Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Aplikácia na monitorovanie sieťovej prevádzky GNU/Linux servera

Bakalárska práca

2023
Jakub Murin

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Aplikácia na monitorovanie sieťovej prevádzky GNU/Linux servera

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Marek Nagy, PhD.

Bratislava, 2023
Jakub Murin

15187893

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jakub Murin
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Aplikácia na monitorovanie sieťovej prevádzky GNU/Linux servera
Application for Network Traffic Monitoring of GNU/Linux Server

Anotácia: Aplikácia sa spustí na monitorovanom GNU/Linux serveri a prostredníctvom webového rozhrania bude poskytovať v reálnom čase informácie o sieťovej prevádzke servera. Okrem štatistických výstupov bude aplikácia umožňovať aj blokovanie nežiadúcej sieťovej aktivity. Dôležitým prvkom je, aby aplikácia svojou činnosťou nespomaľovala sieťovú prevádzku.

Cieľ: Cieľom je vytvoriť aplikáciu pomocou nástroja Node.js a vhodných jeho modulov. Využije sa programovací jazyk JavaScript.

Vedúci: RNDr. Marek Nagy, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 29.06.2022

Dátum schválenia: 17.10.2022 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcem sa vrúcne poďakovať svojmu školiteľovi RNDr. Marekovi Nagyovi, PhD. za ochotu, pomoc, cenné rady, ľudský prístup a celkovo za vedenie a obetovaný čas.

Abstrakt

Bezpečie a ochrana sú neoddeliteľnou súčasťou našej existencie. Avšak v dnešnej dobe sme konfrontovaní s novým druhom hrozieb - hrozbami v digitálnom svete. Je dôležité mať dostupné nástroje, ktoré umožňujú sledovať sieťovú prevádzku a analyzovať ju, aby sme mohli identifikovať potenciálne hrozby a chrániť naše systémy. Naša práca sa zameriava na vytvorenie aplikácie, ktorá umožní administrátorovi jednoduché monitorovanie a analýzu prichádzajúcej sieťovej prevádzky na GNU/Linux serveri. Výsledná aplikácia poskytuje webové rozhranie na zobrazenie štatistiky a poskytuje možnosť blokovania nežiadúcej aktivity.

Kľúčové slová: GNU/Linux server, sieťová prevádzka, monitorovanie, firewall

Abstract

Safety and security are an main part of our existence. However, nowadays we are confronted with a new kind of threats - threats in the digital world. It is important to have tools to monitor and analyze network traffic in order to identify potential threats and protect our systems. Our thesis focuses on creating an application that will allow an administrator to easily monitor and analyze incoming network traffic on a GNU/Linux server. The resulting application provides a web interface for viewing statistics and provides the ability to block unwanted activity.

Keywords: GNU/Linux server, network traffic, monitoring, firewall

Obsah

Zoznam obrázkov

Zoznam ukážok kódu

Úvod

Už od počiatkov ľudstva je potreba sa chrániť pred vonkajšími vplyvmi. Či už pred predátormi alebo pred počasím. Človek rýchlo pochopil, že na ochranu treba nastaviť pred to bariéru. Najskôr sa skrýval do jaskýň, kde ho chránili masívne skaly a neskôr si začal stavať obydlia so stenami a strechou. Časom však prišli aj oveľa nebezpečnejšie hrozby, a to ľudia sebe navzájom. Túžba zisku a moci spôsobovala vzájomné útoky a zlomyseľnosť. A tak človek začal stavať opevnenia a hrady, aby mal bezpečie a mohol sa chrániť. Táto ľudská vlastnosť pretrvala a potreba aj preventívnej ochrany sa ukázala ako nevyhnutná.

Podobne sa aj v informatike snažíme chrániť počítač aj server. Je potrebné postaviť ochranu medzi našim vlastníctvom a ostatným svetom. Tento krát však nie fyzicky, ale vo virtuálnom priestore. Zvlášť v dnešnom digitálnom veku, keď sa bezpečnosť a monitorovanie siete stali kritickými aspektmi udržiavania efektívneho výpočtového prostredia. Ako sa organizácie čoraz viac spoliehajú na servery GNU/Linux pre svoju infraštruktúru, potreba efektívnych nástrojov na monitorovanie sieťovej prevádzky exponenciálne vzrástla. Spolu s tým narástli aj útoky, hoci takto vo virtuálnom svete, lebo informácie a dáta v súčasnosti majú cenu zlata.

Preto sa v nasej práci budeme venovať vytvoreniu aplikácie na sledovanie prichádzajúcej sieťovej činnosti. Jej následnou analýzou a možnosťou blokovania, budeme administrátorovi túto činnosť uľahčovať.

V našej práci sa najskôr v prvej kapitole budeme venovať firewallom v Linuxe. Následne si v druhej kapitole rozoberieme už existujúce monitorovacie nástroje a pozrieme sa na uľahčenie činnosti administrátora hľadaním podobností v sieťovej prevádzke. V tretej kapitole sa budeme venovať návrhu aplikácie s využitím vytvorených modulov. Rozhodneme, ktorú editačnú vzdialenosť implementujeme na porovnanie a rozdelíme aplikáciu na viacero komponentov. Ďalej navrhneme databázu a tabuľky v nej. Na záver si opíšeme problémy, ktoré nastali pri implementácii, ako sme ich vyriešili a predstavíme si výslednú aplikáciu s postupom inštalácie a testovaním. V prílohe sa potom nachádza popis zdrojových súborov aplikácie.

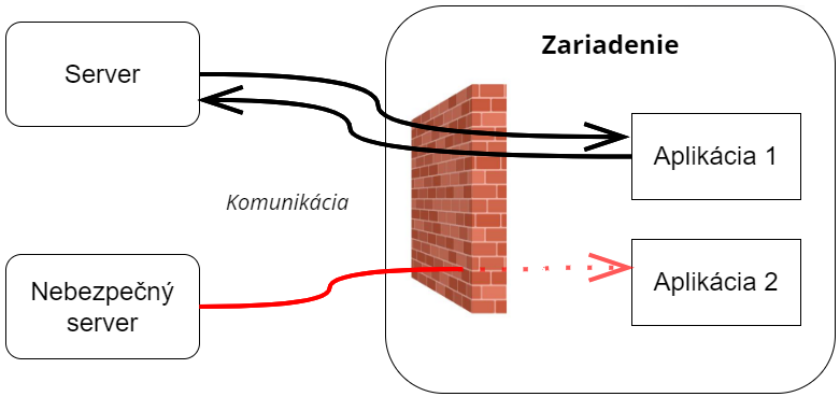
Kapitola 1

Princípy firewallu v Linuxe

Majme server poskytujúci nejakú službu na internete. Prístup k nemu majú všetci používatelia nezávisle na úmysle. Všade sa však nájdu aj takí, čo sa snažia všetko využiť vo svoj prospech alebo uškodiť druhým. Títo útočníci sa snažia použiť každú medzeru na preniknutie do systému. Riešením je odpojenie sa od internetu, avšak tak by sa zrušila aj požadovaná činnosť. Vhodnejšie je zabezpečiť sa voči takémuto správaniu. Preto prišli v 80. rokoch 20. storočia americké spoločnosti Cisco Systems a Digital Equipment Corporation s prvými firewallmi [9]. Tieto systémy posudzovali pakety na základe odosielateľa, cieľa a typu spojenia. Tým položili základ budúcich firewallov.

Firewall je bezpečnostný systém na sledovanie systémovej činnosti a sieťovej prevádzky. Na základe definovaných pravidiel sa snaží identifikovať nežiadúcu činnosť, ktorú zablokuje a zvyšné akcie umožní vykonať, viď obrázok 1.1. Pre sieťovú časť monitoruje vstupné a výstupné pakety. Delí sa na dve časti na základe použitia: sieťovo zameraný a používateľsky zameraný. Sieťovo zameraný firewall sa používa na chránenie LAN (local area network) voči WAN (wide area network). Používateľsky zameraný sa orientuje na konkrétneho používateľa alebo konkrétny systém. V našej práci sa budeme zaoberať týmto druhým typom firewallu.

V tejto kapitole opíšeme niektoré možnosti na spravovanie sieťovej činnosti, ktoré existujú pre operačný systém Linux.



Obr. 1.1: Funkcia firewallu

1.1 Iptables

Iptables [2] sa používajú na nastavenie, údržbu a kontrolu tabuľky pravidiel na filtrovanie paketov. Sú súčasťou linuxového subsystému netfiler, ktorý umožňuje vytváranie kernel modulov na správu sieťovej prevádzky. Najčastejšie využitie je NAT a podpora pre firewall. Pakety sú spracovávané postupne prechádzajúc pravidlá v reťazi, čo je zoradený list pravidiel. Pravidlo rozhoduje o tom čo spraviť so zhodujúcim sa paketom, či ide do stavu REJECT (paket sa zahodí s odoslaním zamietnutia), DROP (paket sa zahodí v tichosti) alebo ACCEPT (akceptuje sa a prepustí). Jednotlivé pravidlá môžu obsahovať skok na inú reťaz, kde pokračuje v prechádzaní pravidiel. Každý paket prejde aspoň jednou reťazou.

Každý protokol má svoj vlastný modul:

- IPv4 - iptables
- IPv6 - ip6tables
- ARP - arptables
- Ethernet frames - ebtables

Boli pridané do 2.4.0 Linux kernel v roku 2001 a v súčasnosti sú stále využívané. Vďaka svojej jednoduchosti, rozmanitosti parametrov filtrovania a stabilite poskytujú administrátorom spoľahlivé riešenie. Skrze svoju dlhovekosť veľa systémových správcov a sieťových inžinierov je oboznámených so syntaxov a používaním, čo robí iptables atraktívne aj pre firmy.

Ukážka kódu 1.1: Príkaz pridávajúci pravidlo do iptables

```
$ sudo iptables -A INPUT -s 168.63.129.16 -j DROP
```

Príklad príkazu možno vidieť na ukážke kódu 1.1 na pridanie pravidla na koniec reťaze *INPUT*, ktorá spracováva príchodzie pakety. Zahadzuje pakety z adresy 168.63.129.16. Keďže ide o citlivé nastavenia, na vykonanie príkazu potrebujeme administrátorské práva.

1.2 Nftables

Je to framework od Netfilter Project [13], ktorý slúži na filtrovanie paketov, poskytuje network address translation (NAT) a triedenie paketov. V kernely ho môžeme nájsť ako modul nf_tables. Sú nástupcom iptables a vylepšujú nedostatky tohto modulu. Fungujú na takzvanom in-kernel virtal machine, ktorý spúšťa bytecode na prehľadávanie paketov a získavanie informácii z nich. Podporujú aj multi-threading a paralelizmus, čo značne zvyšuje výkon pri frekventovanej sieťovej prevádzke. Pre ovládanie z príkazového riadku obsahuje nástroj nft, ktorý používa vysoko-úrovňový jazyk na definovanie a úpravu pravidiel. Vďaka tomu sú čitateľnejšie a viac sa podobajú na programovací jazyk.

Na zefektívnenie kontroly pravidiel a zjednodušenie syntaxe sú v nftables zavedené množiny a slovníky, ktoré sú ukladané do hešovacích tabuliek a červeno-čiernych stromov. Obe dátové štruktúry môžu byť definované anonymne priamo v pravidle, alebo pomenované a v pravidle sú ako premenná. To umožňuje používateľovi vytvoriť jedno pravidlo pre viacero rôznych scenárov. Kód sa tak stáva prehľadnejší a čitateľnejší. Slovníky môžu byť aj typu *verdict maps*, teda slovníky obsahujúce ako hodnotu príkaz. Súčasťou syntaxe príkazov sú aj operácie *and* a *or*.

Nftables boli vydané 19. januára 2014 a pripojené do Linux kernel mainline vo verzii 3.13.

Ukážka kódu 1.2: Príkaz pridávajúci pravidlo do nftables

```
$ sudo nft add rule inet filter input \
ip saddr 168.63.129.16 drop
```

Na ukážke kódu 1.2 je príkaz na pridanie pravidla na koniec reťaze *input* do tabuľky *filter* pre *inet* (spoločné pre IPv4 a IPv6). Zahadzuje pakety z adresy 168.63.129.16. Keďže ide o citlivé nastavenia, spustenie príkazu si vyžaduje administrátorské práva. Môžeme si všimnúť, že príkaz je čitateľnejší v porovnaní s iptables.

1.3 BPFilter

Berkeley Packet Filter [1] je nástupca nftables a bol vytvorený s úmyslom modernizácie sieťových aplikácii v Linuxe. Je založený na virtual machine podobne ako nftbales, ale je oveľa komplexnejší. Funguje na nižšej vrstve kernelu ako iptables čo značne zrýchľuje proces spracovania paketu. BPF nie je modernejší len poskytnutím svojej novej inštrukčnej sady, ale aj ponukou ďalšej infraštruktúry okolo nej. Príkladom sú mapy, ktoré fungujú ako efektívne úložiská kľúčov a hodnôt, ale aj pomocné funkcie na interakciu a využitie funkčnosti kernelu. To spôsobilo, že sa nepoužíva len na spracovanie sieťovej prevádzky, ale aj ako základ pre nové moduly a programy, ktoré majú potrebu bezpečnosti. Jeho veľkou výhodou je, že nemusí kopírovať pakety, ale posiela ďalej len tie pakety, ktoré prešli filtráciou. Na rozdiel od svojich predchodcov je lepšie integrovaný s kontajnerizačnými technológiami ako Docker a Kubernetes, čím ho sprístupňuje aj pre prostredie cloudu.

BPFilter je dostupný od Linux kernel 3.18 vydaného 7.decembra 2014.

1.4 Nftables vs Iptables

Výhodou nftables je zjednodušenie kódu. Zatiaľ čo iptables potrebujú pravidlo špeciálne pre každý protokol (IPv4, IPv6, ARP, Ethernet bridging), tak nftables to dokáže jednou všeobecnou konfiguráciou a príkazom nft. Príkladom je aj vytvorenie skupiny *inet*, ktorá združuje administráciu pre IPv4 a IPv6.

Nftables obsahuje moduly na preklad syntaxe iptables, a tak výrazne zjednodušuje prechod na túto novšiu verziu. Neobsahuje preddefinované reťaze pravidiel ani tabuľky, čo poskytuje používateľom väčšiu personalizáciu. Ďalšou výhodou je možnosť použitia viacerých akcií v rámci jedného pravidla pomocou logických operácii *and* a *or*. Umožňujú použite zreťazenia, pri ktorých hlavná myšlienka bola ukladanie hashovaných n-tíc s rýchlosťou vykonania blížiacej sa k $O(1)$. [12]

Ukážka kódu 1.3: Pravidlo využívajúce anonymné zreťazenie na filtrovanie na základe protokolu, odosielateľa a prijímateľa

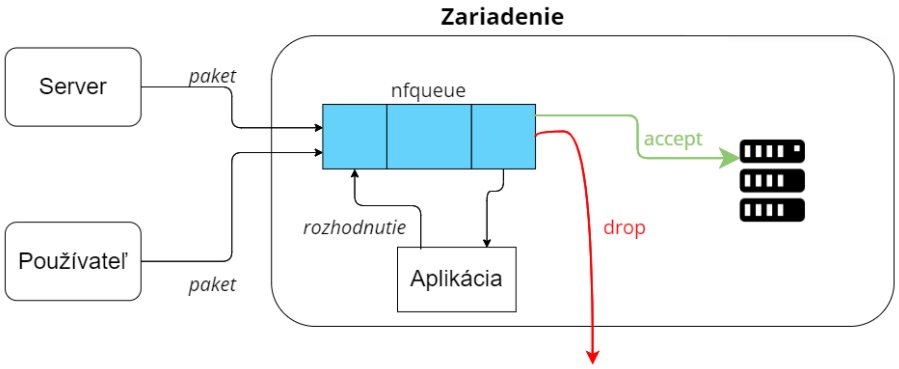
```
$ nft add rule ip filter input ip saddr . ip daddr \
. ip protocol { 1.1.1.1 . 2.2.2.2 . tcp, \
1.1.1.1 . 3.3.3.3 . udp} counter accept
```

Na ukážke 1.3 je príkaz pridávajúci pravidlo, ktoré kontroluje pakety na základe ip adresy z ktorej prichádza, na ktorú je posielaný a protokol paketu. Akceptuje pakety spĺňajúce všetky tri podmienky pre každý prvok v množine.

1.5 Nfqueue

Nfqueue [14] je súčasťou modulu iptables aj nftables a používa sa na zachytávanie paketov do fronty, aby sa následne mohli spracovať aplikáciou zo strany používateľa. Je implementovaný ako linked-list a má zadanú konkrétnu dĺžku.

Keď paket pri prechode reťazou pravidiel spĺňa pravidlo obsahujúce presmerovanie do nfqueue, tak sa prechádzanie pravidiel preruší a paket sa premiestni do fronty. Ak je už fronta plná, pakety sa rovno zahadzujú. Na ukladanie poradia priraďuje každému paketu číslo, ktoré slúži ako id pre nasledujúcu komunikáciu a posielanie informácii o danom pakete. Následne sa odošle informácia cez *libnfnetlink*, soketom zameraným na komunikáciu medzi kernelom a používateľským priestorom. Potom program čakajúci na správy z nfqueue musí rozhodnúť o tom, čo s paketom má kernel vykonať. Buď ho označí *accept* na prepustenie, alebo *drop* na zahodenie. Rozhodnutie môže byť vykonané asynchrónne a je umožnené uskutočňovať zmeny paketu. Na obrázku 1.2 je znázornený prechod paketu cez nfqueue.



Obr. 1.2: Prechod príchodzieho paketu cez nfqueue, aplikácia rozhodne, čo sa s paketom vykoná

Kapitola 2

Analýza

Pred samotným návrhom aplikácie je potrebné sa zamyslieť nad požiadavkami. Do úvahy musíme zobrať zameranie aplikácie, používateľov a taktiež prostredie, v ktorom sa bude aplikácia využívať. Vhodné je pozrieť sa aj na už existujúce riešenia podobného zamerania. Pomôže nám to lepšie identifikovať potreby a poskytne lepší náhľad na výslednú aplikáciu. Tiež sa chceme zamerať na uľahčenie práce pre administrátora, a tak si rozoberieme možnosti ako identifikovať podobnú činnosť z rôznych ip adries.

2.1 Špecifikácia

Hlavným výstupom aplikácie by malo byť monitorovanie vstupnej sieťovej prevádzky a následná štatistika. Monitorovanie je určené pre server s operačným systémom GNU-/Linux. Výstup a spravovanie aplikácie bude prebiehať cez webovú stránku. O každej ip adrese, z ktorej prichádzajú pakety, zistí vlastníka a rozsah adries, ktorý danej organizácii prislúcha. Tieto dáta sú verejne dostupné a jednou z najrozšírenejších webových stránok zameraných na tento účel je `who.is`.

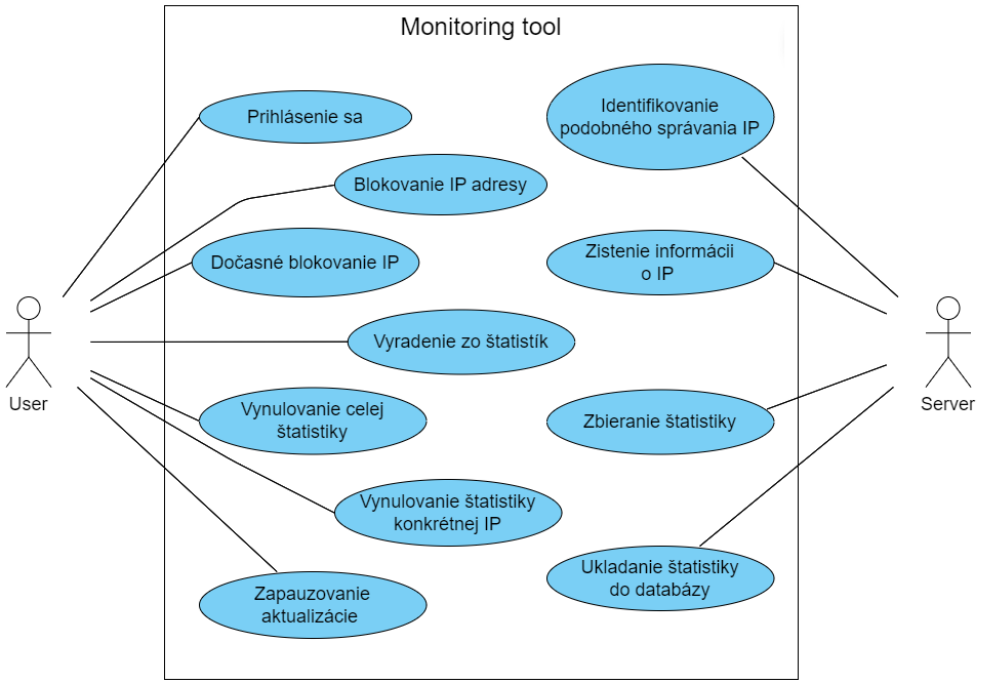
Na základe zobrazených údajov bude mať administrátor možnosť spravovania ip adresy, a teda aj paketov z nej. Použiť môže jednu z možností blokovanie konkrétnej adresy, blokovanie rozsahu adries, do ktorého patrí podľa organizácie, dočasné blokovanie a vyradenie zo štatistík. Súčasťou stránky bude aj spravovanie už vytvorených pravidiel.

V prípade podobnej činnosti z rôznych ip adries, aplikácia upozorní administrátora na toto správanie. Vhodným adeptom sú algoritmy pomenované editačná vzdialenosť. Tie porovnávajú dva a viac textových reťazcov, ako veľmi sa od seba líšia. Aplikovať to však vieme aj na akékoľvek postupnosti s použitím odchýlky.

Pri samotnej implementácii a architektúre bude potrebné zohľadňovať časovú zložitosť jednotlivých častí aplikácie, aby zobrazovala údaje v reálnom čase a hlavne nespomaľovala činnosť servera. V opačnom prípade by dochádzalo k nežiadúcim obme-

dzeniam pre používateľov.

Taktiež je potrebné sa zamerať na bezpečnosť aplikácie, vzhľadom na citlivosť monitorovaných údajov. Aplikácia by nemala umožňovať neautorizovanému používateľovi zobraziť štatistiku ani spravovať pravidlá blokovania a povolovania príchodzích paketov.



Obr. 2.1: Usecase diagram zobrazujúci úlohy, ktoré má poskytovať server a môže vykonať používateľ

Na obrázku 2.1 sú znázornené možnosti, ktoré by mala aplikácia umožniť vykonať používateľovi a čo je úlohou servera.

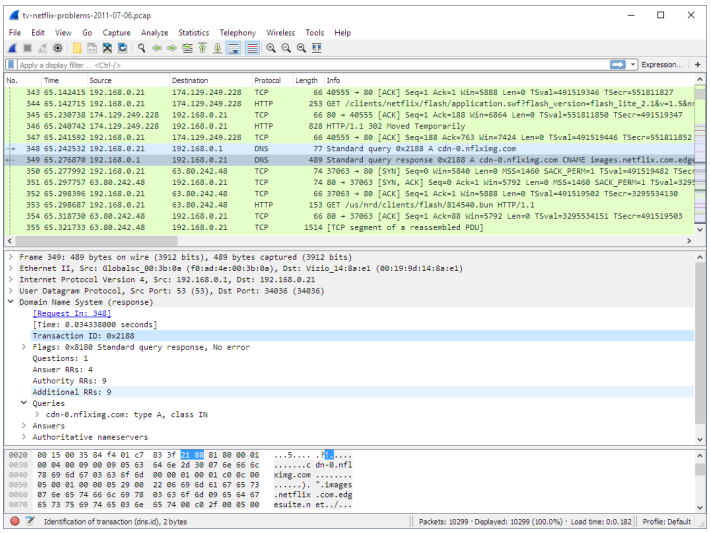
Na ukážku si uveďme príklad zo života. Knižnica na dedine sa rozhodla presunúť do internetového prostredia, aby si napríklad dôchodcovia mohli požičať knihy z pohodlia domova. Na vytvorenie knižného preukazu je potrebné sa identifikovať občianskym preukazom, a tak sú tieto dáta uložené aj na serveri. Vzhľadom na pravdepodobnejšiu neskúsenosť dôchodcov s internetovým prostredím je to lákavý cieľ pre útočníka, ktorý chce získať osobné údaje poprípade peniaze, ak je služba spoplatnená. Preto chcú majitelia knižnice sledovať činnosť na ich serveri, či nie je niečo podozrivé. Vzhľadom na cieľ servera je nepravdepodobný veľký prenos dát, ani veľká frekventovanosť komunikácie zo strany jedného klienta. Preto, keď sa na grafe zrazu zobrazí výrazne vyšší stĺpec, je to podozrivé. Majiteľ knižnice chce mať možnosť zablokovať túto činnosť. Mohlo sa však stať, že sa niekomu zasekol počítač, a tak naklikal veľa dopytov, preto je vhodné mať možnosť aj dočasne blokovať. Zároveň by mal monitorovací nástroj ponúkať zobrazenie podobnej činnosti, aby sa dalo preventívne predchádzať útokom.

2.2 Existujúce riešenia

V tejto podkapitole si opíšeme už existujúce riešenia na podobný problém, ktorými sa môžeme v našej aplikácii inšpirovať. Väčšinou sú to aplikácie a nástroje so širším zameraním na analýzu a spravovanie sieťovej komunikácie.

2.2.1 Wireshark

Wireshark [8] je populárny open-source analyzátor sieťových protokolov, ktorý umožňuje správcom siete zachytávať a analyzovať sieťovú prevádzku v reálnom čase. Používa sa na odstraňovanie problémov so sieťou, vykonávanie analýzy zabezpečenia siete a optimalizáciu výkonu siete.



Obr. 2.2: Ukážka používateľskej obrazovky pre aplikáciu Wireshark so záznamom paketov

Wireshark funguje tak, že zachytáva sieťové pakety a zobrazuje informácie v nich obsiahnuté. Dokáže zachytávať pakety na Ethernet, Wi-Fi a iných sieťových rozhraniach a dokáže analyzovať širokú škálu protokolov vrátane TCP, UDP, HTTP, DNS a SSL/TLS.

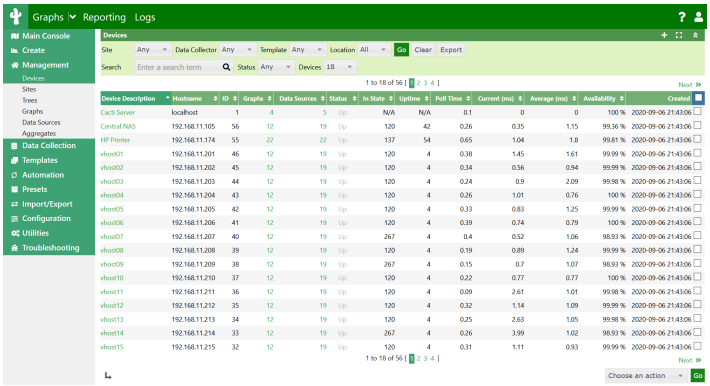
Akonáhle sú pakety zachytené, Wireshark poskytuje komplexnú sadu nástrojov na analýzu a rozdelenie zachytených dát. Dokáže zobraziť podrobnosti o jednotlivých paketoch vrátane ich zdrojových a cieľových adries, typu protokolu a samotný obsah. Na obrázku 2.2 sú tieto informácie zobrazené na obrazovke aplikácie. Môže tiež vykonávať operácie filtrovania a vyhľadávania paketov, čo umožňuje správcom nájsť konkrétne pakety na základe ich atribútov.

Wireshark tiež obsahuje pokročilé funkcie, ako sú dekodéry protokolov, ktoré dokážu analyzovať a zobraziť obsah paketov na základe používaného protokolu. To uľahčuje pochopenie obsahu paketov a identifikáciu akýchkoľvek problémov alebo anomálií.

Okrem svojich základných funkcií má Wireshark veľkú komunitu používateľov, ktorí vyvinuli širokú škálu doplnkov a rozšírení na zlepšenie jeho funkčnosti. Patria sem ďalšie dekodéry protokolov, vlastné analytické nástroje a podpora nových sieťových rozhraní a protokolov.

2.2.2 Cacti

Cacti [17] je nástroj na monitorovanie siete, ktorý poskytuje grafické rozhranie na vizualizáciu a správu sieťovej infraštruktúry. Umožňuje správcom monitorovať a zhromažďovať metriky výkonu z rôznych zariadení, ako sú smerovače, prepínače, servery a aplikácie, a na základe zhromaždených údajov vytvárať vlastné ovládacie panely a zostavy.



Obr. 2.3: Ukážka používateľskej obrazovky pre aplikáciu Cacti so záznamami ip adries

Cacti používa SNMP (Simple Network Management Protocol) na získavanie údajov zo sieťových zariadení a môže podporovať aj iné monitorovacie protokoly ako WMI (Windows Management Instrumentation) a SSH (Secure Shell).

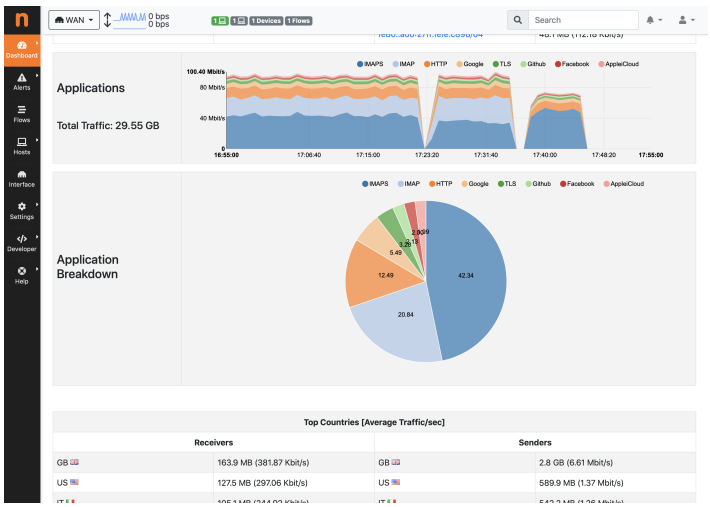
Tento nástroj ukladá zhromaždené údaje do databázy MySQL a umožňuje správcom vytvárať vlastné grafy a zostavy na vizualizáciu údajov v reálnom čase alebo neskôr spätne. Tieto grafy môžu zobrazovať údaje, ako je využitie procesora, využitie pamäte, sieťová prevádzka a ďalšie metriky výkonu.

Cacti tiež obsahuje pokročilé funkcie, ako sú prahové hodnoty a upozornenia, ktoré umožňujú správcom nastaviť prahové hodnoty pre konkrétne metriky výkonu a prijímať upozornenia, keď sú tieto limity prekročené. To pomáha správcom rýchlo identifikovať a riešiť problémy skôr, ako ovplyvnia výkon siete.

Okrem toho Cacti poskytuje webové používateľské rozhranie, ktoré umožňuje správcom konfigurovať a spravovať nástroj, ako aj pristupovať k rôznym zostavám a dashboardom. Podporuje tiež prostredia pre viacerých používateľov s riadením prístupu na základe rolí, vďaka čomu je ideálny pre väčšie organizácie.

2.2.3 ntopng

Ntopng [7] je pokročilejšia verzia ntop, nástroja na monitorovanie sieťovej prevádzky. Poskytuje analýzu sieťovej prevádzky v reálnom čase, čo umožňuje správcom siete monitorovať a odstraňovať problémy v ich sieťach a optimalizovať výkon siete.



Obr. 2.4: Ukážka používateľskej obrazovky pre aplikáciu Ntopng

Ntopng používa webové rozhranie, ktoré zobrazuje komplexné štatistiky návštevnosti, ktoré je možné triediť a filtrovať podľa rôznych parametrov, ako sú IP adresy, protokoly a aplikácie. Umožňuje tiež prispôsobiť si ovládacie panely a upozornenia, čo uľahčuje sledovanie konkrétnych oblastí záujmu. Na obrázku 2.4 je znázornená základná obrazovka po načítaní webovej stránky

Jednou z kľúčových vlastností ntopng je jeho schopnosť poskytnúť podrobné zobrazenie siete analýzou prevádzky na 2. (linkovej), 3. (sieťovej) aj 4. (aplikačnej) vrstve. Dokáže rozpoznať a klasifikovať tisíce aplikácií a protokolov vrátane šifrovanej prevádzky a identifikuje najväčších používateľov siete.

Ntopng je veľmi flexibilný a poskytuje podrobnú prispôsobiteľnosť, ktorá umožňuje vývojárom tretích strán rozšíriť funkčnosť nástroja. To dovoľuje správcom siete pridávať funkcie a funkcie, ktoré sú špecifické pre ich potreby.

2.3 Identifikácia podobného správania

Počas monitorovania sieťovej prevádzky sa vyskytne veľké množstvo rozdielnych ip adries. Ak by mal administrátor kontrolovať všetky po jednom, bolo by to jednak neefektívne a dvojak by pri tom strávil veľké množstvo času. Preto je potrebné si to uľahčiť. Ak by sme triedili adresy podľa spoločných znakov, stačilo by potom určiť, čo sa stane s celou skupinou, a tak by sa značne zmenšil počet rozhodnutí.

Na posúdenie podobností záznamov o ip adrese po jej zablokovaní využijeme editačnú vzdialenosť [6]. Metóda sa používa primárne pri textových reťazcoch, ale je možné ju použiť na porovnanie ľubovoľných postupností. Pri tejto metóde sa posudzuje koľko operácii je potrebné vykonať, aby sme z jednej postupnosti získali druhú. Postupne prechádzame postupnosť a vykonávame jednu operáciu z množiny dostupných operácii. Počet vykonaných operácii sa nazýva editačná vzdialenosť. Existuje viacero prístupov, kde každý z nich poskytuje rôzne množiny operácii na zmenu postupnosti. V tejto kapitole si rozoberieme niektoré z nich.

2.3.1 Levenshteinova vzdialenosť

Metódu vymyslel sovietsky matematik Vladimír Levenshtein v roku 1965. Keďže je najvšeobecnejšia, je známa aj pod názvom editačná vzdialenosť, hoci toto pomenovanie združuje viacero typov. Umožňuje tri operácie a to vymazanie, vloženie a zámenu znaku za iný.

Má široké využitie v bioinformatike, počítačovej vede a lingvistike. Používa sa na identifikáciu preklepov a následné poskytnutie alternatív. V bioinformatike zas na porovnanie dvoch reťazcov DNA a ich zhodu.

Časová zložitosť pre dve postupnosti dĺžok n, m je $O(n \cdot m)$. Pre porovnanie postupnosti dĺžky n s M postupnosťami dĺžok $m_1, ..., m_N$ je časová zložitosť, kde m_{max} je najdlhšia postupnosť:

$$O\left(n \cdot m_{max}\right) \tag{2.1}$$

2.3.2 Hammingova vzdialenosť

Je pomenovaná po autorovi, americkom matematikovi Richadovi Hammingovi, ktorý pracoval okrem iného aj na projekte Manhattan a získal aj Turingovu cenu v roku 1968. Predstavuje počet totožných pozícii, na ktorých sa líšia dve postupnosti. Umožňuje používať len operáciu zámeny znaku za iný, a teda postupnosti musia byť rovnakej dĺžky.

Značne sa využíva v teórii kódovania, špecificky v blokovom kóde (spôsob kódovania na prenos kanálom) alebo pri detekcii chýb v správach.

Zložitosť algoritmu pre N postupností dĺžky n:

$$O\left(n \cdot (N-1)\right) \tag{2.2}$$

2.3.3 Najdlhšia spoločná postupnosť

Z anglického *longest common subsequence* sa používa skrátený názov LCS. Predstavuje najdlhšiu postupnosť znakov spoločnú pre porovnávané postupnosti. Od najdlhšieho spoločného podreťazca sa odlišuje tým, že spoločná postupnosť sa nemusí nachádzať vcelku, teda môže byť prerušovaná. Povolené operácie sú len vloženie a vymazanie znaku.

Využíva sa ako základ porovnávania dátových setov (napríklad nástroj *diff* v Unixe), v počítačovej lingvistike a bioinformatike. Najbežnejšie je používaný v nástrojoch na detekciu zmeny vo verziách ako napríklad Git.

Časová zložitosť algoritmu na zistenie najdlhšej spoločnej postupnosti pre N postupností dĺžok $n_1, ..., n_N$ s použitím greedy algoritmu:

$$O\left(2^{n_1} \sum_{i>1} n_i\right) \tag{2.3}$$

Avšak porovnať jednu postupnosť s inou je jednoduchšie a s použitím dynamického programovania klesne zložitosť na rovnakú ako pri sekcii 2.3.1:

$$O\left(n \cdot m_{max}\right) \tag{2.4}$$

Kapitola 3

Návrh

Už vieme, čo má naša aplikácia obsahovať, preto sa v tejto kapitole zameriame na to, ako čo najlepšie dosiahnuť splnenie požiadaviek.

Ak už existuje vhodné riešenie, je výhodné ho použiť. Ušetrí to čas a môžeme sa zaoberať primárnym cieľom. Node.js poskytuje viecero správcov modulov, ktorí jednoduchým spôsobom umožňujú zapracovanie do projektu. My budeme používať správcu *npm*. Node má veľkú základňu modulov a používateľov, ktorí prispievajú novými modulmi. Využijeme niektoré z nich, ktoré budú vyhovovať našej aplikácii.

Tiež sa musíme rozhodnúť, ktorú editačnú vzdialenosť použijeme na analýzu v aplikácii. Nenašli sme vhodný modul, preto ju budeme implementovať my.

3.1 Moduly pre node.js

Keďže aplikáciu vytvárame v node.js, je vhodné použiť už vytvorené moduly, aby sme nemuseli robiť všetko od začiatku. V tejto podkapitole sa budeme venovať node modulom, ktoré poskytujú požadovanú funkcionalitu a môžeme ich použiť v implementácii.

3.1.1 nfqueue

Je modul vytvorený používateľom na github pod používateľským menom atoy20 [4]. Slúži na spojenie linuxovej netfilter aplikácie libnetfilter_queue a node.js, a tým umožňuje filtrovanie paketov pomocou javascriptu. Filtrovanie sa vykonáva asynchrónne vďaka knižnici libuv poll. Programovaný bol prioritne pre iptables, ale dá sa aplikovať aj pre nftables.

Modul poskytuje vytvorenie queue handlera, ktorý získava informácie o presmerovaných paketoch do libnetfilter_queue, čo je API na spracovanie paketov pomocou iných aplikácií, ako samotným kernelom. Paket sa ukladá to triedy NFQueuePacket, kde sú o pakete zaznamenané údaje z hľadiska nfqueue (dĺžka, koľký je v poradí, sieť, protokol a podobne) a zároveň samotný paket.

Modul umožňuje aj jednotlivé pakety označiť, ako ich má potom nasledujúci program nftables alebo iptables spracovať. Možné verdikty sú:

- NF_DROP zahodí packet
- NF_ACCEPT akceptune packet a pokračuje v iterácii
- NF_QUEUE presmeruje packet do inej queue
- NF_REPEAT ziteruje rovnaký cyklus ešte raz
- NF_STOP akceptuje packet, ale už nepokračuje v iterácii

3.1.2 socket.io

Socket.io [3] je knižnica, ktorá poskytuje obojsmernú komunikáciu medzi serverom a klientom v reálnom čase. Umožňuje výmenu dát a komunikáciu riadenú udalosťami cez jediné pripojenie TCP soketu. Používa websockets ako primárny transportný protokol, ktorý umožňuje efektívnu a plne duplexnú komunikáciu.

Cez Socket.io môže server aj klient vysielať a čakať na udalosti. Dátová časť je plne upraviteľná, čo umožňuje flexibilnú a prispôsobenú komunikáciu. Táto architektúra riadená udalosťami uľahčuje vytváranie aplikácií, ktoré vyžadujú aktualizácie v reálnom čase, ako sú chatovacie aplikácie, nástroje na spoluprácu a analytické nástroje s dátami v reálnom čase.

3.1.3 pcap

Tento modul poskytuje spracovanie paketov, ich dekódovanie alebo analýzu, z čoho je aj vytvorený názov ako skratka Packet Capture [16]. Prepája linuxovú knižnicu libpcap, ktorá sa používa v programoch ako tcpdump alebo wireshark na odchytávanie paketov. Pakety zbiera na linkovej vrstve.

Za knižnicou stojí tím programátorov z Lawrence Berkeley National Laboratory, University of California a momentálne ju spravuje The Tcpdump Group. Je napísaná v jazyku C, čo umožňuje jej široké využitie. Výhodou knižnice je podpora uloženia odchytených packetov do súbora.

V našej aplikácii využijeme len funkcionalitu dekódovania nespracovaného paketu.

3.1.4 charts.js

Slúži na zobrazovanie grafov pomocou javascriptu na webstránkach [10]. Poskytuje 8 najčastejšie používaných typov grafov. Jeho veľkou výhodou je, že grafy vykresľuje do canvas elementu na rozdiel od väčšiny knižníc na zobrazovanie grafov, ktoré používajú

SVG. Tým menej zaťažuje renderovanie stránky a je tak efektívnejší. To však prináša aj nevýhodu v podobe nepoužiteľnosti CSS.

Graf sa preto musí konfigurovať a štylizovať pomocou vstavaných operácii. Je prispôsobený na spracovávanie veľkých datasetov, s ktorými vie efektívne pracovať, čo poskytuje používateľom jednoduchší prístup. Grafy sú dopredu predkonfigurované, čo umožňuje aj novému používateľovi vytvoriť prehľadný graf, ktorý zaujme. Zároveň je vysoko prispôsobiteľný vďaka custom plugins a teda poskytuje veľkú variabilitu.

Modul má už zakomponované typovanie pre TypeScript, takže je kompatibilný so všetkými populárnymi frameworkami JavaScriptu, ako sú napríklad Vue, React, Angular alebo Svelte.

3.1.5 mariadb

Umožňuje spojenie s databázou dvoma spôsobmi [11]. Prvý spôsob je Promise API, ktorý je odporúčaný a prednastavený pri inštalácii cez node package manager. Využíva funkcionalitu node.js na vykonávanie asynchrónnych operácii, kde výstupom je objekt, čakajúci na dokončenie udalosti.

Druhým spôsobom je Callback API, v ktorom posielame pri prístupe do databázy ako parameter funkciu na spracovanie získanej odpovede. Výhodou tohoto prístupu je kompatibilita s *mysql API* a *mysql2 API*.

3.2 Editačná vzdialenosť

Pre samotné začlenenie do aplikácie sme sa museli rozhodnúť, ktorú editačnú vzdialenosť použijeme, a prečo je najvýhodnejšia.

3.2.1 Levenshteinova vzdialenosť

Implementovanie tohto algoritmu poskytuje analýzu dát za určené posledné obdobie. Tým, že je nezávislé od dĺžky postupností, môžeme analyzovať dáta všetkých správaní spätne od času vybrania ip adresy. Tri možnosti úpravy postupnosti poskytujú dostatočnú voľnosť analýzy. Keďže neporovnávame reťazce, ktoré sú tvorené obmedzeným počtom elementov, ale postupnosti čísel, zaviedli sme si odchýlku. Tá zaručuje, že ak je rozdiel čísel v stanovenom intervale, považujeme čísla za rovnaké. Odchýlka bude predstavovať percentuálny podiel čísla z postupnosti, ktorú berieme ako predlohu porovnávania.

Z dôvodu najvšeobecnejšieho využitia sme sa rozhodli tento algoritmus implementovať a použiť ho v aplikácii. Nachádza sa v súbore *editDistance.js*.

3.2.2 Hammingova vzdialenosť

V našej práci sme ju použili vďaka jej nízkej časovej zložitosti. Dokáže spracovať väčšie množstvo dát za kratší čas. Je však nepravdepodobné, že by niektoré ip adresy mali úplne totožné správanie, preto budeme používať toleranciu. Ak sa rozdiel v správaní nachádza v stanovenej odchýlke, považuje sa správanie za totožné.

Keďže algoritmus potrebuje rovnakú veľkosť porovnávaných postupností, z databázy budeme pristupovať k posledným n údajom pre každú ip adresu. To prináša nevýhodu, že porovnávané ip adresy musia mať dostatočný počet záznamov, inak nebudú vyhodnotiteľné. Ďalšou nevýhodou v porovnaní s Levenshteinovou vzdialenosťou je, že nezachytáva časový rámec aktivity ip adresy. Je to spôsobené nutnosťou rovnakej dĺžky postupností, a teda jedna z nich môže mať dáta 10 minút spätne, zatiaľ čo pre druhú ip adresu sú prvé údaje staré týždeň.

Tento algoritmus sme implementovali, ale nakoniec sme sa ho rozhodli nepoužiť v aplikácii. Ostáva však súčasťou komponentu *editDistance.js* a je možné jeho pripojenie.

3.2.3 Najdlhšia spoločná postupnosť

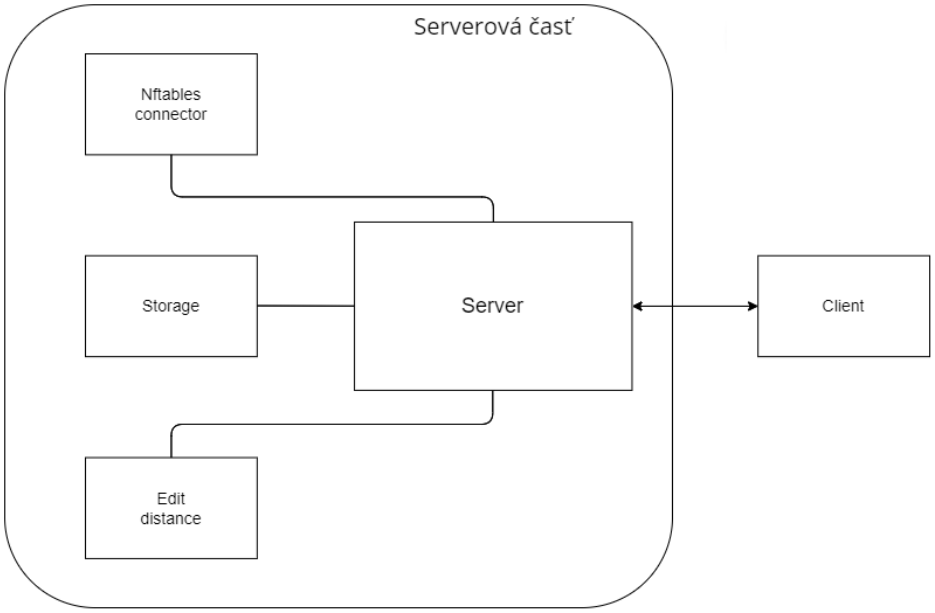
Tak ako pri predchádzajúcich algoritmoch, aj pri implementácii tohoto je nutné si zaviesť odchýlku. Poskytuje najväčšiu podobnosť voči predchádzajúcim algoritmom, vďaka prístupu k postupnosti ako celku v čase. Výhodou tohto algoritmu v našej práci je detekcia správania na základe následnosti udalostí.

Je náchylný na podobnosť neaktívnych alebo málo-aktívnych ip adries, ktoré tým vykazujú veľkú podobnosť.

3.3 Architektúra

Prvý návrh aplikácie spočíval v zakomponovaní obidvoch verzii linuxového firewallu iptables aj nftables. To by znamenalo vytvorenie správy pravidiel pre každý firewall zvlášť. Pri analýze možných riešení sme však dospeli k záveru, že postačujúce bude implementovanie iba nftables z dôvodu jednoduchšej syntaxe a poskytovaným dátovým štruktúram. Takisto väčšina aktuálnych distribúcii Linuxu obsahuje predinštalované nftables. Hoci iptables sú stále podporované, pomaly sa od nich upúšťa a administrátori sa prikláňajú k nftables.

Pre použitie aplikácie s iptables je možné to doimplemetovať a vďaka modularite napojiť na zadávanie pravidiel do firewallu.

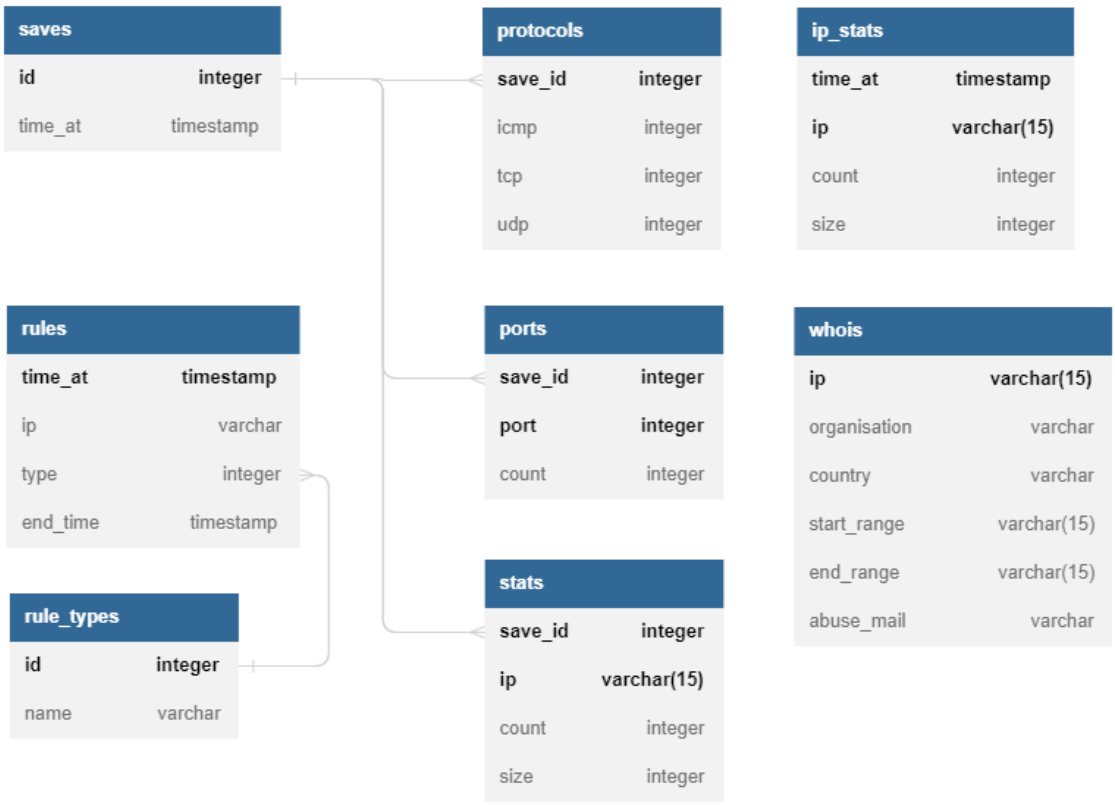


Obr. 3.1: Prepojenie komponentov v aplikácii

3.3.1 Komponenty

Naša aplikácia je rozčlenená do viacerých komponentov pre väčšiu prehľadnosť a modularitu viď obrázok 3.1:

- Server - hlavný komponent celej aplikácie. Združuje ostatné komponenty a spracováva prichádzajúce pakety, z ktorých vytvára štatistiku. Je spustený na serveri, ktorého sieťovú činnosť chceme monitorovať.
- Client - používateľské rozhranie, cez ktoré komunikuje administrátor s aplikáciou. Dostáva informácie od servera, ktorému posiela užívateľskú odozvu. Má na starosti vykresľovanie grafov a už existujúcich pravidiel.
- Nftables connector - poskytuje funkcie na pridávanie a mazanie príkazov v linuxovom module nftables. Zároveň si uchováva informáciu o už existujúcich pravidlách.
- Storage - zabezpečuje komunikáciu s databázou mariadb. Obsahuje preddefinované dopyty na databázu a formátovanie získaných údajov do použiteľného formátu.
- Edit distance - komponent na vyhodnocovanie podobností postupností.



Obr. 3.2: Entitno-relačný model databázy

3.3.2 Databáza

Do databázy si budeme ukladať údaje, ktoré potrebujeme mať zachované aj po vypnutí aplikácie. Zároveň je efektívnejšia a rýchlejšia na prehľadávanie dát.

Ukladajú sa štyri typy údajov: štatistika ip adries v čase, celková štatistika, aktuálne pravidlá a informácie o ip adrese. Pre každý z nich sme vytvorili tabuľky tak, aby čo najlepšie pokryli uloženie údajov a zároveň boli dobre spravovateľné. Entitno-relačný model databázy si môžeme pozrieť na obrázku 3.2.

- **Tabuľka so štatistikou ip adries v čase** - Z dôvodu porovnávania aktivity jednotlivých ip adries si budeme ukladať o každej ip adrese počet paketov a aj ich celkovú veľkosť. V pravidelných intervaloch sa budú pridávať záznamy s aktuálnym časom, ktoré potom budeme analyzovať. Tabuľka *ip_stats* bude mať ako primárny kľúč dvojicu čas záznamu a samotnú ip adresu. Potom dva stĺpce koľko paketov prišlo a aké množstvo dát sa prenieslo.

- **Celková štatistika** - Hlavnú časť aplikácie, a teda samotnú štatistiku, je dobré ukladať z dôvodu pokračovania po znovu spustení aplikácie. Taktiež bude mať administrátor záznam o tom, ako sa menila v čase a možnosť prípadnej analýzy. Potrebujeme uložiť dáta o štatistike portov, protokolov a celkovú štatistiku ip adries. Pre každý z údajov budeme mať jednu tabuľku, ale keďže sa budú všetky tieto údaje ukladať do databázy naraz, vytvoríme ešte jednu tabuľku *saves* pre uloženie času. Tá bude predstavovať zoznam uložení štatistických dát, a tak jednoduch zitíme čas posledného uloženia pre načítanie.

  V tabuľke pre protokoly *protocols* bude stĺpec pre každý z protokolov. Budeme zaznamenávať protokoly icmp, tcp a udp. Záznam bude mať primárny kľúč id času uloženia z tabuľky *saves*.

  Tabuľka s portami *ports* bude obsahovať záznamy pre každý port v danom čase, koľko paketov naň prišlo. Keďže vopred nevieme na aké porty sa budú posielať pakety, pre každý port bude jeden riadok v tabuľke. Preto ako primárny kľúč bude dvojica číslo portu a id času uloženia z tabuľky *saves*.

  Poslednou tabuľkou pre uloženie celkovej štatistiky je uloženie pre ip adresy *stats*. Budeme postupovať podobne ako pri portoch, a teda primárnym kľúčom bude dvojica ip adresa a id času uloženia. Pre každú ip adresu sa ešte zaznamená počet príchodzích paketov a celkový objem dát z nej.

- **Aktuálne pravidlá** - Aby sme mohli pri spustení aplikácie editovať už vytvorené pravidlá v nftables, budeme si ukladať informáciu o nich v databáze v tabuľke *rules*. Pri blokovaní ip adresy alebo inej akcie s ňou si pridáme záznam do tabuľky o adrese, kedy to bolo vykonané a o aký typ akcie sa jednalo. Keďže možných akcií je viacero a je triviálne ich pridávať, vytvoríme si druhú tabuľku *rule_types*, ktorá bude obsahovať tieto typy. Potom ju budeme využívať ako cudzí kľúč (angl. *foreign key*) v hlavnej tabuľke pre pravidlá. Typy akcií budú napríklad *blocked, tmp_blocked*, atď.

- **Informácie o ip adrese** - O každej ip adrese budeme zisťovať informácie pod akou organizáciou je registrovaná, v akej krajine, rozsah ip adries, ktorému prislúcha a email na nahlásenie neželanej činnosti. Podľa toho vytvoríme aj jednotlivé stĺpce v tabuľke *whois*. Záznamy budú uložené na konkrétnu ip adresu ako primárny kľúč.

Kapitola 4

Implementácia

Opíšeme si jednotlivé časti implementácie, problémy a ich riešenia, na ktoré sme narazili počas programovania nášho návrhu.

Keďže sme sa rozhodli pre aplikáciu zamerať len na nftables, je potrebné mať tento Linuxový subsystém nainštalovaný. Ďalším predpokladom je nainštalovaný node.js, ktorý bude našu aplikáciu spúšťať.

Aplikácia je rozdelená do viacerých komponentov, z ktorých má každý svoj samostatný súbor. Základným je *server.js*, ktorý spája všetky ostatné komponenty, vytvára webovú stránku pre klienta a sprostredkuje s ním komunikáciu. V súboroch *client.js, client.css, index.html* je definovaný vizuál a funkcionalita na strane klienta. Správa pravidiel filtrovania paketov je pre nftables v súbore *nft.js*. Na identifikáciu podobnej sieťovej aktivity rôznych ip adries sú v súbore *editDistance.js* implementované algoritmy editačnej vzdialenosti na porovnanie postupností. Poslednou súčasťou aplikácie je *db.js*, v ktorom sa nachádzajú metódy na získavanie a ukladanie dát do databázy.

4.1 Presmerovanie paketov do aplikácie

Prvým krokom implementácie bolo presmerovanie prichádzajúcich paketov z nftables do nfqueue, aby node modul nfqueue z nej vedel čítať. Dokumentácia node modulu definuje príkaz na presmerovanie paketov pre iptables. Tento príkaz sme museli preložiť do syntaxe nftables.

Ukážka kódu 4.1: Pravidlo na presmerovanie paketov do aplikácie - iptables

```
$ sudo iptables -I INPUT 1 -p icmp -j NFQUEUE --queue-num 1
```

Príkaz z ukážky 4.1 vloží pravidlo na prvé miesto reťaze INPUT, ktoré sa vzťahuje na pakety s protokolom icmp a presmeruje ich na do nfqueue do fronty číslo 1. My však potrebujeme pakety všetkých protokolov, teda tento prepínač nepoužijeme. Keďže v

nftables nemáme ešte žiadne definované pravidlá, nové pravidlo nemusíme príkazom umiestňovať, stačí ho pridať. V syntaxi nftables potrebujeme definovať takzvanú *family* filtrovaných paketov, čo predstavuje akého typu sú, či IPv4, IPv6 atď. Naša aplikácia má sledovať celú prevádzku, takže použijeme inet, ktorá zahŕňa aj tieto formy. Výsledný príkaz pridania pravidla vyzerá nasledovne v ukážke 4.2:

Ukážka kódu 4.2: Pravidlo na presmerovanie paketov do aplikácie - nftables

```
$ sudo nft add rule inet filter input queue num 1
```

4.2 Transfer dát medzi serverom a klientom

Na komunikáciu medzi serverom a klientom sme použili node module *socket.io*. Ten poskytuje komunikáciu pomocou web soketov medzi klientom aj serverom v obidvoch smeroch. Na komunikáciu sa využívajú udalosti, ktorým je možné priradiť meno. Na strane príjemcu sa tieto udalosti zachytávajú a vykoná sa príslušná časť kódu. Socket.io umožňuje aj odosielať odpoveď na tieto udalosti. Túto funkcionalitu sme využili v našej práci.

Naša aplikácia má dva smery komunikácie. Prvým je odosielanie štatistiky zo servera na stranu klienta. Druhým je komunikácia v opačnom smere a to od klienta k serveru. Týmto smerom sa posielajú informácie o vykonaných akciách používateľom.

Zo strany servera je základom posielať informácie o sieťovej prevádzke v reálnom čase. Udalosti by teda mali obsahovať dáta so štatistikou. Socket.io umožuje posielať dáta vo forme javascriptového objektu. Preto ako dátovú štruktúru pre zber štatistiky použijeme práve túto formu. Vyhneme sa tým následnej transformácii na potrebný formát, ktorá by zaberala zbytočne čas a zdroje. Pri každom príchodzom pakete sa na základe jeho ip adresy a informáciám z hlavičky pripočítajú tieto údaje už k existujúcim v objekte. Na zaznamenávanie štatistiky konkrétnej ip adresy sme ako kľúč použili samotnú jej hodnotu a odkazuje na zozbieranú štatistiku počtu, stav v nftables a objem dát. Pre porty je podobná štruktúra, ale kľúčom je číslo portu a hodnota je len počet paketov. Pri protokoloch sme využili pole, kde jednotlivé indexy predstavujú definovaný protokol pre ne. Z týchto troch objektov sme vytvorili jeden, ktorý obsahuje celú štatistiku. Na stranu klienta ho odosielame udalosťou s priradeným menom *data update* každých n sekúnd podľa nastavenia administrátora. Následne sa na strane klienta spracuje a zoradí štatistika pre konkrétne ip adresy.

Klient odosiela serveru viac typov udalostí, ale tie zväčša neobsahujú komplexnejšie dáta, len jednu hodnotu. Jednotlivé interakcie s klientom majú rôzne pridelené mená udalostí, aby sa jednoduchšie špecifikovali potreby každej jednej. Každý typ úpravy

pravidiel nftables má vlastné meno pozostávajúce z efektu daného pravidla. Napríklad *remove from stats* a *add to stats* pre vyradenie ip adresy zo štatistík a jej pridanie naspäť. Tieto udalosti využívajú možnosť odpovedi, a to konkrétne, či sa daná akcia podarila vykonať. Ďalším prípadom udalosti od klienta je dopyt na server informácie o konkrétnej ip adrese s menom *ip address*. Požaduje tým informácie z whois a názve organizácie, rozsahu, krajine a mailu na nahlásenie. To sa pošle ako odpoveď. Poslednými typmi udalosti sú *delete* a *delete_all*. Prvá je použitá na vynulovanie štatistiky pre konkrétnu ip adresu a druhá na vynulovanie celej štatistiky.

4.3 Zisťovanie informácií o ip adrese

K stránke *who.is* je aj node modul s rovnakým názvom. Keď sa vyskytne ip adresa, ktorá ešte nebola zaznamenaná, zavolá sa funkcia *lookup* z whois, na zistenie informácii o nej. Tie sa potom uložia do databázy, kde primárnym kľúčom je ip adresa a ďalšie stĺpce sú informácie o nej. Výstupom z whois.lookup je však text, z ktorého je potrebné vyparsovať potrebné informácie. Na to sme vytvorili metódu *getValueByKey* na ukážke 4.3, ktorá vyhľadá na základe kľúča hodnotu. Formát sa však môže líšiť a preto funkcia poskytuje zadanie viacero kľúčov na vyhľadanie hodnoty. Funkcia hľadá pomocou regex.

Ukážka kódu 4.3: Získavanie hodnoty podľa klúča z textu

```
1  function getValueByKey(text, ...keys) {
2    let keyReg = keys[0];
3    if (keys.length > 1) {
4      keyReg = '(' + keys.join('|') + ')';
5    }
6    const regex = new RegExp("^" + keyReg + ":(.*)$", "m");
7    const match = regex.exec(text);
8    if (match) return match[2].trim();
9    return null;
10 }
```

O každej ip adrese sa uloží informácia o názve organizácie, rozsahu ip adries (z dôvodu možnosti blokovanie celej organizácie), krajina registrácie a email na ktorý sa nahlasuje zneužitie.

4.4 Správa nftables

Spravovanie pravidiel a informácie o tom, aké ip adresy sú blokované, sme oddelili do triedy NFT do súboru *nft.js*. Ako spomíname v podkapitole 1.4, výhodou nftables je dynamické pridávanie pravidiel za použitia dátových štruktúr. To nám umožňuje vytvorenie preddefinovaných pravidiel, ktoré porovnávajú ip adresy v množine. Teda naša aplikácia nebude pridávať nové pravidlá do reťaze, ale bude pridávať a odoberať ip adresy z množín. Aplikácia má ponúkať blokovanie, dočasné blokovanie a vyradenie zo štatistík. Preto potrebujeme minimálne tri pravidlá, kvôli prehľadnosti. Nftables umožňuje pridávanie aj adries s maskou, a teda ich môžeme použiť aj pre celú organizáciu.

Vďaka preddefinovaným pravidlám a dynamickým množinám, môžeme vytvoriť súbor *rules.nft*, ktorý načíta pravidlá pre nftables pred spustením aplikácie. Definované množiny s názvami *not_process, blocked, blocked_temporary*:

Ukážka kódu 4.4: Definícia množín na spracovanie paketov pre nftables

```
1  set not_process {
2      type ipv4_addr
3      flags interval
4      elements = { 127.0.0.1 }
5  }
6
7  set blocked {
8      type ipv4_addr
9      flags interval
10     elements = { 168.63.129.16, 172.64.0.0/13}
11 }
12
13 set blocked_temporary {
14     type ipv4_addr
15     flags interval, timeout
16 }
```

V ukážke kódu 4.4 na riadkoch 2, 8, 14 môžme vidieť definíciu prvkov v množine. Tieto sú pre adresy typu IPv4. Na riadkoch 3, 9, 15 sú doplňujúce špecifikácie množiny, a to *interval* umožňujúci pridávať ip adresy s maskou. Množina *blocked_temporary* akceptuje aj adresy s časovačom, po uplynutí ktorého sa daná adresa vymaže z množiny. Na riadkoch 4 a 10 si môžeme všimnúť už inicializované alebo pridané prvky v množine.

Pravidlá využívajúce predchádzajúce sety sú definované nasledovne v reťazi input

v tabuľke pre *inet*:

Ukážka kódu 4.5: Definícia pravidiel na spracovanie paketov využívajúca množiny

```
1 chain input {
2     type filter hook input priority filter; policy accept;
3     ip saddr @blocked drop
4     ip saddr @blocked_temporary drop
5     ip saddr != @not_process queue num 1
6 }
```

Riadok 2 v ukážke kódu 4.5 je hlavička, v ktorej je ukázané poradie informácii v pravidle. Riadok 3 definuje pravidlo pre pakety spĺňajúce, že ip adresa odosielateľa sa nachádza v množine *blocked*. Tieto pakety sa zahodia. Podobne aj na riadku 4 pre množinu *blocked_temporary*. Pravidlo z riadka 5 presmerováva všetky pakety, ktoré nemajú adresu odosielateľa v množine *not_process* do fronty 1. Z tejto fronty potom číta pakety node modul *nfqueue*.

Pridávanie do množín sme realizovali pomocou systémovej funkcie z node.js *exec()*, ktorá umožňuje používateľovi z prostredia javascriptu spúšťať príkazy do terminálu. Do textovej šablóny sa doplní adresa, cieľová množina a následne sa príkaz vykoná. Keďže prístup k správe nftables potrebuje administrátorské oprávnenia, naša aplikácia tiež musí byť spustená s týmito oprávneniami.

Aby bola aplikácia rýchlejšia, stav blokovaných a vyradených adries sa uchováva v množinách v triede NFT. Keď administrátor zablokuje ip adresy na základe organizácie, je potrebné, aby užívateľské rozhranie zobrazilo, že dané ip adresy sú zablokované. Preto je potrebné skontrolovať množiny, či sa v nich nachádza ip adresa z daného rozsahu.

Nenašli sme vhodný node modul, tak sme si vytvorili vlastnú triedu *IPSet*, ktorá pri každom pridaní a odstránení rozsahu vráti aj adresy, ktoré do neho patria a sú už blokované. Adresy sú transformované na číslo, a tie sú potom jednoducho porovnávateľné. Na transformáciu ip adresy z reťazca na číslo máme definovanú funkciu na ukážke 4.6:

Ukážka kódu 4.6: Transformácia ip adresy na číslo

```
1 static ipToNum(ip) {
2     return +ip.split('.').map(d => {
3         ('000' + d).substring(d.length)
4     }).join('');
5 }
```

4.5 Ukladanie dát

Databázu využívame mariadb. K nej existuje node modul, pomocou ktorého sme implementovali spojenie s databázou. Vytvorili sme komponent *db* s triedou DB, v ktorom bude všetko definované.

Prvým krokom bolo vytvorenie komunikačného kanálu medzi aplikáciou a databázou. Prihlasovacie údaje sú definované v procesných premenných, z ktorých sa doplnia počas chodu aplikácie. Následne každá funkcia si zistí spojenie, na ňom vykoná potrebné operácie a uzavrie spojenie. K tomu sme definovali funkcie na uloženie dát zo servera. Ukladanie prebieha v pravidelných intervaloch na základe definovanej premennej v procesných premenných.

Keďže z dát z tabuľky *ip_stats* sa robí porovnanie správania pomocou editačnej vzdialenosti, tieto dáta potrebujeme dostať v správnom formáte. Editačná vzdialenosť si vyžaduje dáta spätne od súčasného času a zoradené podľa adresy. Na to sme vytvorili dopyt do databázy na ukážke 4.7, ktorý obsahuje premennú *$lastHours* umožňujúcu nastavenie odpočítania hodín.

Ukážka kódu 4.7: Dopyt do databázy pre analýzu podobnosti správania

```
SELECT * FROM ip WHERE time_at >= DATE_SUB(NOW(), \
INTERVAL '0 ${lastHours}:0:0' DAY_SECOND) \
ORDER BY ip_name ASC, time_at ASC;
```

Následne získané dáta transformujeme funkciou *transformData* do objektu, kde kľúčom je ip adresa a hodnotami sú pole s postupnosťou množstva paketov a pole s postupnosťou veľkosti prenesených dát. Takto pripravené dáta sú analyzovateľné funkciami editačnej vzdialenosti.

Celkovú štatistiku ukladáme do databázy v intervaloch, ako si definoval používateľ v procesných premenných. Využívame na to vstavanú funkciu javascriptu *setInterval*. Z objektu, v ktorom je štatistika uložená si rozoberieme jednotlivé časti pre tabuľky a zavoláme funkcie, ktoré to pošlú do databázy. Kľúčové je zaznamenať čas a dátum uloženia, ktorý sa ukladá do zvlášť tabuľky *saves*. Následne sa jeho id použije ako primárny kľúč pri zaznamenávaní štatistiky ip adries do *stats*, portov do *ports* a protokolov do *protocols*. Pri novom spustení aplikácie sa tieto dáta načítajú znova do objektu v node, aby boli prístupné k zaznamenávaniu. Najskôr sa vyberie najneskorší dátum záznamu a na základe jeho id sa potom do objektov načítajú aj dáta z ostatných tabuliek so štatistikou. Keď používateľ zadá vymazať celú štatistiku, do tabuľku *saves* sa pridá nový záznam s aktuálnym dátumom a časom. Do iných tabuliek sa nič nepridáva. A preto keď sa bude aplikácia znova spúšťať, nenájde žiaden záznam, a tým je štatistika vynulovaná.

Pre tabuľku s aktuálnymi pravidlami *rules* je iné správanie pridávania. Vždy keď je vykonaná udalosť meniaca pravidlá v nftables, pridá sa nový riadok s časom, ip adrese a typom udalosti, ktorá sa vykonala. V aplikácii je definované, ktoré id patrí ktorému typu udalosti z tabuľky *rule_types*, takže nie je potrebné to zisťovať pri každom zázname. Tá tabuľka slúži hlavne na prehľadávanie histórie pravidiel cez samotnú databázu. Keď sa k serveru pripojí nový klient je potrebné mu poslať informácie o aktuálnych pravidlách. Preto sa vyberú z databázy a udalosťou s menom *current rules* vo forme objektu sa pošlú klientovi.

4.6 Autentifikácia a zabezpečenie

Naša aplikácia má prístup k dôverným a citlivým informáciám, preto je dôležité myslieť na bezpečnosť. Problémom je, že prístup ku klientovi je verejný. To znamená, že ktokoľvek sa vie dostať ku všetkým informáciám a môže vykonávať zmeny. Vhodná je autentifikácia používateľa pomocou mena a hesla. Pred prihlásením sa by klient nemal mať prístup k ničomu a po prihlásení sa je mu umožnené vidieť štatistiku a vykonávať zmeny. Pre každý smer komunikácie sme to vyriešili inak.

Smerom od servera ku klientovi, teda posielanie štatistiky, posielame udalosti len tým klientom, ktorý prešli autentifikáciou. Po prihlásení, sa aktuálny soket pridá do poľa autentifikovaných. Potom pri pravidelnej aktualizácii štatistiky sa udalosť posiela len cez sokety v danom poli. Prihlasovacie údaje je možné zmeniť v súbore *.env*, odkiaľ si aplikácia načítava premenné. Spojenie cez socket.io medzi klientom a serverom je vykonávané cez http, takže nie je šifrované. Je možnosť využiť aj https, ku ktorému stačí dodať certifikát a kľúč. Pre nešifrovanú komunikáciu je potrebné šifrovať aspoň heslo. Na to využijeme node modul *md5-js-tools*, ktorý poskytuje šifrovanie MD5. Následne zašifrujeme heslo aj na strane servera a ak sa zhodujú, klient sa úspešne autentifikoval.

V opačnom smere sme potrebovali zabezpečiť, aby server vykonával akcie len od prihlásených používateľov. Tu sa naskytlo jednoduché riešenie spôsobené implementáciou socket.io. Stačí, že sme ešte pre konkrétny soket nedefinovali pravidlá na odchytenie udalostí, a teda aj keď ich daný klient bude posielať, nebudú spracované.

Odhlásenie nemusíme poskytovať vzhľadom na to, že pri znova načítaní stránky alebo jej zatvorení, soket sa preruší a je vyžadované prihlásenie.

4.7 Aktivácia nfqueue pri spustení

Počas testovania aplikácie sme narazili na problém so súčasnou implementáciou. Pri zapnutí Linux servera sa načítajú nftables pravidlá z ukážky 4.5 a riadok 5 presmeruje pakety do nfqueue. Pokiaľ ale nebola spustená aplikácia, neboli vykonávané verdikty o

jednotlivých paketoch, a tak čakali vo fronte. Keď sa tá zaplnila, príchodzie pakety sa zahadzovali a nedochádzalo ku komunikácii. To malo za následok, že nebola dostupná žiadna činnosť z internetu a nebolo možné ani pristupovať k serveru zvonku.

Preto sme sa rozhodli pozmeniť pravidlá v nftables tak, aby sa pravidlo na presmerovanie do nfqueue pridalo, keď sa aplikácia spustí a vymazalo, keď sa aplikácia korektne vypne. Správa pravidiel v nftables neumožňuje pravidlo vymazať [15] jeho zadaním s prepínačom delete, hoci táto funkcionalita je plánovaná. Alternatívou k tomu je vymazanie pravidla pomocou čísla *handle*, čo predstavuje akési id daného pravidla. Nevýhoda spočíva v potenciálnom pridaní nových pravidiel do nftables. V takom prípade sa *handle* pravidla môže meniť. Ďalšou možnosťou je vymazanie všetkých pravidiel z reťaze. Preto sme sa rozhodli pravidlo presmerovania delegovať do inej reťaze, v ktorej bude jediným pravidlom. Tak ho môžeme jednoducho vymazať bez ovplyvnenia iných pravidiel a prístupu k sieti. Namiesto pôvodného presmerovania dáme pravidlo na skok na novú reťaz ako môžeme vidieť na ukážke kódu 4.8. Tá ak neobsahuje pravidlo, bude sa správať akoby tam ani nebola a porovnávanie paketu s ostatnými pravidlami bude pokračovať. Ďalšou výhodou takejto implementácie presmerovania je, že pravidlá o blokovaní ip adries ostávajú aktívne aj po vypnutí aplikácie.

Ukážka kódu 4.8: Presmerovanie paketov do aplikácie v novej reťazi

```
1       ip saddr != @not_process jump monitoring
2  }
3  chain monitoring {
4       queue num 1
5  }
```

4.8 Výsledná aplikácia

V tejto podkapitole si ukážeme našu výslednú aplikáciu, ako ju spustiť a výsledky niekoľkých testov. Zdrojové súbory sú dostupné v repozitári na Github [5], keďže sa jedná o open source projekt.

4.8.1 Postup inštalácie

Základným predpokladom je mať distribúciu GNU/Linux s nainštalovaným firewallom nftables. Ďalej je potrebné mať nainštalovaný node.js a databázu. Našu aplikáciu sme vyvíjali na Debian 11 bullseye, verzii node 12.22.12 a databázou 10.5.18-MariaDB.

Ako prvé nakonfigurujeme nftables, aby používalo naše pravidlá a po vypnutí servera sa znova načítali. Pri štarte sa načítava konfiguráciu zo súbora */etc/nftables.conf*,

preto do neho prekopírujeme obsah súbora *rules.nft*. Na okamžité aplikovanie pravidiel zadáme príkaz *sudo systemctl restart nftables* na reštartovanie nftables.

Ďalším krokom je vytvorenie databázy a tabuliek v nej. Za týmto účelom je vytvorený súbor *create.sql*, ktorý obsahuje potrebné príkazy a vytvorenie nového používateľa databázy na bezpečný prístup. Pred spustením je nutné prepísať meno a heslo nového používateľa a tie potom doplniť do súboru *.env*, aby sa vedela naša aplikácia prihlásiť. Prihlásime sa do mariadb a príkazom v ukážke 4.9 spustíme pripravený script. Je potrebné však nahradiť názov súboru celým umiestnením súbora v konkrétnom počítači/serveri.

Ukážka kódu 4.9: Príklad spustenia skriptu na vytvorenie databázy

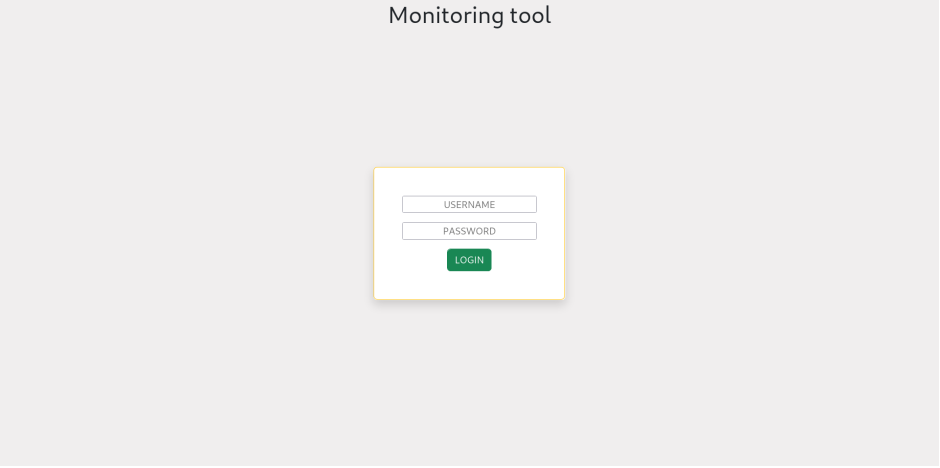
```
MariaDB [(none)]> source full/path/to/sql/create.sql;
```

Keď máme vytvorenú databázu, nasleduje aktivácia node.js. Pred samotným spustením príkazom v terminály 4.10 nainštalujeme chýbajúce knižnice pre node modul nfqueue.

Ukážka kódu 4.10: Inštalácia c knižníc v Debian

```
$ sudo apt-get install libnetfilter-queue-dev libpcap-dev
```

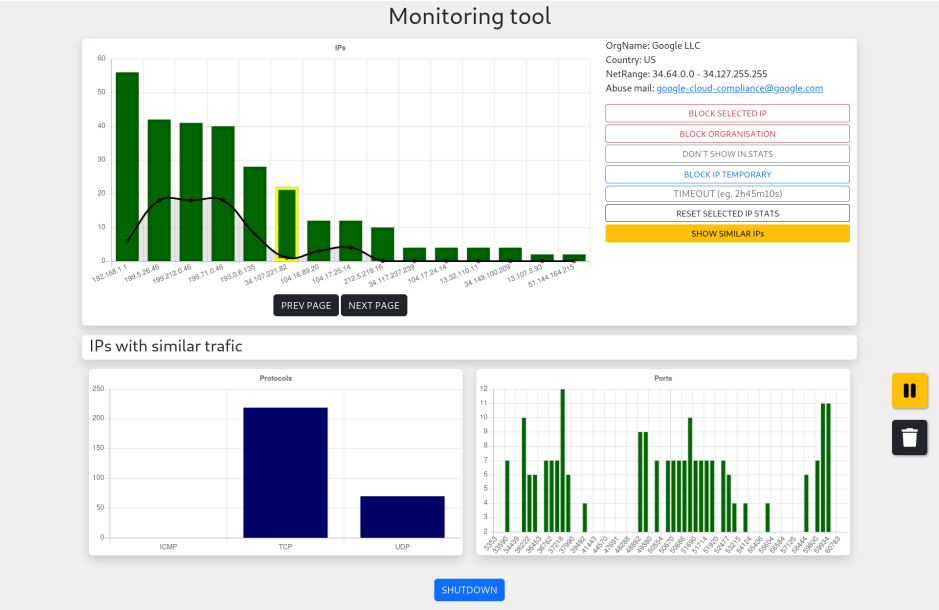
V súbore *.etc* sú definované premenné na nastavenie mena a hesla používateľa, frekvenciu ukladania do databázy a tiež na konfiguráciu editačnej vzdialenosti. Následne príkazom *npm i* nainštalujeme potrebné moduly a príkazom *sudo npm start* spustíme našu aplikáciu. Vo webovom prehliadači po zadaní domény nášho servera a portu 9000 si zobrazíme používateľské rozhranie, ktoré môžeme vidieť na obrázku 4.1.



Obr. 4.1: Výsledná aplikácia - prihlasovanie sa do webového rozhrania

4.8.2 Predstavenie aplikácie

Na obrázku 4.1 môžeme vidieť úvodnú stránku po načítaní url adresy klienta aplikácie. Je na nej prihlasovací formulár, v ktorom sa zadáva meno a heslo. Po kliknutí na tlačidlo Login sa odošle informácia serveru a ak sú prihlasovacie údaje správne, zaradí nášho klienta do zoznamu autentifikovaných. Náš klient začne dostávať štatistiku. S príchodom prvých dát formulár zmizne a na stránke sa zobrazia grafy so štatistikou ako môžeme vidieť na obrázku 4.2.



Obr. 4.2: Výsledná aplikácia - zobrazenie grafov

Hlavným prvkom stránky je prvé okno s grafom zobrazujúcim štatistiku jednotlivých ip adries. Stĺpce predstavujú počet prijatých paketov z danej ip a čierna čiara predstavuje objem prenesených dát. Sú zoradené na základe počtu paketov. V grafe kvôli prehľadnosti sa zobrazuje maximálne 15 stĺpcov na jednotlivých stránkach. Medzi nimi sa dá presúvať pomocou tlačidiel pod grafom. Stĺpce môžu mať jednu zo 4 farieb podľa pravidla, aké je priradené prislúchajúcej ip adrese.

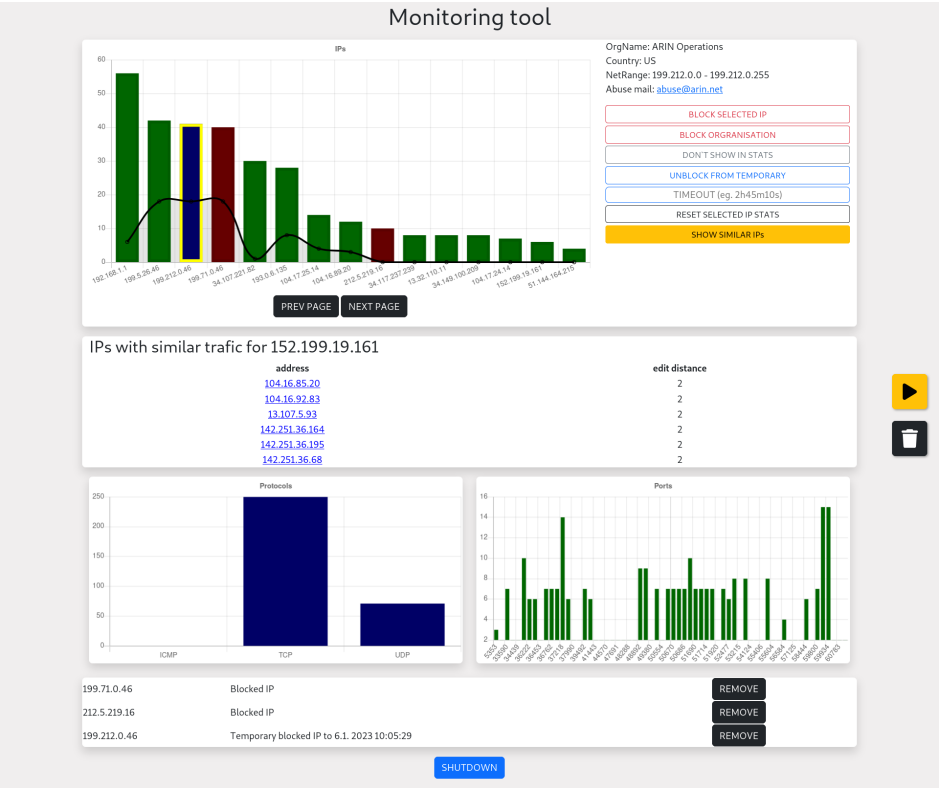
- zelená - pakety sa akceptujú
- modrá - ip adresa je dočasne zablokovaná, pakety sa zahadzujú
- červená - ip adresa je blokovaná a pakety sú zahodené
- sivá - pakety z danej ip adresy sa už nepočítajú do štatistiky

Po kliknutí na jeden zo stĺpcov sa jeho okraj zafarbí na žlto a naľavo od grafu sa zobrazia údaje o danej ip adrese. Sú to informácie z who.is o organizácii, do ktorej ip adresa patrí, krajina, rozsah a mail, na ktorý sa dá obrátiť v prípade nevhodnej činnosti.

Pod týmito informáciami sa nachádzajú tlačidlá na správu vybranej adresy. Farebné okraje tlačidiel a podfarbenie pri prechode myšou zodpovedá zafarbeniu stĺpca po aplikovaní danej akcie. Prvé dve tlačidlá sú na zablokovanie vybranej adresy a rozsahu, do ktorého patrí. V takom prípade sa na červeno zafarbia aj ostatné stĺpce prislúchajúce rozsahu. Nasleduje sivé tlačidlo neukazovať v štatistike. Pod ním je modré tlačidlo slúžiace na dočasné blokovanie. Dĺžka blokovania sa určuje textovým poľom pod tlačidlom. Zadáva sa vo formáte 8h27m14s, kde by tento čas predstavoval 8 hodín 27 minút a 14 sekúnd. Ak je vybraná ip adresa už s priradeným pravidlom, zmení sa popis súvisiacemu tlačidlu naopak. Napríklad BLOCK SELECTED IP na UNBLOCK SELECTED IP. Ďalej je čierne tlačidlo, ktoré vynuluje aktuálne počítadlá pre konkrétnu adresu. Posledné žlté tlačidlo porovná správanie vybranej ip adresy s ostatnými za posledné obdobie a výsledok zobrazí v nasledujúcom bloku.

Ďalším prvkom stránky je okno, kde sa zobrazujú ip adresy s podobnou činnosťou ako posledná adresa, pri ktorej bolo vykonané porovnanie. Po kliknutí na zobrazenú ip sa zvýrazní na žlto a vyberie prislúchajúci stĺpec. Pri každej adrese sa zobrazuje aj editačná vzdialenosť, ako veľmi sa líši s porovnávanou.

Nasledujú dva grafy, naľavo pre protokoly paketov a napravo pre porty. Tieto štatistiky sú spoločné pre všetky príchodzie pakety a vyjadrujú počet paketov spadajúcich do kategórie. Túto skutočnosť môžeme vidieť na obrázku 4.3.



Obr. 4.3: Výsledná aplikácia - boli aplikované pravidlá a zobrazené podobné ip adresy

Na spodnej časti stránky sa nachádzajú už vytvorené pravidlá. Môžu byť pre konkrétne adresy, ale aj rozsahy. Používateľ má možnosť tieto pravidlá zrušiť pomocou tlačidla REMOVE nachádzajúcim sa v každom riadku. Táto akcia je obdobou kliknutia na príslušné tlačidlo odblokovania v prvom bloku pod informáciami o ip adrese.

Na pravom okraji sú dve tlačidlá zafixované na mieste. Žlté tlačidlo funguje na pauzu aktualizácie štatistiky pre konkrétneho klienta. Po kliknutí sa zmení ikona na znovu spustenie aktualizácie. Čierne tlačidlo so smetným košom je na vymazanie celej štatistiky. Po kliknutí sa otvorí vyskakovancie okno, či naozaj to chceme vykonať a na základe potvrdenia sa štatistika reštartuje odznova, alebo nie.

Vypnutie aplikácie sa vykoná stlačením tlačidla s popisom SHUTDOWN na úplnom spodku stránky. Následne sa ukáže potvrdzovacie okno. V prípade potvrdenia sa aktuálna štatistika uloží do databázy a aplikácia sa ukončí.

4.8.3 Testovanie

Prvým krokom testovania bolo, či štatistika je posielaná len autentifikovaným používateľom a či je možné vykonať zmeny pravidiel aj bez prihlásenia. Vzhľadom na našu implementáciu webstránky, to bolo jednoducho realizovateľné. Keďže sa jedná jednostránkovú webovú aplikáciu, sú na nej prítomné všetky elementy od začiatku, len niektoré sú skryté pomocou atribútov CSS. Porovnávali sme dvoch pripojených klientov. Prvý klient sa prihlásil menom a heslom a zobrazovala sa mu štatistika. Druhý klient sa neprihlásil. Cez developerské nastavenia sme zmenili triedy elementu tak, aby bol viditeľný. Následne sme stlačili tlačidlo vymazať všetku štatistiku a nič sa v aplikácii nezmenilo. Teda aplikácia prijíma len od autentifikovaných používateľov.

Keďže pri lokálnom spustení je klient aj server na jednom zariadení, môžeme otestovať, či naša aplikácia nespomaľuje bežnú činnosť. V takomto prípade, každý podnet zvnútra von si vyžaduje spätnú komunikáciu, a tá je vo forme paketov.

Vyskúšali sme stiahnuť súbor z webovej stránky. Konkrétne sa jednalo o balíček s aktualizáciou o veľkosti 91 Mb. Pokiaľ nebola spustená aplikácia, doba stiahnutia sa pohybovala v okolí 35s čo predstavuje približne 2.6 Mb/s. Pri spustenej aplikácii stiahnuť rovnaký balíček trvalo približne 45s, teda s priemernou rýchlosťou 2 Mb/s. Tento výsledok mierne zaostal za očakávaním, ale zároveň toto spomalenie nie je fatálne a stále sa dá s ním pracovať.

Ďalej sme skúšali spustiť video na Youtube. Vzhľadom na možnosti virtuálneho počítača, na ktorom sme vytvárali aplikáciu, sa video zasekávalo aj keď nebola spustená aplikácia. Po jej spustení sme nezaznamenali väčší rozdiel v zhoršení kvality. Rozdiel sme si všimli v zvukovej stope, ktorá so spustenou aplikáciou sekala v pravidelnejších intervaloch.

Vzhľadom na efektivitu nftables bol ďalší test zbytočný. Spočíval v pridávaní väčšieho množstva pravidiel, avšak nezaznamenali sme žiaden rozdiel v správaní sa servera aj aplikácie. Pridali sme približne 100 ip adries na zablokovanie.

Poslednou časťou, ktorú sme testovali, bolo identifikovanie podobnej činnosti. Postupne sme prechádzali jednotlivé záznamy o ip adrese a používali tlačidlo na zobrazenie podobných ip adries. Identifikovali sme analogickú činnosť ip adries od spoločnosti Google a viacero iných organizácii, ako napríklad Microsoft. Väčšinou podobné adresy patrili pod rovnakú organizáciu, čo považujeme za úspešné.

Záver

Cieľom práce bolo monitorovanie sieťovej prevádzky. Vytvorili sme aplikáciu pomocou nástroja Node.js a jeho vhodných modulov. Inšpirovali sme sa už existujúcimi aplikáciami a navrhli sme vlastné riešenie. Zamerali sme sa na vyhodnotenie podobnej sieťovej činnosti jednotlivých ip adries pre zjednodušenie práce administrátora. Výhodou našej aplikácie je jej jednoduchá úprava a možné pridávanie grafov, poprípade modulov.

Počas implementácie sme narazili na viacero problémov, ale podarilo sa nám ich vyriešiť a naprogramovať funkčnú aplikáciu, ktorá zbiera dáta o príchodzej sieťovej prevádzke. Tie následne analyzuje a v reálnom čase sa zobrazujú vo webovom prehliadači vo forme grafov. Administrátor si môže zobraziť údaje o konkrétnej ip adrese a v prípade potreby je umožnené ju zablokovať. Aplikáciu sme následne otestovali na virtuálnom stroji a porovnali, či veľmi nezaťažuje server. Hoci niektoré ukazovatele naznačujú spomalenie, je to spôsobené primárne malým výkonom hardvéru, na ktorom bola aplikácia testovaná.

Stále sa však ponúka veľa príležitostí a možností kam by sa mohla naša aplikácia rozvíjať. Jednou z nich je použitie neurónovej siete na analýzu činnosti z ip adries. Po dôkladnom natrénovaní má potenciál pokrývať oveľa viac druhov správania sa ako editačná vzdialenosť, ktorá sa zameriava na lineárne postupnosti. Mohla by sledovať závislosť množstva paketov od ich veľkosti alebo analyzovať aj na základe času prijatia paketu. Iným vylepšením je implementovanie nových možností regulácie činnosti. Napríklad určením limitu množstva dát za čas alebo počet paketov. Z hľadiska monitorovania sa ponúka zobrazenie činnosti viacerých ip adries na grafe v čase. Taktiež sa dá rozvinúť štatistika týkajúca sa portov, aby boli konkretizované na adresy. Priamym vylepšením našej aplikácie je podpora správy paketov IPv6.

Literatúra

[1] Cilium Authors. Bpf architecture. [Citované 2023-05-15] Dostupné z https://docs.cilium.io/en/latest/bpf/architecture/.

[2] Herve Eychenne. iptables(8) - linux man page. [Citované 2023-05-18] Dostupné z https://linux.die.net/man/8/iptables.

[3] Damien Arrachequesne Guillermo Rauch. Introduction, 2023. [Citované 2023-04-29] Dostupné z https://socket.io/docs/v4.

[4] Anthony Hinsinger. node-nfqueue, 2022. [Citované 2023-04-29] Dostupné z https://github.com/atoy40/node-nfqueue.

[5] Murin Jakub. Monitoring-tool, 2023. [Citované 2023-05-31] Dostupné z https://github.com/JakubMurin/Monitoring-tool.

[6] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[7] The ntop Team. ntopng. [Citované 2023-04-10] Dostupné z https://www.ntop.org/products/traffic-analysis/ntop.

[8] The Editors of Encyclopaedia Britannica. Wireshark · about. [Citované 2023-04-10] Dostupné z https://www.wireshark.org/about.html.

[9] The Editors of Encyclopaedia Britannica. Firewall | definition, types, & facts | britannica, 2022. [Citované 2023-05-17] Dostupné z https://www.britannica.com/technology/firewall.

[10] Chart.js Team. Chart.js, 2023. [Citované 2023-04-29] Dostupné z https://www.chartjs.org/docs/latest.

[11] MariaDB Dev Team. About mariadb connector/node.js, 2023. [Citované 2023-04-29] Dostupné z https://mariadb.com/kb/en/about-mariadb-connector-nodejs.

[12] Netfilter Core Team. Main differences with iptables, 2021. [Citované 2023-05-10] Dostupné z https://wiki.nftables.org/wiki-nftables/index.php/Main_differences_with_iptables.

[13] Netfilter Core Team. What is nftables? - nftables wiki, 2021. [Citované 2023-05-05] Dostupné z https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F.

[14] Netfilter Core Team. libnetfilter_queue: Main page, 2022. [Citované 2023-04-15] Dostupné z https://netfilter.org/projects/libnetfilter_queue/doxygen/html.

[15] Netfilter Core Team. Simple rule management, 2022. [Citované 2023-05-20] Dostupné z https://wiki.nftables.org/wiki-nftables/index.php/Simple_rule_management.

[16] Ujjwal Thaakar. node_pcap, 2020. [Citované 2023-04-29] Dostupné z https://github.com/node-pcap/node_pcap.

[17] Inc. The Cacti Group. What is cacti?, 2021. [Citované 2023-04-10] Dostupné z https://www.cacti.net/info/cacti.

Príloha A: Štruktúra zdrojových súborov

Zdrojové súbory aplikácie a popis čo obsahujú:

- **README.md** - postup inštalácie
- **www** - priečinok so súbormi ku klientovi
  - **chartsConfig.js** - konfigurácia grafov
  - **client.css** - kaskádové štýly pre webstránku
  - **client.js** - funkcionalita webstránky klienta
  - **index.html** - definovanie domovskej webstránky
- **create.sql** - SQL skript na vytvorenie a konfiguráciu databázy
- **db.js** - funkcionalita na komunikáciu s databázou
- **editDistance** - funkcie na porovnávanie postupností na základe algoritmov editačnej vzdialenosti
- **nft.js** - zadávanie pravidiel do nftables a správa už vytvorených pravidiel
- **package-lock.json** - použité Node.js moduly
- **package.json** - definícia Node.js aplikácie
- **rules.nft** - definícia reťazí a množín nftables na správu sieťovej prevádzky
- **server.js** - hlavný súbor, v ktorom je implementovaná celková funkcionalita aplikácie