un riferimento all'hash corrispondente, altrimenti viene inserito il contenuto del chunk.
Gli hash di ciascun chunk devono essere mantenuti in una *hashtable* per individuare in tempo lineare se un chunk è già presente nel database. Per la funzione di hash è necessario trovare un compromesso tra una funzione crittografica efficace che minimizza il numero di collisioni e una funzione efficiente che richiede meno risorse computazionali.
La tecnica di deduplica è molto efficiente quando i chunk all'interno delle connessioni sono allineati, ovvero quando i chunk iniziano sempre nelle stesse posizioni. Quando i chunk non sono allineati, ovvero quando c'è del conte-

nuto in lunghezza variabile prima di un chunk, non si rileveranno mai chunk duplicati. Una soluzione a questo problema è quella di utilizzare una finestra scorrevole per calcolare i digest dei chunk. Con questo metodo, però, è necessario calcolare l'hash per ogni posizione della finestra. Questa operazione richiede però un utilizzo della CPU molto elevato.

Implementare un meccanismo di deduplica dei dati può quindi risolvere il problema di memorizzare nel database connessioni duplicate. Questa tecnica però è inefficace nel caso in cui due chunk non siano perfettamente identici, ma simili. Nel caso di protocolli di rete, come descritto precedentemente, capita infatti spesso di avere contenuti con una struttura fissa ma alcune parti variabili, che farebbero cambiare il digest prodotto dalla funzione di hash e rendere i chunk unici.

Nuova proposta: indicizzazione per differenza

Per risolvere i problemi dei due metodi riportati nella sezione precedente si propone una nuova tecnica: individuare le connessioni simili, estrarre un pattern in comune e memorizzare nel database soltanto le differenze. Per pattern comune si intende una stringa che è contenuta in tutte le connessioni a cui è associato quel determinato pattern. Non è necessario che tutti i byte o caratteri del pattern siano adiacenti nel contenuto delle connessioni, l'unico vincolo è che si presentino nello stesso ordine.

Per fare un esempio, il pattern `ABCDEF` può generare le stringhe `ZABCDEFY` e `TABBCGDEQF`, ma non può generare la stringa `FABCDE`, perché il carattere `F` si deve trovare dopo il carattere `E`.

Rispetto alla semplice compressione e alla compressione con dizionario in comune questo metodo velocizza la ricerca, perché questa viene effettuata soltanto sulle parti univoche del contenuto delle connessioni mentre viene saltata la ricerca sulle parti in comune, solitamente non interessanti. Rispetto alla deduplica dei dati questa tecnica è resistente alle connessioni simili ma non identiche, ovvero la maggior parte delle connessioni di rete.

Individuazione delle connessioni simili

Il primo passo per poter generare un pattern comune è quello di individuare le connessioni simili. Il procedimento volto alla selezione e al raggruppamento di connessioni omogenee è definito *clustering*. La bontà dei risultati ottenuti dalle tecniche di clustering dipende molto dalla scelta della metrica con cui è calcolata la distanza fra gli elementi, ovvero una misura quantificabile che indica quanto gli elementi siano diversi fra loro. Gli algoritmi di clustering raggruppano gli elementi sulla base della loro distanza reciproca, e quindi l'appartenenza o meno a un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso.
Di seguito verranno descritti gli algoritmi e le tecniche per calcolare la distanza o la similarità fra stringhe testuali o binarie che sono stati presi in considerazione per individuare connessioni simili. Infine, saranno riportati gli algoritmi che hanno ottenuto punteggi più alti nel contesto delle connessioni di rete.

Distanza di Levenshtein

Anche chiamata distanza di edit, è un algoritmo per misurare la distanza fra due stringhe. Il risultato della funzione è una metrica che indica quanto due stringhe siano simili, ed è calcolata sommando il numero di operazioni di cancellazione di un carattere, sostituzione di un carattere con un altro e inserimento di un carattere per trasformare una stringa in un'altra.
Per sapere quando due stringhe sono simili è possibile calcolare un punteggio a partire dalla distanza di Levenshtein, utilizzando la seguente formula:

$$score = \frac{levenshtein(A, B)}{max(len(A), len(B))}$$

Il punteggio ottenuto è molto rappresentativo riguardo alla similarità fra le due stringhe e fra quanto queste contengano dei sottoinsiemi di caratteri o byte in comune. Il problema della distanza di Levenshtein consiste nel fatto che per essere calcolata richiede una complessità computazionale approssimata a $O(n^2)$, dove n è la lunghezza delle stringhe. Applicare Levenshtein fra tutte le coppie di stringhe per individuare quelle simili richiede un tempo troppo elevato.

sdhash

È una libreria che permette di calcolare i *similarity digest* di stringhe testuali o binarie. L'algoritmo di `sdhash` rientra nella categoria dei *fuzzy hashing*, delle speciali funzioni di hashing che generano digest che possono essere comparati per similarità. Due file simili producono due digest simili. I fuzzy hash sono utili perché sono solitamente non dispendiosi in termini di risorse per calcolarli, ed è immediato compararli.
`sdhash`, descritto originalmente in [13], è stato progettato per essere rapido ed efficiente. sdhash non restituisce sempre risultati ottimali, ma grazie alla sua rapidità può essere utilizzato durante le fasi iniziali di individuazione di file simili.
Per calcolare i similarity digest sdhash internamente utilizza i bloom filters. Questo algoritmo può essere utilizzato principalmente per l'identificazione di frammenti e verificare la correlazione tra versioni di file. L'identificazione di segmenti serve per cercare una porzione di stringa all'interno di un file più grande. Nella correlazione tra versioni di file si è interessati a ricercare in un insieme di documenti quelli simili in dimensioni e contenuto.

TLSH

TLSH è l'acronimo di Trend Micro Locality Sensitive Hash ed è una libreria di *fuzzy matching*. L'algoritmo utilizzato è descritto in [14]. Data una stringa testuale o binaria lunga almeno 50 byte, TLSH genera un *similarity digest* che può essere comparato con *similarity digest* di altre stringhe. L'hash calcolato ha dimensione fissa, ed è lungo 35 byte. TLSH richiede che le stringhe in input abbiano una complessità sufficiente. Una stringa con byte tutti uguali non produce alcun digest.
L'algoritmo utilizzato da TLSH utilizza la tecnica del **locality-sensitive hashing**, che consiste nell'inserire negli stessi bucket con alte probabilità input simili, avendo il numero di bucket molto più piccolo dell'universo dei possibili valori di input. Poiché elementi simili finiscono negli stessi bucket, questa tecnica può essere utilizzata per il raggruppamento degli elementi. Si differenzia dalle tecniche di hashing convenzionali in quanto le collisioni di hash sono massimizzate, non minimizzate.

ssdeep

ssdeep è un altro strumento o libreria per calcolare *similarity digest* a partire da stream di byte. ssdeep è basato su una tecnica chiamata *context triggered piecewise hashes* e come altre tecniche di fuzzy hashing serve per individuare in un insieme di stringhe o file quelli simili fra loro. In particolare, ssdeep è ottimizzato per individuare sequenze di byte identici nello stesso ordine, sebbene i byte tra queste sequenze possano essere diversi sia nel contenuto che nella lunghezza.
La tecnica utilizzata, CTPH, descritta in dettaglio in [15], si basa sull'utilizzo di un *rolling hash*. *Rolling hash* è una funzione di hash che processa l'input tramite una finestra scorrevole sull'input stesso. Alcune funzioni hash permettono il calcolo molto rapido, aggiornando il risultato in base al valore dell'hash nella finestra precedente e in base al valore aggiunto e rimosso dalla finestra all'iterazione corrente. CTPH utilizza i rolling hash per impostare i confini al tradizionale approccio dei *piecewise hashes*.
ssdeep è utilizzato in molte applicazioni, soprattutto nell'ambito dell'individuazione e riconoscimento di malware. Nonostante esistano algoritmi più precisi, ssdeep è uno dei principali strumenti utilizzati perché è veloce.

Nilsimsa Hash

Nilsimsa è un algoritmo di fuzzy hashing della categoria dei locality-sensitive hashing, come TLSH. Nilsimsa è stato proposto inizialmente come algoritmo per realizzare filtri anti-spam, poi è stato rivisto e pubblicato in [16]. L'obiettivo di Nilsimsa è quello di generare un hash digest di un messaggio di posta elettronica in modo tale che i digest di due messaggi simili siano simili tra loro.
Il paper che descrive Nilsimsa indica tre requisiti essenziali che l'algoritmo deve soddisfare:

- il digest che identifica ogni messaggio non dovrebbe variare in modo significativo per i cambiamenti che possono essere introdotti da strumenti automatici;

- la codifica deve essere robusta contro gli attacchi intenzionali;

- la codifica dovrebbe supportare un rischio estremamente basso di falsi positivi.

Effettuando la comparazione con altri algoritmi di fuzzy hashing come TLSH, Ssdeep e Sdhash, si è notato però che Nilsimsa ha un tasso di falsi positivi significativamente più alto rispetto ai precedenti algoritmi.

Byte histogram counting

Tra tutti quelli descritti è l'algoritmo più semplice. Consiste nel creare un istogramma ed incrementare ad ogni occorrenza incontrata in uno stream di byte la colonna corrispondente al byte. L'istogramma sarà formato da 256 colonne, ovvero l'universo possibile di byte.
Se per ogni colonna viene utilizzato un byte come contatore, il digest risultato dell'operazione è lungo 256 byte. Molte colonne saranno vuote, utilizzare quindi un algoritmo di compressione come LZ4 permette di ridurre notevolmente la lunghezza del digest.
Due digest possono essere comparati andando a sommare la differenza di occorrenze per ogni colonna. Più due file sono simili, più la somma delle differenze sarà inferiore. La differenza di due digest di due file differenti è quindi zero.
Questo algoritmo è molto efficace per stringhe o file di piccole dimensioni e che contengono un numero limitato di byte aggiunti o rimossi. Ha pessime prestazioni al contrario con file di grandi dimensioni e funziona solamente se gli input hanno dimensioni simili.

Bloom filter

Genericamente un Bloom filter è una struttura dati probabilistica efficiente in termini di spazio che viene utilizzato per verificare se un elemento è un membro di un set. Un Bloom filter è un array di bit, la cui lunghezza è `m`. All'inizio tutti i bit dell'array sono impostati a zero. Vengono inoltre definite `k` funzioni di hash che mappano ogni input fornito in un indice che corrisponde alla posizione di un bit sull'array del Bloom filter. Le funzioni hash devono generare una distribuzione casuale uniforme.

I Bloom filter possono essere utilizzati per confrontare la similarità tra stringhe testuali o binarie. È necessario stabilire a priori un metodo per suddividere l'intero input in elementi su cui è possibile applicare le funzioni hash. I metodi più utilizzati sono due:

- suddividere l'input in *n-gram*, ovvero spezzare l'input in tanti elementi che hanno la stessa lunghezza, indicata dalla lettera `n`. Lo svantaggio di questo metodo è che se due stringhe sono simili ma non sono allineate, tutti gli *n-gram* saranno diversi e le funzioni di hash produrranno risultati diversi;

- tokenizzare l'input in elementi di lunghezza variabile, utilizzando i caratteri di spaziatura come divisori. Il problema di questo metodo è quello che non può essere utilizzato su stringhe binarie, perché non contengono elementi di separazione noti. Questa tecnica però è resistente al disallineamento dell'input, al contrario di quella precedente.

Per verificare se due stringhe sono simili occorre confrontare i due vettori che l'algoritmo dei Bloom filter ha generato. Ogni volta che una funzione di hash produce un risultato si setta a uno il bit del vettore alla posizione dell'indice ottenuto come risultato. Se confrontando due array la maggior parte dei bit nella stessa posizione corrispondono significa che le stringhe che si stanno comparando sono simili.

Confronto e scelta algoritmi per clustering

Per effettuare un confronto degli algoritmi proposti si sono prima estratte tutte le connessioni TCP da una registrazione del traffico di rete. Il file PCAP in questione conteneva i pacchetti catturati durante una competizione di tipo attacco/difesa effettuata recentemente[2]. Le connessioni TCP presenti nella registrazione di rete si basavano tutti su protocolli diversi, tra cui anche HTTP, SMTP e IMAP. Non erano presenti connessioni cifrate. Per ciascuna connessione individuata sono stati estratti i due flussi unidirezionali e salvati su file differenti attraverso lo strumento `tcpflow`[3].

[2]https://ctf.mhackeroni.it/

[3]https://github.com/simsong/tcpflow

Per calcolare il punteggio per ogni algoritmo proposto è stato realizzato uno strumento per effettuare la prova delle prestazioni. Lo strumento ha lo scopo di leggere da disco i file che contengono ciascuno il contenuto di uno stream e chiamare per tutte le coppie di stream presenti gli algoritmi per il calcolo di similarità sopra riportati.
Lo strumento è stato scritto in linguaggio Go. Per molti algoritmi esisteva già una implementazione in Go, per altri è stato necessario reimplementarli, come nel caso di `sdhash`[4]. Siccome è difficile stabilire una metrica esatta per indicare quanto due stringhe siano simili si è deciso di salvare in un file CSV tutti i punteggi ottenuti per ciascun algoritmo per ciascuna coppia di stream estratti e confrontare i risultati a mano.
Gli algoritmi di fuzzy hashing come Nilsimsa, ssdeep, sdhash e TLSH forniscono già un punteggio di similarità quando vengono comparati i digest di due stringhe. Gli algoritmi ssdeep e sdhash producono un risultato da 0 a 100, dove 0 indica completamente differenti, e 100 completamente uguali. TLSH non ha un range di valori in cui rientra il risultato, ma più la comparazione dei digest produce un risultato che si avvicina allo zero più due file sono simili. I risultati prodotti da Nilsimsa sono invece compresi tra 0 e 128, dove 0 indica che due file sono completamente differenti, e 128 uguali o molto simili.
Il punteggio relativo alla distanza di Levenshtein viene calcolato sottraendo a 1 il risultato ottenuto con la formula riportata sopra. Per i metodi di Byte histogram counting e Bloom filter il punteggio viene calcolato sottraendo a 1 il rapporto tra la distanza dei vettori risultato e la lunghezza massima dei due file confrontati. In tutti e tre gli algoritmi un punteggio che si avvicina a 1 indica che due file sono identici, mentre un punteggio uguale o inferiore a zero indica che due file sono completamente differenti.

Risultati ottenuti

Per confrontare i risultati si è proceduto ad analizzare manualmente i punteggi scritti nel file CSV generato. Ogni riga del file CSV rappresenta un confronto tra due file. In ciascun file è contenuto il flusso unidirezionale di

[4]https://github.com/eciavatta/sdhash

una connessione. Nelle colonne sono indicati i nomi dei due file che vengono confrontati, la loro dimensione, e i punteggi ottenuti per ciascun algoritmo. Di seguito sono elencate alcune considerazioni che sono state rilevate quando si sono analizzati i risultati.

- Tutti gli algoritmi di fuzzy hashing non restituiscono risultato o restituiscono un risultato scorretto quando almeno uno dei due file ha una dimensione limitata. La dimensione minima perché l'algoritmo fornisca risultati soddisfacenti varia da algoritmo ad algoritmo.
  - *ssdeep* e *sdhash* non restituiscono risultato per file con dimensione inferiore a 4 kilobyte. In generale questi algoritmi producono risultati migliori con file di dimensione nettamente superiore a questa soglia.
  - *TLSH* non restituisce un risultato per file di dimensione inferiore a 50 byte, ma produce punteggi soddisfacenti con file di medie e grosse dimensioni.
  - `Nilsimsa` produce un risultato qualsiasi sia la dimensione dei file da confrontare ma produce innumerevoli risultati falsi positivi.

- Si è notato che l'algoritmo descritto in *Byte histogram counting* produce un punteggio molto simile rispetto al risultato prodotto dalla distanza di Levenshtein per file di dimensioni limitate (inferiore a 2kb). La tecnica descritta in *Byte histogram counting* produce però risultati sbagliati quando vengono confrontati file di media o grossa dimensione. Questo è dovuto al fatto che il vettore utilizzato per contare in numero di occorrenze per ogni singolo carattere si satura quando vengono processati molti bytes.

- La distanza di Levenshtein produce risultati scorretti quando gran parte dei due file è identica, ma in uno dei due file da confrontare sono aggiunti blocchi di dati non presenti nell'altro file. In questo speciale caso gli algoritmi di fuzzy hashing producono un risultato che indica similarità, mentre il punteggio ottenuto dalla distanza di Levenshtein non

suggerisce che i due file siano simili. Questo fenomeno è poi accentuato nei file di dimensioni elevate.

- Entrambe le tecniche presentate nella sezione dei Bloom filter non forniscono risultati adeguati e producono punteggi inferiori agli altri algoritmi riportati.

Algoritmi scelti

Analizzando le connessioni contenute nel file PCAP registrato durante una competizione di tipo attacco/difesa si è notato che la maggior parte di queste aveva un contenuto inferiore al kilobyte. Prendendo in considerazione il protocollo HTTP la maggior parte delle richieste e delle risposte delle pagine web ha infatti dimensione limitata. Alcuni stream utilizzati per il download di file statici superavano però nettamente questa soglia, e arrivavano ad avere dimensioni vicine al megabyte.
Si è deciso di scegliere quindi l'algoritmo che ottenesse i punteggi migliori in questa gamma di valori. Siccome nessun algoritmo però ha prodotto risultati soddisfacenti indipendentemente dalla dimensione dei file da confrontare si è scelto di utilizzare due algoritmi e di combinare i risultati ottenuti da questi.
La tecnica che dà risultati migliori nel confronto di file di piccole dimensioni (inferiori al kilobyte) è la distanza di Levenshtein. Siccome però l'algoritmo di *edit distance* è oneroso in termini computazionali si è scelto di utilizzare la tecnica descritta in *Byte histogram counting* che produce risultati simili ma è più facile e veloce da calcolare. Per i file di medie e grosse dimensioni si è scelto l'algoritmo di fuzzy hashing *TLSH*, l'unico che producesse risultati soddisfacenti per file di medie dimensioni (inferiori a 4 kilobyte) e che minimizzasse il numero di falsi positivi.
Per stabilire se due connessioni sono simili è stata definita la seguente strategia:

- se la dimensione massima dei due file è inferiore a 128 byte viene utilizzato soltanto la tecnica *Byte histogram counting* per valutare la similarità;

- se la dimensione massima dei due file è superiore a 4096 byte viene utilizzato soltanto l'algoritmo *TLSH* come metrica di comparazione;

- altrimenti vengono utilizzate entrambe le due tecniche e si considerano simili le connessioni che vengono considerate simili da almeno uno dei due algoritmi.

Procedimento per il raggruppamento

Scelta la metrica per stabilire quando le connessioni sono simili occorre un procedimento per raggruppare le connessioni in set distinti. Ciascun set deve contenere connessioni simili tra loro. Il procedimento utilizzato per il raggruppamento è il seguente:

1. quando si estrae una nuova connessione dal flusso di rete si calcolano i digest di *Byte histogram counting* e *TLSH*;

2. per ciascun set precedentemente individuato si confrontano i digest della nuova connessione con i digest dell'elemento "creatore" del set, tenendo in considerazione la dimensione delle due connessioni da confrontare come indicato nella sezione sopra;

3. al primo set in cui si verifica similarità la connessione viene aggiunta al set individuato e il procedimento termina; se la connessione non appartiene a nessun set viene creato un nuovo set con elemento "creatore" coincidente con la connessione che si sta processando.

Estrazione dei pattern

Individuato un procedimento per raggruppare le connessioni simili occorre poi estrarre un pattern comune che può generare tutte le connessioni presenti nell'insieme, come spiegato precedentemente. In questo modo nel database è possibile memorizzare i pattern che generano le connessioni e per ciascuna connessione memorizzare soltanto le parti che si differenziano dal pattern.
Per l'estrazione di pattern ci si è ricondotti al problema del *longest common subsequence* (LCS), un problema noto in letteratura che serve per trovare la

sottosequenza più lunga comune a tutte le sequenze in un insieme di sequenze. In questo caso specifico l'insieme di sequenze è un set di connessioni, dove ciascuna connessione è una stringa testuale o binaria. La sottosequenza più lunga non necessariamente deve contenere caratteri tutti adiacenti fra loro: esiste un altro problema che risolve questo caso che è il longest common substring, che però è meno efficace rispetto al problema corrente.

Il problema LCS è un problema NP-hard quando il numero di sequenze in input è arbitrario, mentre quando il numero di sequenze è costante il problema è risolvibile in tempo polinomiale mediante la programmazione dinamica. Il numero di connessioni simili da individuare prima di eseguire l'algoritmo LCS può essere definito e stabilito a priori per rendere l'algoritmo risolvibile in tempo polinomiale.

Il risultato dell'algoritmo LCS è il pattern, ovvero la sottostringa presente in tutte le connessioni che sono state utilizzate come input dell'algoritmo. Le parti che differiscono tra il pattern e una delle connessioni dell'insieme vengono definiti *placeholder* e contengono le parti di stringa univoche non contenute nel pattern. Quando la connessione coincide con il pattern non sono presenti *placeholder*, mentre quando differisce in più punti sono presenti più *placeholder*. Nella figura 5.1 è presente un esempio in cui è mostrato come l'algoritmo LCS deve generare i pattern e i *placeholder* date tre stringhe simili in input.

```
LOGIN S9QMwQGU612X@ctf.mhackeroni.it "okXIdS1V9rdD" <- stringa 1
LOGIN ADkh89mdQm36@ctf.mhackeroni.it "hnjKsLWdDG2n" <- stringa 2
LOGIN i7GDgWGih8ff@ctf.mhackeroni.it "hiI9Nd7wNAaM" <- stringa 3
======!!!!!!!!!!!!====================!!!!!!!!!!!!=
LOGIN             @ctf.mhackeroni.it "            " <- pattern
      S9QMwQGU612X                     okXIdS1V9rdD  <- placeholder 1
      ADkh89mdQm36                     hnjKsLWdDG2n  <- placeholder 2
      i7GDgWGih8ff                     hiI9Nd7wNAaM  <- placeholder 3
```

Figura 5.1: Risultato di estrazione del pattern e dei *placeholder* quando viene eseguito l'algoritmo LCS.

Nell'esempio riportato in figura 5.1 per motivi di semplicità tutte le stringhe in input sono allineate, ovvero i *placeholder* hanno tutti lunghezza uguale. La maggior parte delle volte questo non succede, ma l'algoritmo LCS è in grado, comunque, di estrarre un pattern comune e generare i *placeholder*, anche se questi hanno lunghezza diversa.
Per una questione di ottimizzazione e per ridurre il numero di *placeholder* si evita di considerare come parti in comune le sottostringhe composte da caratteri adiacenti uguali di lunghezza inferiori a una costante prestabilita.
Il pattern viene memorizzato nel database in una collezione separata, mentre per ogni connessione viene inserito il riferimento al pattern da cui viene generata e l'insieme di *placeholder* che contengono le sottostringhe da cui il contenuto della connessione differisce dal pattern. Associato a ciascun *placeholder* viene anche memorizzato un indice in riferimento al pattern che indica in quale posizione del pattern deve essere inserito il contenuto del *placeholder*. Per ricostruire il contenuto di una connessione occorre quindi ottenere il pattern e inserire in questo i *placeholder* nella posizione stabilita da ciascun indice.

Valutazione delle performance

Per valutare l'efficacia del metodo proposto si è realizzato un prototipo in linguaggio Python. Il prototipo implementa entrambe le due funzionalità alla base dell'indicizzazione per differenza, ovvero il raggruppamento delle connessioni per similarità attraverso gli algoritmi di fuzzy hashing e l'individuazione di un pattern comune attraverso l'algoritmo LCS.
Il prototipo è stato realizzato per poter confrontare l'efficacia del metodo di memorizzazione e indicizzazione proposto con altri metodi, come quello della compressione delle connessioni. Il prototipo non è stato realizzato per misurare l'efficienza di questo metodo, e gli algoritmi implementati non sono stati ottimizzati per misurare le performance in termini di risorse computazionali.
Come metrica principale è stata utilizzata la dimensione dell'output generato dai veri metodi di memorizzazione. Come campione di confronto è stato utilizzato un PCAP con del traffico di rete catturato durante una competizione di tipo attacco/difesa.

Il metodo proposto, l'indicizzazione per differenza, è stato confrontato con altri quattro metodi: la memorizzazione dei segmenti in formato grezzo, la memorizzazione delle connessioni ricostruite (come avviene ora) e la memorizzazione del contenuto delle connessioni compresso sia con algoritmo LZ4, sia con algoritmo GZIP. La figura 5.2, che mostra i risultati ottenuti, evidenza che il metodo proposto è quello che richiede uno spazio di memorizzazione inferiore e supera i livelli degli algoritmi di compressione più evoluti.

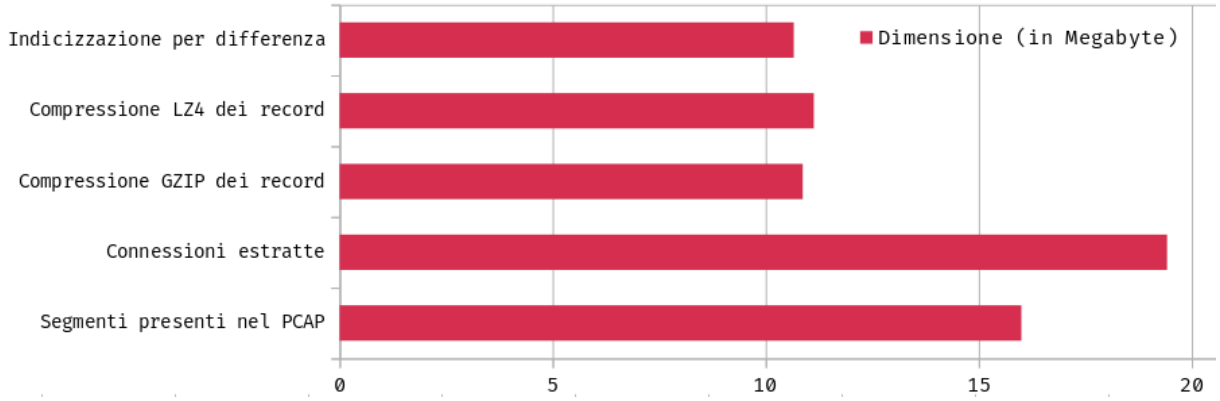

Figura 5.2: Grafico che mostra il confronto delle dimensioni delle tecniche di memorizzazione prese in considerazione.

Ottimizzazione della ricerca

Il metodo più semplice per effettuare la ricerca nel database indicizzato per differenza è quello di ricostruire prima della ricerca ogni singola connessione, inserendo il contenuto dei *placeholder* all'interno dei pattern. Questo procedimento è però lento e non ha vantaggi rispetto alla normale ricerca sequenziale per stringa o per espressione regolare nel database.
Un metodo più efficace consiste nell'effettuare la ricerca prima nel pattern, per verificare se si incontrano occorrenze già in quello per saltare la ricerca in tutte le connessioni generate dal pattern individuato. Se non ci sono *match* nelle parti in comune occorre cercare nel contenuto dei *placeholder* di ciascuna connessione. Ai *placeholder* possono essere inseriti dei vincoli, come quelli basati sulla lunghezza, per poter velocizzare la ricerca saltando la verifica dei *placeholder* incompatibili.

Il procedimento in realtà è molto più complesso di quello appena descritto, perché un'occorrenza può trovarsi a cavallo tra il pattern e un *placeholder*. Occorrono quindi tecniche più elaborate per verificare la compatibilità di una ricerca con un *placeholder* che non sono state approfondite in questo progetto di tesi.

5.2 Tracciamento dei dispositivi

Per tracciamento di dispositivi si intende individuare da quale dispositivo o client una connessione è stata iniziata. L'obiettivo non è quello di riconoscere o dare un'identità al dispositivo che ha effettuato una richiesta, bensì riuscire ad individuare tutte le connessioni che provengono dallo stesso client.

In un normale contesto di rete è più facile individuare e raggruppare le richieste che vengono effettuate per client che le ha effettuate. Senza considerare eventuali tecniche che offuschino o mascherino l'indirizzo del mittente, tramite l'indirizzo IP è possibile individuare tutte le connessioni effettuate dallo stesso client. Durante una competizione di tipo attacco/difesa solitamente gli indirizzi dei client dai quali provengono le richieste vengono riscritti utilizzando NAT, una tecnica che consiste nel modificare gli indirizzi IP contenuti negli header dei pacchetti in transito. Un router modifica quindi l'indirizzo IP dei client che effettuano le richieste e lo sostituiscono con quello del router stesso. Guardando gli indirizzi IP dei pacchetti, che vengono ricevuti dalla macchina vulnerabile e che sono analizzati dallo strumento, diventa quindi impossibile distinguere da quale client sono state effettuate le richieste.

La riscrittura degli indirizzi IP tramite NAT per nascondere l'indirizzo IP del client di provenienza viene effettuata per impedire l'utilizzo di firewall per bloccare gli indirizzi IP dei team nemici e permettere soltanto gli indirizzi del game master, il servizio che si occupa di inserire le flag nella macchina vulnerabile e di controllare che i servizi siano attivi e funzionanti.

Per effettuare il tracciamento dei dispositivi occorre quindi una tecnica diversa dal controllo degli indirizzi IP sorgente. Una tecnica proposta che verrà descritta di seguito è quella di utilizzare i valori contenuti nel campo Options nei record di tipo Timestamp del protocollo TCP. Tramite i timestamp

contenuti nei segmenti è possibile tentare di individuare tutte le connessioni effettuate dallo stesso client.

Tracciamento con le opzioni Timestamp di TCP

Il protocollo TCP definisce nella struttura dei segmenti un campo opzioni in cui possono essere inserite informazioni aggiuntive facoltative. Nel campo `Options` le informazioni sono inserite sotto forma di record. Ciascun record ha il primo byte che indica la tipologia, il secondo byte che indica la lunghezza del record, e i restanti byte variano in base alla tipologia di record indicata.

Esiste un tipo di record per indicare i timestamp (tipo=8). Il record Timestamp nelle opzioni TCP è stato inserito dall'RFC1323[5] intitolato "TCP Extensions for High Performance", che però è stato sostituito più recentemente dall'RFC 7323[6]. Come suggerisce il titolo degli RFC l'opzione Timestamp è stata aggiunta per migliorare le performance del protocollo TCP e per rendere il protocollo più stabile ed efficiente.

TCP è un protocollo simmetrico, che consente di inviare i dati in qualsiasi momento in entrambe le direzioni. Il record di tipo Timestamp consente di associare al segmento due timestamp: il `Timestamp Value` (TSval) e il `Timestamp Echo Reply` (TSecr). Data la simmetria del protocollo, entrambe le parti della connessione inseriscono e calcolano i timestamp dei propri segmenti in modo indipendente.

Il campo `Timestamp Value` serve per specificare il valore corrente del timestamp di TCP relativo all'invio di un segmento. Entrambe le parti di una connessione specificano questo valore, sia nei segmenti con dati, sia nei segmenti con solo ACK. Il campo `Timestamp Echo Reply` serve per fare eco dei valori di `Timestamp Value`. Ciascuna parte della connessione inserisce quindi in questo campo il TSval della controparte. Per efficienza TCP combina i due campi in un'unica opzione TCP Timestamp, anche se i due timestamp si riferiscono sempre a segmenti diversi e non hanno significato se combinati insieme.

[5] https://tools.ietf.org/html/rfc1323
[6] https://tools.ietf.org/html/rfc7323

Di seguito è mostrata la struttura di un record TCP Timestamp. Entrambi i timestamp hanno una dimensione fissa di 4 byte, e sono specificati immediatamente dopo la tipologia dell'opzione (primo byte) e alla lunghezza del record (secondo byte).

```
+-------+-------+--------------------+--------------------+
|Kind=8 |  10   |   TS Value (TSval) |TS Echo Reply (TSecr)|
+-------+-------+--------------------+--------------------+
  1       1              4                     4
```

Il campo TSecr è valido solo se il bit ACK è impostato nell'header TCP. Se il bit ACK del segmento inviato non è impostato nell'header TCP, il mittente di quel segmento deve impostare il campo TSecr a zero. Quando il bit ACK è impostato in un segmento da inviare, il mittente deve inserire nel campo TSval il valore più recente di TSval ricevuto dalla controparte. Quando il bit ACK non è impostato, il ricevitore deve ignorare il valore del campo TSecr.

Scopo e utilizzo delle opzioni Timestamp

L'opzione Timestamp è stata introdotta per permettere di misurare agilmente i tempi che intercorrono tra l'invio e la ricezione dei pacchetti, ma è utilizzata anche per risolvere altri problemi. È utilizzata per misurare il round-trip time (RTT) di ogni singolo segmento che è stato confermato attraverso un ACK. Il round-trip time è il tempo che intercorre tra l'invio di un pacchetto e la ricezione della conferma del pacchetto stesso. Ciascuna parte di una connessione può calcolare l'RTT sottraendo dal valore del contatore del timestamp corrente il valore del timestamp contenuto in TSecr in un segmento ricevuto che è stato confermato con un ACK.
Un calcolo accurato dell'RTT è importante perché aiuta a stabilire quale dovrebbe essere il valore di timeout di ritrasmissione (RTO). L'RTO è responsabile della ritrasmissione dei pacchetti che non sono stati confermati dopo un certo periodo di tempo.
L'opzione Timestamp TCP è anche utilizzata in un algoritmo conosciuto come `Protection Against Wrapped Sequence numbers` (PAWS). PAWS viene utilizzato quando la finestra di ricezione di TCP raggiunge il limite e deve riniziare da capo, resettando il numero di sequenza.

Anche l'algoritmo di rilevamento Eifel, descritto nell'RFC 3522[7], utilizza timestamp TCP per determinare se una ritrasmissione di un segmento è dovuta a un pacchetto perso o a un pacchetto fuori ordine.

Utilizzi malevoli delle opzioni Timestamp

Le specifiche di implementazione delle opzioni TCP Timestamp non indicano a cosa deve corrispondere il timestamp da utilizzare nel campo TSval. Non indicano neppure ogni quanto il timestamp deve essere incrementato, ma è specificato solamente che i tick dovrebbero essere incrementati in intervalli regolari. Il timestamp deve essere quindi incrementato proporzionalmente. La maggior parte dei sistemi operativi inizializza il timestamp all'avvio di ogni connessione in maniera pseudo casuale, e lo incrementa a ogni millisecondo. In alcuni sistemi operativi, soprattutto nei più vecchi, il timestamp da utilizzare come TSval viene inizializzato a zero ogni volta che il sistema viene avviato. In questo modo è possibile calcolare facilmente l'uptime di un host, ovvero da quanto tempo il sistema è acceso. Oltre a questa informazione è possibile anche tracciare agevolmente i dispositivi che inizializzano i timestamp in questo modo. Prese due connessioni da una registrazione di rete, si individua in ciascuna connessione un segmento che abbia TSval impostato e lo si sottrae dall'istante di tempo in cui il segmento è stato inviato. Se i risultati delle due operazioni si assomigliano entro un certo margine di errore, è molto probabile che le due connessioni siano state effettuate dallo stesso host.
Nei sistemi operativi bastati su Linux è possibile configurare il parametro `net.ipv4.tcp_timestamps` per scegliere il comportamento dei timestamp. Sono possibili tre valori:

- **0**: i timestamp sono disabilitati e non viene inserita l'opzione TCP Timestamp nei segmenti;

- **1**: i timestamp sono abilitati come definito nell'RFC 1323. I timestamp sono calcolati sommando al tempo corrente un offset casuale. Di seguito è spiegato meglio come sono stati implementati;

[7] https://tools.ietf.org/html/rfc3522

- **2**: come nell'opzione 1, ma senza offset casuale.

L'opzione 1 è quella abilitata di default.

Implementazione dei TCP Timestamp nel kernel Linux

Per impostazione predefinita nei sistemi operativi Linux, quelli maggiormente utilizzati nelle competizioni di tipo attacco/difesa, i timestamp sono abilitati e sono calcolati sommando al tempo corrente un offset casuale. L'offset casuale viene scelto ogni volta che una nuova connessione viene inizializzata attraverso la funzione `secure_tcp_ts_off`, implementata nel file `net/core/secure_seq.c` nel sorgente Linux[8], riportata nel listato 5.1.
La funzione restituisce un offset pari a zero nel caso in cui l'opzione 1 non sia abilitata. Altrimenti chiama la funzione `ts_secret_init` che genera un segreto sicuro, come è possibile vedere nel listato 5.2. Il segreto viene generato soltanto la prima volta, e riutilizzato per tutte le connessioni che saranno effettuate. L'offset viene generato utilizzando la funzione `siphash_2u32`, una funzione pseudocasuale della famiglia *add–rotate–xor*. L'offset è creato in funzione dell'indirizzo IP sorgente di una connessione, l'indirizzo IP destinazione di una connessione, e il segreto precedentemente generato.

Listing 5.1: Implementazione della funzione che genera offset casuali.

```
u32 secure_tcp_ts_off(const struct net *net, __be32 saddr,
                      __be32 daddr)
{
        if (net->ipv4.sysctl_tcp_timestamps != 1)
                return 0;

        ts_secret_init();
        return siphash_2u32((__force u32)saddr,
                            (__force u32)daddr,
                                    &ts_secret);
}
```

[8]https://elixir.bootlin.com/linux/v5.11.14/source/net/core/secure_seq.c#L117

Listing 5.2: Implementazione della funzione che inizializza il segreto.

```
static __always_inline void ts_secret_init(void)
{
        net_get_random_once(&ts_secret, sizeof(ts_secret));
}
```

Sfruttare l'offset del timestamp fisso per il tracciamento

Date le considerazioni presentate nella sezione precedente, tutte le connessioni provenienti dallo stesso host avranno lo stesso offset, perché la coppia (indirizzo sorgente, indirizzo destinazione) rimane sempre la stessa e il segreto non cambia tra una connessione e l'altra. Questo significa quindi che se le opzioni Timestamp di TCP non vengono rimosse dai segmenti da un router durante le competizioni, è possibile tracciare tutti gli host che effettuano le connessioni per identificare gli avversari.
Il metodo più efficace per individuare se una connessione è stata effettuata da un host già conosciuto, e nel caso individuare tutte le altre connessioni che ha effettuato, è il seguente:

1. inizializzare un dizionario che ha come chiave l'offset dei timestamp e come valore un identificativo dell'host;

2. quando si riceve una nuova connessione si preleva un segmento e si calcola l'offset del timestamp, sottraendo dal valore del campo TSval del segmento il timestamp dell'orario attuale;

3. se nel dizionario è già presente un offset che è vicino a quello calcolato entro un margine ristretto di errore, si associa alla connessione l'identificativo dell'host trovato;

4. altrimenti si aggiunge al dizionario l'offset calcolato e si genera un nuovo identificativo per il nuovo host individuato.

Conclusioni

Nonostante sia ancora in fase embrionale lo strumento si è rilevato fondamentale nelle competizioni di tipo attacco/difesa alle quali ho partecipato recentemente con i team di cui faccio parte, *CeSeNA* e *Ulisse*. Queste competizioni sono state fondamentali per verificare le prestazioni dello strumento, perché la quantità di traffico ricevuto durante una prova di tipo attacco/difesa è elevato ed è un compito difficile individuare tra le migliaia di richieste effettuate ogni minuto quelle che contengono attacchi malevoli.
Lo strumento ha sempre mantenuto ottime prestazioni durante tutto lo sviluppo delle competizioni. Anche alla fine delle competizioni, quando nel database è memorizzato e indicizzato tutto il traffico ricevuto durante le gare, lo strumento si è rivelato reattivo e non ha avuto cali di prestazioni.
L'interfaccia grafica semplice, pulita e intuitiva permette anche agli utenti meno esperti di utilizzare lo strumento senza difficoltà. L'interfaccia web condivisa permette inoltre a tutti i membri del team, non solo quelli in difesa, di individuare gli attacchi e capire come gli avversari hanno sfruttato le vulnerabilità per esfiltrare le flag.

Da quando lo strumento è stato rilasciato con licenza open source GPL-3.0 su Github esso ha ricevuto diverse attenzioni dalla comunità legata alle competizioni di sicurezza informatica. Numerosi team che partecipano ad eventi di *Capture The Flag* hanno adottato lo strumento nella loro infrastruttura e lo usano regolarmente per analizzare il traffico di rete durante le competizioni di tipo attacco/difesa. Il progetto su Github ha già diversi contributori ed è stato pubblicato su testate che si occupano di sicurezza informatica come *The Hacker News*[9].

[9]https://twitter.com/TheHackersNews/status/1380198653994823681

Sviluppi futuri

Oltre alle proposte sperimentali riportate nel Capitolo 5 sugli aspetti avanzati, che sono state implementate ma non ancora integrate nello strumento, ci sono innumerevoli funzionalità che possono essere implementate. Molte delle funzionalità da inserire sono state proposte dalla comunità open source o ricevute attraverso feedback dai membri di altri team che hanno utilizzato lo strumento.
Di seguito è riportato un elenco di funzionalità e miglioramenti che possono essere adottati:

- aggiungere i parser per estrarre informazioni da altri protocolli, come i protocolli FTP, SMTP e IMAP;

- migliorare il meccanismo delle regole, permettendo di individuare ed etichettare le connessioni non solo utilizzando espressioni regolari o tramite la ricerca per stringhe, ma anche effettuando pattern matching a un più alto livello, a partire dai metadati estratti dal protocollo di ciascuna connessione;

- permettere di effettuare il pattern matching e visualizzare le connessioni cifrate con RSA specificando la chiave di decifrazione;

- migliorare il sistema di notifiche quando vengono individuate le connessioni, anche attraverso l'integrazione di servizi esterni;

- permettere l'esportazione e l'importazione delle regole in formati compatibili per altri IDS o IPS;

- implementare funzionalità di prevenzione per bloccare le connessioni malevoli individuate;

- migliorare l'esportazione degli exploit utilizzando tecniche più avanzate per individuare le vulnerabilità presenti nelle connessioni.

Ringraziamenti

Giunto al termine di questo percorso di studio non posso che ringraziare i miei genitori, che mi hanno dato l'opportunità di compierlo e che mi hanno sempre sostenuto con affetto, standomi vicino e cercando di supportarmi anche e soprattutto nei momenti più difficili.
Così come mi ha sopportato mio fratello, che ringrazio di cuore.

Un pensiero ai miei nonni, che non ho potuto vedere in questo triste periodo di isolamento, ma che si sono interessati chiedendomi a ogni telefonata notizie dei miei studi.

Ci tengo a ringraziare i miei amici di sempre, con i quali da tanto tempo ormai ci siamo potuti sentire solo virtualmente. Torneremo a divertirci e ad abbracciarci non appena ci sarà concesso.

Ringrazio i miei compagni di corso, con cui ho condiviso sofferenze e soddisfazioni. Mi avete fatto arrabbiare, mi avete fatto piangere, ma ci siete sempre stati e con voi ho passato i momenti più belli. Abbiamo creato un legame indistruttibile, siete la cosa più preziosa che questi due anni di studi mi hanno regalato.

Un immenso grazie anche ai ragazzi di YoungAlanTuring, il gruppo giovani dell'Arcigay di Rimini, che ho il piacere di coordinare. Siete la mia seconda famiglia, mi avete insegnato tanto e da voi non smetterò mai di imparare. In questo mondo ancora pieno di pregiudizi e discriminazioni mi fate sentire meno solo. Torneremo a vederci presto, per continuare con quell'impegno civile che negli ultimi anni ha arricchito la mia vita.

Un doveroso ringraziamento va ai ragazzi di Ulisse, il laboratorio di cybersecurity di cui faccio parte. Grazie al prof. Franco Callegati, al prof. Marco

Prandini e ad Andrea e Davide che mi hanno introdotto in questo mondo e che mi hanno dato l'opportunità di scrivere questa tesi. Grazie anche ai ragazzi del CeSeNA, l'altra squadra di cui faccio parte. Siete tutti fantastici, non potrò mai dimenticare tutti i momenti passati insieme a cercare di risolvere i problemi più impossibili in qualsiasi ora del giorno e della notte durante le competizioni che abbiamo fatto insieme.

Bibliografia

[1] K Scarfone e P Mell. "NIST guide to intrusion detection and prevention systems (idps)". In: *National Institute of Standards and Technology, Special Publication* (2007), pp. 800–94.

[2] Hung-Jen Liao et al. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24.

[3] Dan Boneh. "The decision diffie-hellman problem". In: *International Algorithmic Number Theory Symposium*. Springer. 1998, pp. 48–63.

[4] Basem Shihada e Pin-Han Ho. "Transport control protocol in optical burst switched networks: issues, solutions, and challenges". In: *IEEE Communications Surveys & Tutorials* 10.2 (2008), pp. 70–86.

[5] Ruan Yuan et al. "Robust tcp reassembly with a hardware-based solution for backbone traffic". In: *2010 IEEE Fifth International Conference on Networking, Architecture, and Storage*. IEEE. 2010, pp. 439–447.

[6] Yutaka Sugawara, Mary Inaba e Kei Hiraki. "High-speed and memory efficient TCP stream scanning using FPGA". In: *International Conference on Field Programmable Logic and Applications, 2005.* IEEE. 2005, pp. 45–50.

[7] David Vincent Schuehler. "Techniques for processing TCP/IP flow content in network switches at gigabit line rates". In: (2004).

[8] S Dharmapurikar e V Paxson. "Robust TCP reassembly in the presence of adversaries". In: *Proceedings of 14th USENIX Security Symposium*, pp. 65–80.

[9] Xiang Wang et al. “Hyperscan: a fast multi-pattern regex matcher for modern cpus”. In: *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 2019, pp. 631–648.

[10] Jianwen Wei et al. “Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis”. In: *2011 International Conference on Cloud and Service Computing*. IEEE. 2011, pp. 354–359.

[11] J. Ziv e A. Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343. DOI: 10.1109/TIT.1977.1055714.

[12] Jacob Ziv e Abraham Lempel. *Compression of Individual Sequences via Variable-Rate Coding*. 1978.

[13] Frank Breitinger, Harald Baier e Jesse Beckingham. “Security and implementation analysis of the similarity digest sdhash”. In: *First international baltic conference on network security & forensics (nesefo)*. 2012.

[14] Jonathan Oliver, Chun Cheng e Yanggui Chen. “TLSH–a locality sensitive hash”. In: *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. IEEE. 2013, pp. 7–13.

[15] Jesse Kornblum. “Identifying almost identical files using context triggered piecewise hashing”. In: *Digital Investigation* 3 (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS ’06), pp. 91–97. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2006.06.015. URL: https://www.sciencedirect.com/science/article/pii/S1742287606000764.

[16] Ernesto Damiani et al. “An Open Digest-based Technique for Spam Detection.” In: *ISCA PDCS* 2004 (2004), pp. 559–564.

Elenco delle figure