Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Зіньковського Данила Олександровича*
(ПІБ)

академічної групи *121-17-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка програмного забезпечення для фільтрації та трасування трафіку з використанням технології BPF в Linux*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Сироткіна О.І.*			
розділів:				
спеціальний	*доц. Сироткіна О.І.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента 121-17-1 Зіньковського Данила Олександровича
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення для фільтрації та трасування трафіку з використанням технології BPF в Linux*

затверджена наказом ректора НТУ «ДП» від *07.06.2021* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Сироткіна О.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Зіньковський Д.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р*

РЕФЕРАТ

Пояснювальна записка: 86 с., 24 рис., 3 дод., 20 джерел.

Об'єкт розробки: програмне забезпечення для фільтрації та трасування трафіку з використанням технології BPF в Linux.

Мета кваліфікаційної роботи: розробити програмне забезпечення для фільтрації та трасування мережевого трафіку.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної галузі та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обґрунтовується вибір операційної системи, бібліотек та фреймворків, мов програмування, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці програмного забезпечення, за допомогою якого можна швидко виконати налаштування правил фільтрації та моніторингу мережевого трафіку через зручний користувацький веб-інтерфейс.

Актуальність програмного продукту визначається критичністю несанкціонованого доступу до ресурсів в локальній мережі для малого та середнього бізнесу і великою вартістю аналогічних засобів.

Список ключових слів: BPF, LINUX, UBUNTU, МЕРЕЖЕВИЙ ТРАФІК, МОНІТОРИНГ МЕРЕЖЕВОГО ТРАФІКУ, ФІЛЬТРАЦІЯ МЕРЕЖЕВОГО ТРАФІКУ, LINUX-УТИЛІТА, ВЕБ-ДОДАТОК.

ABSTRACT

Explanatory note: 86 p., 24 figs., 3 apps., 20 sources.

Object of development: software for filtering and tracing network traffic using BPF technology in Linux.

Purpose of the qualification work: development of a software system for filtering and tracing network traffic.

In introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiates the relevance of the topic.

In the first section, a study of the subject area and existing solutions, determines the relevance of the task and the purpose of development, develops the task.

The second section substantiates the choice of operating system, libraries and frameworks, programming languages, program design and development, describes the algorithm and structure of the system, determines the input and output data, provides characteristics of the parameters of hardware, describes the program.

In the economic section, the complexity of the developed software product is determined, the cost of work on creating the application is calculated and the time for its creation is calculated.

Of practical importance is the development of software with which you can quickly configure the rules of filtering and monitoring network traffic through a user-friendly web interface.

The relevance of the software product is determined by the criticality of unauthorized access to resources in the local network for small and medium-sized businesses and the high cost of similar tools.

Keyword list: BPF, LINUX, UBUNTU, NETWORK TRAFFIC, NETWORK TRAFFIC MONITORING, NETWORK TRAFFIC FILTRATION, LINUX UTILITY, WEB APPLICATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ООП – об’єктно-орієнтоване програмування;

ОС – операційна система;

СКБД – система керування базою даних;

BPF – Berkley Packet Filter;

BCC – BPF Compiler Collection.

ВСТУП

Кількість користувачів міжнародною комп'ютерною мережею з часів своєї появи росте непомірно швидко. Сьогодні кількість пристроїв, підключених до мережі Інтернет, приблизно дорівнює 25 мільярдів пристроїв. А з розвитком концепції Internet Of Things, також зростає необхідність в моніторингових інструментах і системах захисту комп'ютерних мереж.

Одними з найбільших проблем сучасних корпоративних і домашніх комп'ютерних мереж є небажаний трафік, складність у налаштуванні захисту та користуванні звичайними власниками підмереж. Так, в абсолютній більшості домашніх мереж адміністратори не мають можливості запобігти небажаному трафіку. Неможливість зумовлена складнощами у використанні сучасних мережевих пристроїв, з часто застарілим і неефективним програмним забезпеченням та високою вартістю цих пристроїв і профілактичних заходів. З аналогічних причин часто виникають проблеми у звичайних користувачів з налагодженням списків заборонених ресурсів для відвідування.

Існуючі сьогодні розробки, що створені для вирішення вищезазначених проблем, часто являють собою великі програмні та програмно-апаратні системи, що вбачають своєю головною задачею надання можливості гнучкого та тонкого налаштування мережі, в збиток простоті використання та невеликим системним вимогам. Також, часто подібні системи є результатом довгих років розробки, через що не є можливим перехід на новіші та більш ефективні фундаментальні технології, так як виникає необхідність у підтримці зворотної сумісності з минулими версіями.

Метою даної кваліфікаційної роботи є розробка системи моніторингу та фільтрації мережевого трафіку, що є простою та зручною у використанні, а також заснованою на найсучасніших технологіях.

Узявши до уваги вищезазначене, вбачається вирішення даних проблем у розробці системи, що дозволяє:

- проводити моніторинг вхідного та вихідного трафіку;
- проводити налаштування фільтрації трафіку в мережі, максимально спростити налаштування для адміністраторів;
- взаємодіяти з системою через простий веб-інтерфейс.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Безпека корпоративних даних, приватних даних в домашній мережі є найважливішим аспектом користування міжнародною комп'ютерною мережею.

Організація малого та середнього бізнесу супроводжується великим обігом документів та інших даних. Забезпечення приватності корпоративних даних є критичною задачею для бізнесу. Домашні й корпоративні мережі часто являються недостатньо захищеними та не мають ніяких моніторингових систем для аналізу вхідного та вихідного трафіку. З спеціалізованого апаратного чи програмного забезпечення невеликі компанії обмежуються базовими мережевими пристроями. Головними причинами даних проблем є висока вартість відповідного програмного або апаратного забезпечення та необхідність в послугах висококваліфікованих спеціалістів.

Безумовним лідером серед операційних систем для серверів є операційні системи на основі ядра Linux. Двадцять років тому з'явилася Linux утиліта iptables, яка стала стандартом у налаштуванні фільтрації трафіку та перенаправлення пакетів в міжмережевих екранах, та замінила застарілу ipchains.

Утиліта iptables базувалась на послідовному ланцюгу правил, за якими відбувалося управління трафіком. Кожний пакет перевірявся на відповідність певним критеріям, і, у разі відповідності, над пакетом виконувалася відповідна дія. Під дією розуміється виконання відкидання, пропускання чи маркування пакету, або перехід на інший користувацький ланцюг правил.

Послідовна перевірка правил недостатньо ефективна, особливо при великій кількості правил. Тому у якості тимчасової більш ефективної альтернативи була представлена утиліта ipset. На той час найбільш вузьким місцем iptables був послідовний обхід великої кількості правил, тому головна зміна в ipset - це

введення можливості спресовувати список правил для кожної комбінації IP-адреси та порту у хеш-таблицю.

Зі зростанням популярності контейнерізаційних рішень (Kubernetes), проблема недостатньої ефективності вищезгаданих утиліт повернулася. На кожний сервіс, який обслуговується Kubernetes необхідно встановлювати свій набір правил iptables. Таким чином, навантаження почали зростати в геометричній прогресії. Ситуацію погіршувала нестача інкрементальних оновлень ланцюгів правил. При додаванні нового правила було необхідно перевстановити весь набір правил, що у випадках великої їх кількості займало години.

На зміну застарівшим рішенням прийшла технологія Berkley Packet Filter (далі BPF). BPF дозволяє писати безпечні програми, які можна безпечно виконувати безпосередньо у просторі ядра, та можуть комунікувати з програмами у користувацькому просторі. Первинна мета розробки даного рішення була можливість написання невеликих програм для керування мережевим трафіком, але технологія BPF також дозволяє вести розробку програм для трасування багатьох інших подій у ядрі. BPF надає не лише власний набір інструкцій, а й інфраструктурні елементи, такі як:

– власні структури даних – спеціальні таблиці, саме завдяки яким і відбувається взаємодія програм з простору ядра та користувацького простору;

– файли з допоміжним кодом (функції, константи, структури даних), які дозволяють взаємодіяти зі спеціальними функціями ядра;

– власна файлова система для зберігання таблиць та BPF програм;

– окрема інфраструктура, що дозволяє завантаження і вивантаження власне BPF програм.

На прикладі BPF програм для роботи з мережевим трафіком можна розглянути ще одну перевагу даної системи перед застарівшими аналогами. BPF програми можуть виконуються безпосередньо у просторі ядра, в найбільш ранній момент обробки ядром мережевого пакету, під час обробки мережевим

драйвером вхідного пакету. До цього моменту не було виконано жодного важкого для опрацювання етапу звичайної обробки пакету: ні виділення пам’яті, ні ініціалізації структури ядра skb, що являє собою власне пакет, ні відправлення skb на мережевий стек. Ця можливість представлена новим фреймворком для технології BPF під назвою XDP (eXpress Data Path) – фреймворк, що дозволяє максимально можливу, з точки зору програмного забезпечення, ефективну обробку мережевих подій. Дані можливості цього фреймворку є критичною перевагою при виборі інструментів для розробки рішень для високонавантажених систем, систем які потребують швидкої обробки багатьох подій у короткий час, та систем на базі обмежених апаратних потужностей.

Користувацькі програми, що виконуються у просторі ядра, можуть викликати нестабільну поведінку системи, наприклад, шляхом доступу до некоректної області пам’яті. Тому користувацькі програми мають бути гарантовано безпечними, і це завдання лежить на спеціальному верифікаторі.

BPF верифікатор виконує верифікацію у два етапи, на першому етапі виконує ряд перевірок:

– перевірка на перевищення максимальної кількості інструкцій в програмі;

– перевірка на наявність циклів (хоча слід зазначити, що прості цикли, у яких верхня межа циклу є константою та умова циклу є простим вираженням, можуть бути розвернуті завдяки директивам компілятора);

– перевірка на наявність викликів сторонніх функцій, та користувацьких функцій, що не являються inline-функціями;

– недосяжні інструкції.

Деякі з вищезазначених перевірок відбуваються завдяки побудові спрямованого ациклічного графу.

На другому етапі виконується перевірка на коректність та не вихід за межі структури через доступ по вказівнику.

Завдяки вищезазначеним аспектам стало можливо безпечно запускати користувацький код у просторі ядра під час виконання, тобто без необхідності

перекомпіляції ядра та його модулів кожний раз, коли виникає необхідність у зміні, створенні чи розширенні логіки програм трасування, фільтрації мережевого трафіку чи балансування навантаження.

В 2014 році було представлено eBPF (extended Berkley Packet Filter). Серед нововведень були адаптація під сучасне апаратне забезпечення, збільшення кількості регістрів у віртуальній машині BPF з двох 32-бітних регістрів до десяти 64-бітних. Ці зміни швидко стали стандартом та надалі, при кожному згадуванні технології BPF, суспільство має на увазі саме eBPF.

Наразі доступні деякі додаткові інструменти для поліпшення розробки BPF програм. Наприклад, з'явилася можливість написання даних програм не у вигляді BPF інструкцій, що мають вигляд коду на асемблері, а у вигляді підмножини мови C. Програми, написані на даній підмножині мови C компілюються в BPF інструкції завдяки LLVM. Після компіляції, код може бути оптимізований спеціальним JIT компілятор у ядрі для більш ефективної роботи програм.

В даній кваліфікаційній роботі технологія BPF є головною складовою, back end частиною системи для налаштування фільтрації та трасування мережевого трафіку. Для зручного користування кінцевим користувачем даної системи, розроблена front end частина, з використанням мови програмування Java, веб-фреймворком Micronaut. Micronaut є фреймворком загального призначення для розробки легких і потужних веб-додатків, а разом с кросплатформеною мовою програмування Java буде забезпечено ефективне рішення для будь-яких апаратних потужностей.

1.2. Призначення розробки та галузь застосування

Під час виконання кваліфікаційної роботи було поставлено завдання розробити «Розробка програмного забезпечення для фільтрації та трасування трафіку з використанням технології BPF в Linux».

Причиною виникнення необхідності розробки програмного забезпечення являється відсутність доступних аналогів в країні, що забезпечують роботу на основі невеликих апаратних потужностей та є простими у використання для кінцевого користувача.

Галузь застосування даного продукту – налаштування домашніх, або корпоративних комп'ютерних мереж.

Призначення даної розробки – це надання зручного та інтуїтивного інтерфейсу для взаємодії з інформаційною системою для адміністраторів комп'ютерних мереж.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

– спеціальність 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 317-с від 07.06.2021 р;

– завдання на кваліфікаційну роботу на тему «Розробка програмного забезпечення для фільтрації та трасування трафіку з використанням технології BPF в Linux».

1.4. Постановка завдання

Метою кваліфікаційної роботи є створення зручного користувацького інтерфейсу інформаційної системи для налаштування фільтрації та трасування

мережевого трафіку, що реалізує можливість блокування трафіку по комбінації IP-адреси та порту, по місцезнаходженню, по протоколам та трасування трафіку для забезпечення моніторингу комп'ютерної мережі через простий веб-інтерфейс.

Система може використовуватися у домашніх мережах, корпоративних мережах малого та середнього бізнесу.

Адміністратор задає набір правил, по яким налаштовується доступ до корпоративної мережі, а система надає можливість здійснення налаштування по замовчуванню, спираючись на популярні користувацькі запити. Система також надає можливість, у вигляді вихідної інформації, моніторингу мережевого трафіку. Загалом, системою передбачено наявність лише однієї групи прямих користувачів – адміністраторів.

Результатом моніторингу комп'ютерної мережі є збір даних про зовнішні ресурси, до яких звертаються користувачі, про звертання до внутрішніх ресурсів з мережі Інтернет, статистика протоколів, циркулюючих в мережі тощо.

Таблиця 1.1

Вихідна моніторингова інформація для адміністратора

Вихідна інформація	**Призначення**
Час	Опис часу отримання пакету в форматі HH:MM:SS
Протокол	Протокол, за яким відбувалася комунікація
IP-адреса відправника	Публічна IP-адреса відправника
IP-адреса одержувача	Публічна IP-адреса одержувача

Основною сутністю в підсистемі фільтрації трафіку є правило. Опис структури сутності "правило" описано в таблиці 1.2.

Налаштування фільтрації трафіку в корпоративній чи домашній мережі є головним аспектом при забезпеченні безпеки приватних даних та запобігання несанкціонованого доступу. У разі необхідності вимкнення вказаних правил фільтрації, це можна зробити в спеціальному інтерфейсі адміністратором. Також, з ціллю зниження порогу входження для користування даною розробкою, передбачено попередньо встановлені стандартні правила фільтрації трафіку.

Таблиця 1.2

Вхідна інформація для налаштування правил фільтрації адміністратором

Вхідна інформація	**Призначення**
IP-адреса	IP-адреса користувача, якому буде заборонено доступ до кінцевих вузлів комп'ютерної мережі
Тип правила	Тип визначає для кого ip-адреса буде недоступна: для зовнішнього пристрою чи для користувача в локальній мережі

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Програмний продукт загалом повинен володіти наступними функціональними характеристиками:

- зручний та адаптивний інтерфейс користувача;
- відправка даних на сервер та обробка відповідей від нього;
- зберігання протягом певного часу даних моніторингу мережі;
- фільтрація трафіку згідно з заданими правилами;

– у разі доступу користувачем заблокованого ресурсу в мережі, відбувається перенаправлення користувача на спеціальну сторінку.

– Адміністраторам надані наступні функції:

– можливість створення нових адміністраторів мережі;

– можливість архівування результатів моніторингу мережі;

– можливість переглядання результатів моніторингу мережі;

– можливість налаштування правил фільтрації трафіку в мережі в діалоговому режимі.

1.5.2. Вимоги до інформаційної безпеки

Захист персональних даних адміністрації, архіву результатів моніторингу повинен здійснюватися за допомогою штатних механізмів розподілення доступу передбачених операційною системою та серверним програмним забезпеченням – входу до особистого кабінету адміністратора за допомогою логіну та паролю. Доступ до особистого кабінету можуть мати лише ті вузли, які знаходяться в локальній мережі.

Захист інформації під час інформаційного обміну між розроблюваним застосунком та серверною частиною повинен забезпечуватися шляхом використання штатного захищеного протоколу обміну інформацією https.

1.5.3. Вимоги до складу та параметрів технічних засобів

У якості апаратної бази для програмного забезпечення даної системи фільтрації та моніторингу мережевого трафіку достатньо апаратних потужностей одноплатного комп’ютера серії Raspberry Pi 3, або його аналогів.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормальної роботи веб-клієнта, що розробляється, необхідна наявність наступних речей:

- будь-яка операційна система, на яку можна встановити браузер;
- стабільне підключення до мережі Інтернет;
- встановлений на комп’ютер браузер.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути розробка, що складається з двох частин: Linux утиліта та веб-додаток на базі Java, а також Spring Framework. Кінцевий користувач взаємодіє безпосередньо з веб-додатком.

Кінцева програмна система, що є результатом виконання кваліфікаційної роботи має наступні функціональні можливості:

– ведення моніторингу трафіку в локальній мережі;

– зберігання даних моніторингу мережі в спеціалізованій базі даних PostreSQL;

– налаштування фільтрації мережевого трафіку шляхом створення правил фільтрації мережевого трафіку;

– редагування існуючих правил фільтрації;

Кінцева програмна система буде також мати наступне експлуатаційне призначення:

– автоматизація налаштування фільтрації та трасування мережевого трафіку;

– надання простого та інтуїтивного інтерфейсу для кінцевих користувачів;

– можливість оптимізації чисельності персоналу за рахунок системних адміністраторів;

– забезпечення елементарної безпеки корпоративних ресурсів в локальній мережі;

2.2. Опис застосованих математичних методів

У розробленому програмному продукті не використовувались математичні методи, тому що вимогами до розробки не було регламентовано ніяких задач математичного характеру.

2.3. Опис використаної архітектури та шаблонів проектування

Розроблена програмна система має два головних компоненти, які працюють тільки за повної наявності та функціонуванні обох частин, хоча веб-додаток та Linux утиліта реалізовані у вигляді двох окремих проектів, навіть з використанням різних мов програмування та фреймворків.

Back-end частина система, Linux утиліта, запускається та працює після коректного налаштування користувацьких правил, або правил за замовчуванням, для фільтрації мережевого трафіку за допомогою веб-додатку, що є front-end частиною системи.

Дані два компоненти продукту можуть бути розгорнені на двох різних пристроях, але лише за тією умовою, що ці два пристрої знаходяться в одній локальній мережі. Така модель розгортання програмних систем є оптимальною, особливо якщо брати до уваги можливе високе навантаження на апаратне забезпечення, що виконує функції фільтрації, аналізу та трасування мережевого трафіку.

Взаємодія Linux програми маршрутизації трафіку та веб-додатку відбувається за допомогою протоколу WebSocket. WebSocket забезпечує повну двосторонню взаємодію двох вузлів поверх одного TCP з’єднання. Для налаштування зв’язку по протоколу WebSocket клієнту необхідно за допомогою протоколу HTTP відправити запит на WebSocket handshake, і уразі успіху сервер повертає HTTP відповідь з кодом 101 Switching protocol.

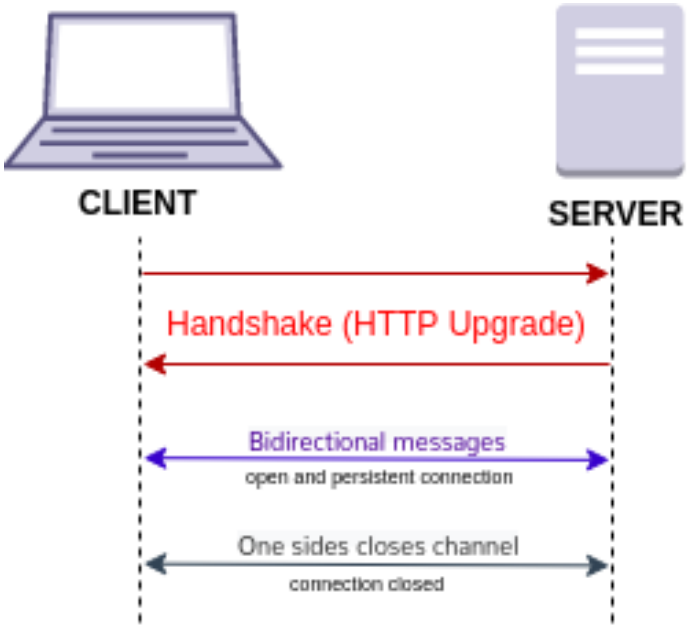

Рис. 2.1. Схема взаємодії пристроїв за протоколом WebSocket

Взаємодія кінцевого користувача з розробленою системою, відбувається з використанням інтернет протоколу прикладного рівня HTTP та клієнт-серверної архітектури. Клієнт-серверна взаємодія визначається наступним чином:

– клієнт спілкується с сервером в шаблоні запит-відповідь; клієнт завжди виступає ініціатором комунікації;

– протокол, згідно з яким відбувається взаємодія, має прикладний рівень;

– сервер має реалізувати чітко визначений набір методів для використання клієнтом, тобто прикладний програмний інтерфейс (API).

Останній з вищезазначених пунктів має бути здійснений шляхом відкриття деякого набору ендпоінтів для веб-клієнта, при звертанні до яких формується відповідь згідно з визначеним форматом. Звертання до таких ендпоінтів має відбуватися з використанням передбачених методів протоколу HTTP.

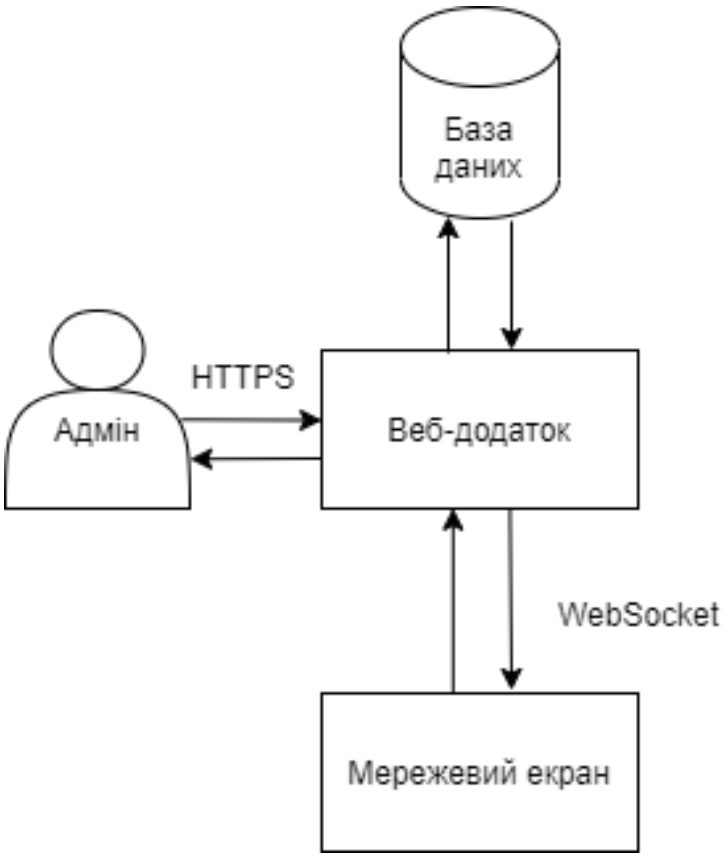

Рис 2.2. Схема взаємодія користувача, веб-додатку, бази даних та мережевого екрану

Розроблена Linux утиліта має модуль, завдяки якому вона ініціює запити до сервера та виступає у ролі клієнта, а веб-додаток виступає у ролі сервера. Їх взаємодія здійснена з використанням деяких елементів архітектурного стилю REST (Representational State Transfer, тобто «передача репрезентативного стану»). Загалом, REST складається з шести принципів:

– Використання клієнт-серверної взаємодії.

– Відсутність інформації о стані користувачах на сервері, а вся необхідна інформація для обробки запиту клієнта має бути наявна у самому запиті.

– Уніфікований інтерфейс взаємодії клієнта з сервером.

– Абстракція від джерела відповіді. Сучасні комп’ютерні мережі можуть мати багато проміжних вузлів між клієнтом та сервером, наприклад, кеш чи балансувальник навантаження.

– Зберігання результатів обробки запитів клієнтів у тимчасовому кеші, наприклад Redis. Звісно, у випадках, коли одні й ті ж самі запити клієнтів з плином часу можуть повертати різні результати від серверу, кешування не є можливим, а тому даний пункт є опціональним.

– Можливість розширення функціоналу front-end частини системи шляхом завантаження додаткового коду чи скриптів для виконання. Цей пункт також є опціональним.

Якщо кожний з цих пунктів (окрім опціональних) виконується при проектуванні взаємодії компонентів розподіленого додатку в мережі, то таку систему можна назвати RESTful. Дані принципи допомагають оптимізувати взаємодію окремих компонентів системи, а також забезпечити великий потенціал для масштабування. Окрім цього, прозорість зв’язку між компонентами та стійкість системи перед відмовою окремих компонентів забезпечують підвищену надійність розподіленого додатку.

Програмну систему, що представлена в даній кваліфікаційній роботі, можна віднести до таких RESTful систем.

Якщо розглядати окремо кожний компонент даного програмного продукту, то слід зупинитися на огляді архітектури веб-додатку, написаного з використанням мови програмування Java. Зараз доцільно розглянути лише декілька використаних фреймворків, тому що саме вони задають шаблони для проектування програм. Мова йде про Spring Core та Spring MVC.

Spring Core є головним фреймворком в своїй групі, тому що він дозволяє використовувати так звану інверсію керування (IoC, Inversion of Control).

Інверсія керування – це ще один принцип об’єктно орієнтованого програмування, що надає можливість за допомогою спеціальних фреймворків реалізувати логіку виконання програми та створення бізнес об’єктів в спеціально наданих callback-методах, тим самим відокремити та абстрагуватися від інших частин програми, підвищити можливість масштабування та здатність до тестування окремих модулів програми;

На Spring Core базується наступний фреймворк – Spring MVC. Він задає MVC шаблон для проектування саме тієї частини програми, що відповідає за взаємодію в мережі. Згідно з MVC, програма поділяється на три частини:

– Model (модель) постачає дані, відповідає за їх обробку чи трансформацію, незалежно від користувацького інтерфейсу. Містить бізнес логіку програмного продукту;

– View (відображення даних для користувача) реагує на зміни в моделі;

– Controller (контролер) відповідає за обробку дій користувача, викликає методи моделі для виконання бізнес логіки.

Даний шаблон проектування надає можливість відокремлення бізнес логіки від відображення даних, ізолює кожну частину програми, тим самим зменшує скованість цих частин, знов-таки підвищуючи потенціал для масштабування, збільшує можливе покриття тестами, тощо.

З плином часу кількість коду в програмному продукті невпинно зростає, а при недалекоглядній архітектурі також експоненціально зростає складність розробки, а разом з нею і вартість. Тому багато разів згадані масштабування, абстрагування та інші аспекти продуманого проектування є необхідними з точки зору бізнесу.

Інша частина розробленого програмного продукту, мережевий фільтр у вигляді Linux утиліти, має дещо простішу будову. Можна виділити декілька головних модулів.

Перший модуль відповідає за обробку аргументів при запуску програми. Ці аргументи визначають які саме фільтри необхідні для налаштування локальної мережі: фільтр занесених до чорного списку зовнішніх ресурсів чи сторінок, протоколів, тощо. Також потребують налаштування правила трасування мережі. Наприклад, трасування пакетів певних протоколів чи трасування здійснених підключень по протоколу TCP. Також, перший модуль відповідає за збір інформації з програм моніторингу та надання їх наступному модулю.

Другий модуль відповідає за взаємодію даної утиліти з сервером, який агрегує, зберігає в базі даних необхідну інформацію та аналізує її.

2.4. Опис використаних технологій та мов програмування

Опис технологічного стеку програмного продукту варто розпочати с використаних мов програмування, тому що вони складають основу для подальшого вибору бібліотек та фреймворків.

Як вже згадувалося раніше, розроблена система складається з двох головних частин:

– back-end утиліти для мережевого екрану написаної на Python, підмножині мови C та спеціальному асемблері BPF;

– front-end веб-додатку з використанням мови програмування Java й html-шаблонізатором Thymeleaf.

При виборі інструментів для розробки мережевого фільтру було досліджено декілька можливих варіантів технологічного стеку.

По-перше, основою будь-якого програмного продукту на будь-якій мові програмування є операційна система. Серед доступних варіантів доступні операційна система Windows 10 та операційна система на базі ядра Linux. Серед цих двох варіантів безумовним лідером є саме Linux з кількох головних причин:

– Linux – це проект ядра операційної системи з відкритим кодом, а тому будь-хто може власноруч провести дослідження програмних кодів для пошуку помилок, вразливостей системи.

– Завдяки пункту вище, дистрибутиви на базі Linux мають безліч корисних безкоштовних утиліт та інших програмних продуктів з відкритим кодом.

– Надійність та відмовостійкість. Багато серверів в мережі Інтернет працюють роками та жодного разу не перезавантажувалися.

– Безпека. Будь-який веб-сервер відкриває свої порти для того, щоб приймати запити від користувачів та здійснювати їх обробку певним чином. Але відкриваючи сервер для доступу з зовні, він одразу стає вразливим перед несанкціонованими діями зловмисників.

– Гнучкість налаштування. З Linux є можливість відмовлятися від певних модулів ядра для полегшення системи, або замінювати компоненти за власним бажанням. Користувачі в змозі встановлювати будь-яке середовище робочого столу з власним набором допоміжних утиліт, наприклад KDE чи GNOME. Яскравим прикладом є порівняння дистрибутивів Debian та Alpine Linux. Перший дистрибутив, за оцінками фахівців, станом на версію 2.2 (її реліз відбувся 14 серпня 2000 року), мала більше 55 мільйонів рядків програмного коду, а в переведенні на людино-години та долари США – це 14 тисяч людино-років та 1.9 мільярдів доларів. Другий дистрибутив цікавий тим, що в ньому розробники відмовилися від багатьох модулів ядра, використовують спеціальну бібліотеку мови С, власний пакетний менеджер, реалізували власний механізм безпеки, що запобігає виникненню цілих класів вразливостей в системі, і завдяки цьому домоглися зменшення образу ядра Linux до 5-8 мегабайт.

– Підтримка великої кількості комп’ютерних архітектур та апаратних засобів.

Серед великої кількість дистрибутивів Linux було обрано Ubuntu 20.04. Такий вибір зумовлений популярністю даного дистрибутиву, але задовольнити умови, поставлені перед операційною системою для виконання даної кваліфікаційної роботи, зможе будь-який інший популярний дистрибутив. Головною вимогою є підтримка технології extended BPF ядром Linux, тобто версія ядра не може бути менша ніж 4.5. Взагалі, чим вища версія ядра, тим більш зручнішою та стабільною буде розробка даного програмного продукту, за рахунок виправлення старих помилок, або багів, та введення нових допоміжних функцій для технології BPF.

Перевірити чи підтримує поточна Linux система технологію BPF можна шляхом перевірки значень конфігураційних опцій. Результат виконання команди має співпадати з зазначеними на рис. 2.3 опціями:

```
grep BPF /boot/config-`uname -r`
```

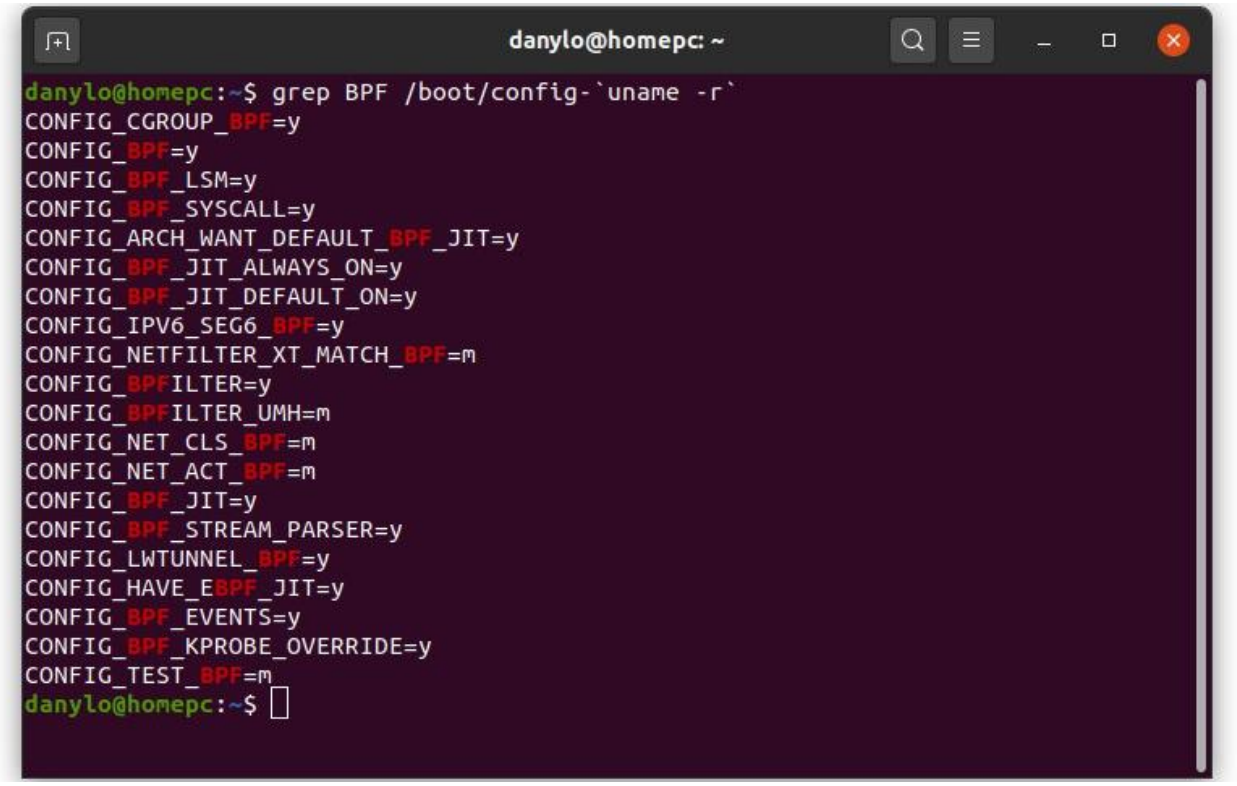

Рис. 2.3. Правильна конфігурація системи

Другим важливим компонентом розробленого програмного комплексу є технологія для написання власне правил фільтрації та трасування мережевого трафіку.

Впродовж багатьох років безумовним лідером серед доступних інструментів для фільтрації трафіку була утиліта iptables. Але з появою BPF ситуація одразу змінилась. BPF дозволяє писати максимально ефективний користувацький код, який завантажується динамічно під час виконання, тому дозволяє запобігти перебудови коду ядра, або довгого переналаштування вже існуючих правил фільтрації, як у випадку з iptables. Також, BPF забезпечує більшу гнучкість за рахунок того, що розробники мають реалізовувати логіку фільтрації шляхом написання власних програм. Автори BPF, звісно, розуміли потенційну небезпеку динамічно завантажуваного коду в ядро, тому з перших версій вони також постачали власний верифікатор BPF коду. Даний верифікатор має перевірити користувацьку програму на відсутність нескінченних циклів, нелегального доступу до ділянок в пам’яті, а також перевіряє чи виконується

існуюче обмеження на кількість інструкцій в одні програмі. Після того, як верифікатор вдало перевіряє код програми, запускається JIT-компілятор, який трансформує асемблер BPF у машинний код, тим самим запобігає надлишкових витрат на інтерпритацію під час виконання.

Окрім використання технології BPF для створення власних мережевих фільтрів, вона також підтримує завантаження користувацького коду в безліч інших місць в ядрі для трасування подій завдяки спеціальним типам BPF програм.

Наприклад, є можливість використовувати так звані kprobe, kretprobe. Kprobe – це спеціальний тип користувацьких BPF програм, які кріпляться до певних функцій ядра і виконуються при викликах вказаних функцій. Наприклад, при з'єднанні kprobe з функцією tcp_v4_connect(), перед кожним її викликом, тобто при кожній спробі налаштування TCP з'єднання, виконується користувацький код, який має в собі певну логіку трасування. Аналогічно відбувається робота з kretporbe, за лише тим виключенням, що програми даного типу викликаються вже після виходу з вказаних функцій ядра. При використанні kprobe і kretprobe варто розуміти, що вони не є абсолютно стабільними точками входу через те, що існує вірогідність змін в коді ядра в нових версіях, які приводять до застарівання певних системних функцій та відмови від них.

Дзеркальним відображенням типів krpobe і kretprobe на користувацький простір є uprobe і uretprobe. Коли користувач створює власну програму типу uprobe або uretprobe, ядро розміщує так звану пастку біля вказаної інструкції. Коли потік виконання доходить до цієї пастки, ядро створює спеціальну подію, в якій користувацька BPF функція вказана як callback.

Серед підтримуваних типів програм для трасування також існують Tracepoints. Tracepoints дозволяють прикріпляти BPF програми до спеціальних статичних обробників в ядрі, які створюються з метою трасування та дебагу. Усі tracepoint визначені в директорії /sys/kernel/debug/tracing/events. Загалом, такий тип програм є менш гнучким, але більш передбачуваними, так як після створення таких статичних обробників, вони гарантовано залишаються в ядрі.

Нарешті, BPF надає можливість використовувати такий тип програм як Express Data Path. XDP програми використовуються в мережевому стеку Linux, а головна їх особливість полягає в тому, що вони дозволяють прикріпляти користувацькі програми до обробників, розташованих на максимально ранніх етапах обробки мережевих пакетів. Так, XDP програми виконуються всередині драйверів мережевих інтерфейсів відразу після обробки переривання, перед виділенням пам'яті для структури socket buffer і її ініціалізацією. Запобігаючи таким дорогим з точки зору процесорного часу операціям, вдається ефективно реалізувати необхідну логіку для високонавантажених систем. XDP визначає низку доступних дій для оброблюваного пакету:

– XDP_DROP виконує відкидання пакету. Відкидання пакету якомога раніше є ключовим фактором ефективного запобігання випадків denial-of-service;

– XDP_PASS дозволяє пакету проходити далі до мережевого стеку;

– XDP_ABORTED виконує відкидання пакету та кидає tracepoint exception;

– XDP_TX змушує пакет повернутися до того мережевого інтерфейсу, з якого він надійшов;

– XDP_REDIRECT змушує пакет відправитися до іншого мережевого інтерфейсу або сокету в користувацькому просторі.

Користувацька XDP програма має повертати цілочисельний код, який відповідає одній з вищевказаних дії.

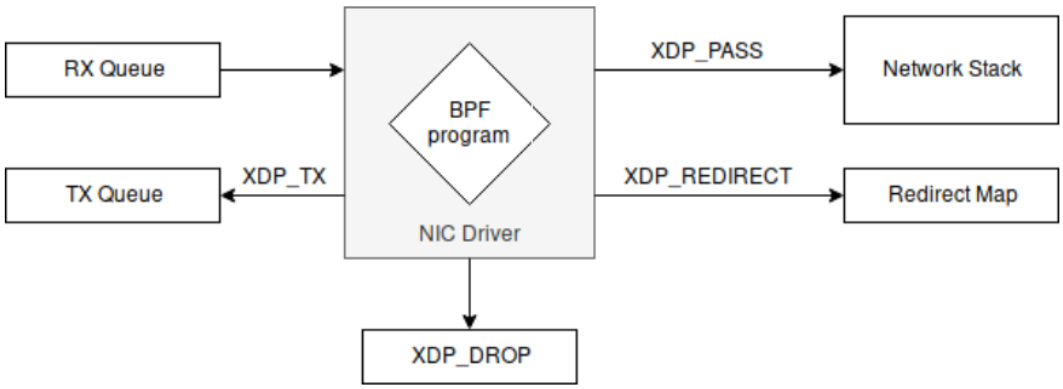

Рис. 2.4. Схема доступних дій XDP

Сьогодні XDP підтримує розміщення користувацьких програм лише для взаємодії з вхідним мережевим трафіком. Але, наразі існує ініціатива, мета якої – забезпечити підтримку XDP для вихідного трафіку. Згідно з цією ініціативою, буде запроваджено новий attachment type BPF_XDP_EGRESS, а також спеціальний прикладний програмний інтерфейс для if_link для прикріплення таких користувацьких програм до мережевих пристроїв.

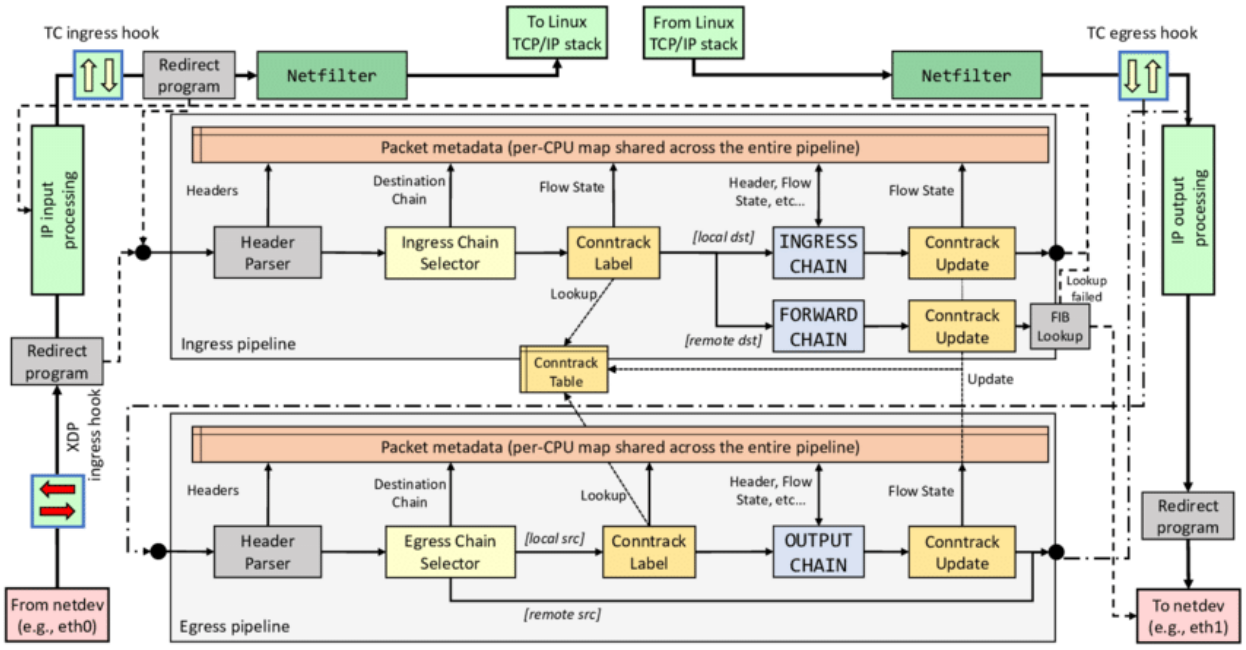

Рис. 2.5. Схематичне зображення хуків в Linux Networking Stack

Іншою важливою технологією, використаною при розробці системи для фільтрації та трасування мережевого трафіку є Traffic Control в Linux. Traffic Control – це загальна назва набору систем і механізмів для класифікації та обробки пакетів мережевого трафіку. На відміну від XDP, Traffic Control підтримує обробку як вхідного трафіку, так і вихідного. Traffic Control дозволяє вирішувати який пакет відкинути, який пакет відправити далі до мережевого стеку Linux та з якою частотою це відправлення відбувається.

BPF підтримує завантаження не однієї, а багатьох користувацьких програм завдяки tail calls. Tail calls – це виклики зовнішніх BPF програм зсередини інших

BPF програм, таким чином створюючи своєрідний ланцюг з цих програм, що дозволяє обійти стандартне обмеження в 4096 інструкцій на одну програму. Це обмеження створене з метою гарантування розумного часу виконання користувацького коду в ядрі і запобіганню блокування потоків виконання ядра, що може привести до помилок і аварійної зупинки системи, які було б дуже важко відстежити, а тому й виправити.

Контекст BPF програми – це аргумент користувацької функції, який вводиться ядром Linux при її виклику. Програми кожного типу (kprobes, uprobe, tracepoint, xdp) мають власний відмінний тип очікуваного аргументу. Наприклад, у випадку програм типу XDP, в якості аргументів передається структура xdp_md, що являє собою пакет, який щойно надійшов на мережевий інтерфейс.

```
/* user accessible metadata for XDP packet hook
 * new fields must be added to the end of this structure
 */
struct xdp_md {
        __u32 data;
        __u32 data_end;
        __u32 data_meta;
};
```

Рис. 2.6. Структура xdp_md

Коли відбувається виклик BPF програми з іншої BPF програми, ядро повністю оновлює їх контекст, а тому неможливо таким чином здійснити взаємодію цих двох програм. Цей фактор зумовив появу спеціальних структур даних типу асоціативного масиву, відомих як BPF-таблиці, які буде описано більш детально в наступних розділах.

Технологія BPF надавала можливість писати користувацький код лише за допомогою власного байт коду. Для наочності, на рис 2.6 вказана схема виконання простої програми Hello World з використанням байт-коду BPF. Такий спосіб створення програм не є оптимальним тому, що байт код є доволі низькорівневим інструментом програмування і через це розробка програм може займати набагато більше часу, ніж, наприклад на C. Тому з метою оптимізації

роботи було запроваджено можливість компіляції програм, написаних на спеціальній підмножині мови С, у байт код BPF за допомогою LLVM/Clang.

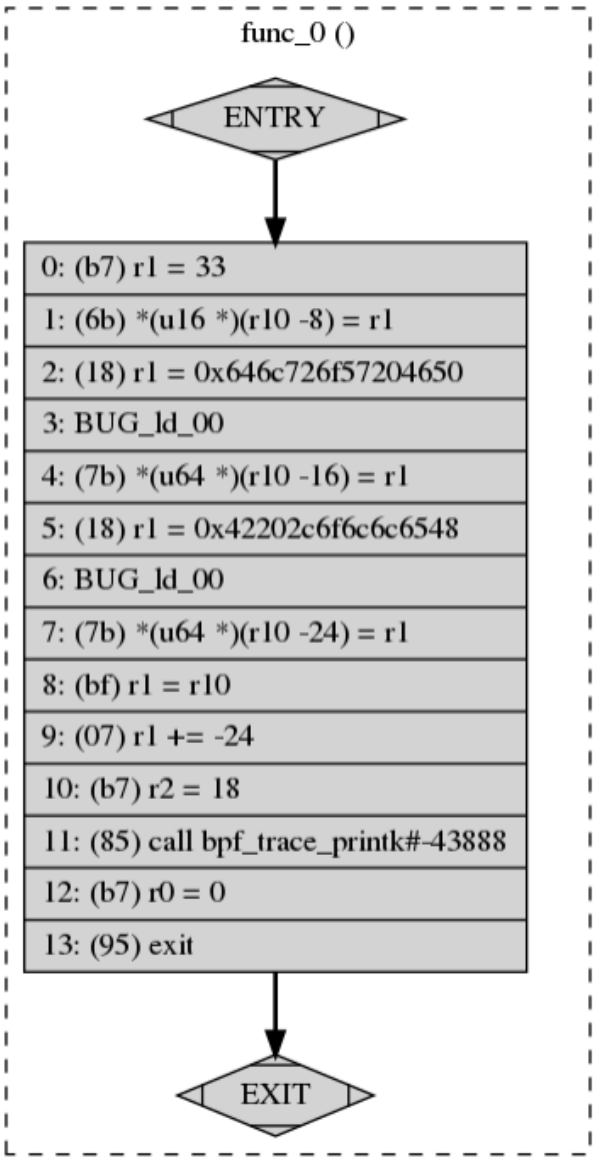

Рис. 2.7 Схема виконання програми Hello World в BPF

Наступною використаною технологією при розробці програмного продукту для кваліфікаційної роботи є бібліотека BCC. BCC – це набір інструментів на Python для створення BPF програм, який полегшує лінкування користувацьких програм з бібліотеками з простору ядра, полегшує компіляцію та завантаження програм в ядро, забезпечує корисні приклади коду для вирішення шаблонних

задач, а також надає інструменти для більш ефективного дебагу та діагностики проблем з продуктивністю програм.

Front-end частина розробленої системи створена з використанням об’єктно-орієнтованої мови програмування високого рівня Java. Головними аргументами в користь використання даної мови програмування є її кросплатформеність, висока ефективність та багатий набір готових інструментів для поліпшення розробки веб-додатків. Окрім Java, використовувались наступні бібліотеки і фреймворки на її базі:

- Spring Core;
- Spring MVC;
- Spring Data;
- Spring Boot;
- Thymeleaf.

Як вже згадувалося раніше, Spring Core являється головним фреймворком, який дозволяє використовувати власний Inversion of Control контейнер та біни, поліпшує роботу з ресурсами та інтернаціоналізацією тощо.

Spring MVC – це фреймворк для реалізації динамічних веб-сайтів, веб-додатків, сервісів тощо. Він заснований на стандартному Java Servlet API і входить у родину Spring фреймворків з самого початку.

Spring Data поліпшує взаємодію веб-додатку з базою даних, автоматизує конвертацію сутностей, описаних в таблицях, в прості Java об’єкти, підтримує декларативний менеджмент транзакцій.

Spring Boot поліпшує створення самостійних додатків на базі Spring. Зокрема, він включає в себе готові контейнери сервлетів, такі як Tomcat чи Jetty, а тому немає необхідності деплоїти WAR файли власноруч, поліпшує конфігурацію додатку шляхом запровадження Spring profiles та автоматизацією конфігурації бібліотек, а також дозволяє відмовитися від застарівших і незручних XML конфігурацій.

2.5. Опис структури програми та алгоритмів її функціонування

Розроблений програмний продукт складається з двох модулів: веб-додаток на базі Java та Linux-утиліти на базі Python. Ці два модулі мають бути розвернуті на двох окремих комп’ютерах, а їх взаємодія відбувається по протоколу WebSocket в локальній мережі.

2.5.1. Структура проектів

Проект Linux-утиліти має структуру, зображену на рис. 2.8.

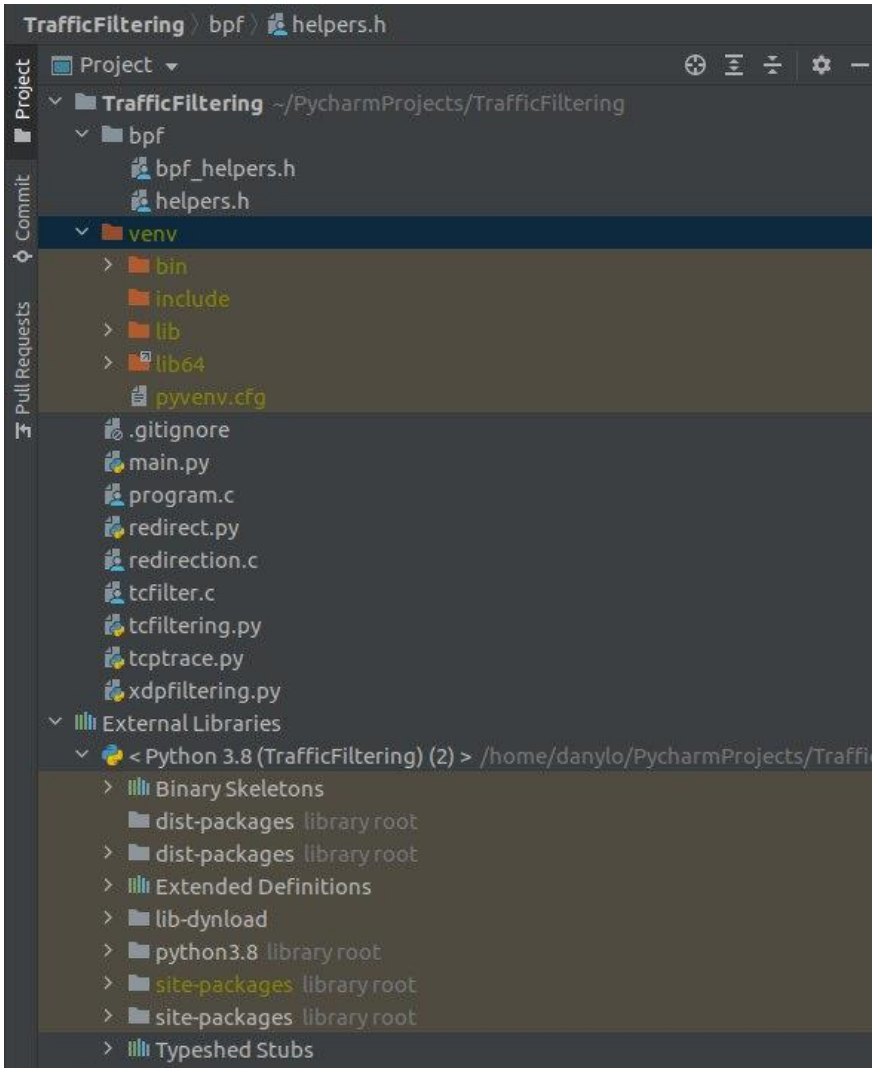

Рис. 2.8. Структура Linux- утиліти

Директорія bpf містить допоміжні header-файли, написані мовою C.

Директорія venv – це, так зване, віртуальне середовище Python, який містить пакети, від яких залежить поточний проетк, а також самий Python інтерпретатор різних версій.

Далі представлені власне файли, що формують кодову базу проекту. Це допоміжні Python скрипти, що забезпечують завантаження BPF програм та взаємодію користувацького простору з цими програмами.

– Файл main.py містить код, який відповідає за взаємодію з веб-додатком на Java.

– Файли redirect.py та redirect.c відповідають за редірект користувачів в локальній мережі на сторінку, в якій описана помилка, якщо ресурс, до якого вони роблять запит є заблокованим адміністратором.

– Файли tcfilter.c, tcfiltering.py, xdpfiltering.py, program.c відповідають за блокування доступу з мережі Інтернет до внутрішніх ресурсів, а також блокування доступу користувачів локальної мережі до інших заблокованих ресурсів.

Веб-додаток на базі Java має структуру зображену на рис. 2.9.

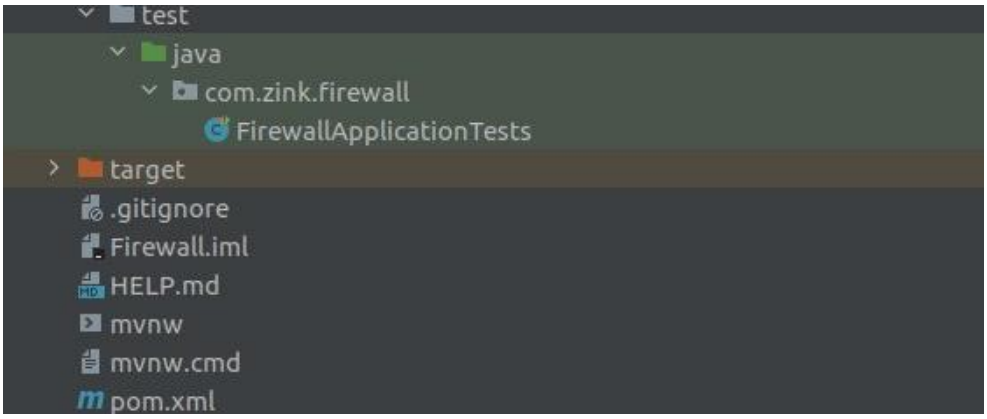

Рис. 2.9.Структура директорії тестів і допоміжних файлів

Рис. 2.10. Структура директорій з програмними кодами веб-додатку

Опис файлів та директорій в даній структурі:

Директорія .idea містить в собі індекси файлів проекту та інших метаданих, що необхідні для роботи інтегрованого середовища для розробки Intellij Idea.

Директорія .mvn містить необхідні плагіни та інші файли для роботи системи автоматизації управління та складання проекту Maven.

Наступний каскад директорій, що починається з директорії src/main містить власне коди розробленого веб-додатку, HTML-файли, SQL-файли, файли конфігурації та інші ресурси. Розглянемо детальніше:

– В директорії dao/entities розташовані Java-класи, що описують об’єкти, на базі яких будуть створені сутності у базі данних.

– Директорія dao/enums містить enum класи для вищезазначених класів Java, що описують існуючі ролі в системі та тип правил.

– В директорії dao/repos містяться Java інтерфейси, на базі яких будуть створені класи для взаємодії Java коду з базою даних (вибірка, зміна та створення нових данних).

– В директорії dtos містяться класи, що описують структуру об’єктів користувачів, логів, правил, які будуть передаватися іншим модулям програмного продукту.

– В директорії services містяться класи, що описують надані веб-додатком сервіси, тобто операції, окремо доступні для шару моделей, і не мають власного стану.

– В директорії utils містяться допоміжні класи.

– В директорії web містяться класи, що безпосередньо забезпечують взаємодію з іншими модулям в мережі, а також класи, що налаштовують цю взаємодію (конфіг-класи).

– В директорії resources/templates містять HTML шаблони, що формують користувацький інтерфейс веб-додатку.

– В файлі application.properties описані змінні середовища, url до бази даних, ім’я користувача тощо.

– В файлі data.sql описані тестові дані для заповнення бази данних.

– В директорії test розміщені тести веб-додатку.

– В директорії target розміщені скомпільовані та складені файли проекту.

– Файл .gitignore вказує git які файли не потрібно вносити у систему контроля версій.

– Файл pom.xml містить використані проектом залежності на інші бібліотеки, фреймворки, плагіни тощо.

2.5.2. Опис структури бази даних

База даних розробленої системи має просту будову і складається лише з 4 сутностей, зображених на рис. 2.11 у вигляді моделі «сутність-зв'язок».

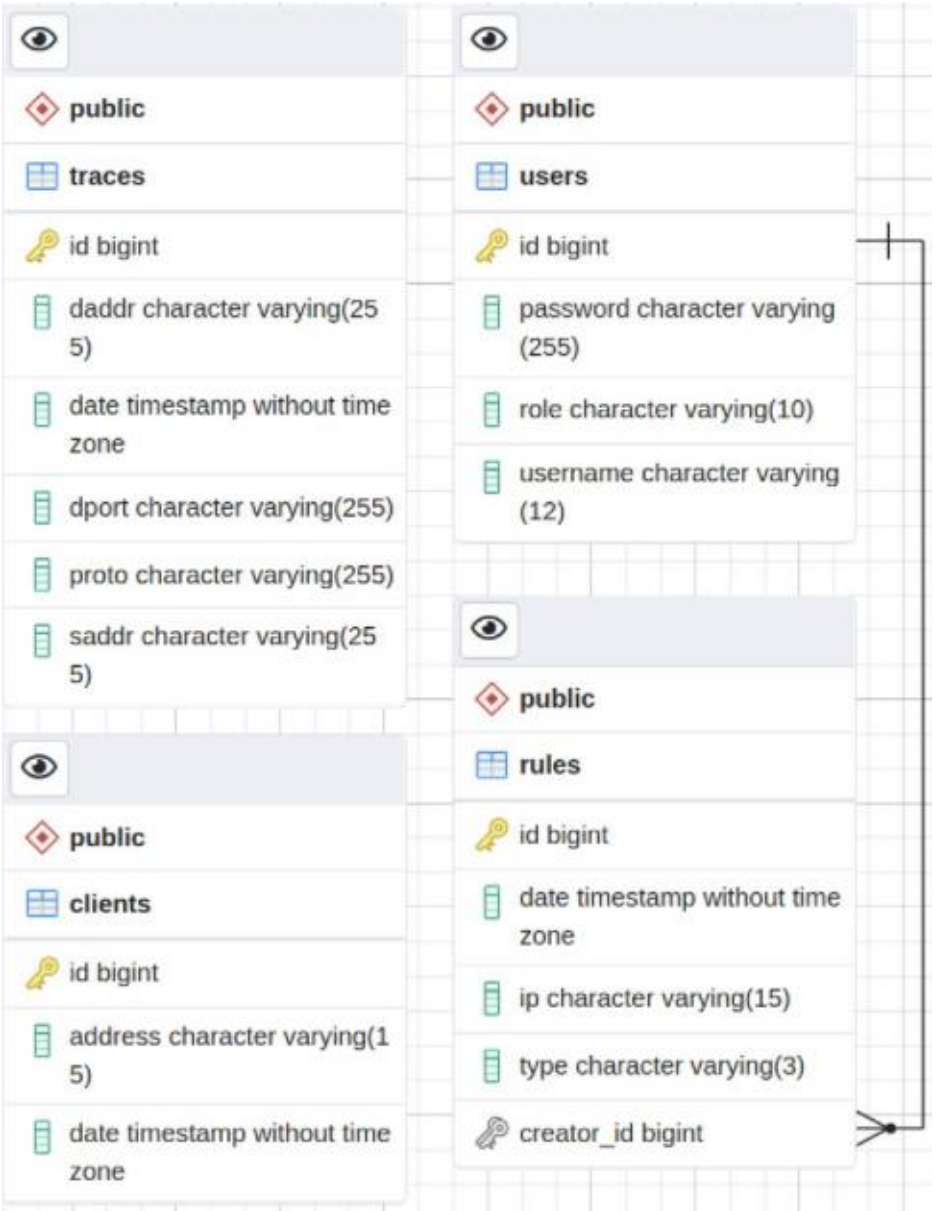

Рис. 2.11 ER-діаграма

2.5.3. Опис алгоритмів функціонування програмного продукту

Адміністратор локальної мережі може створювати правила фільтрації мережевого трафіку, забороняти користувачам локальної мережі доступ до певних ресурсів, переглядати журнал трасування мережевого трафіку. Всі ці дії відбуваються за допомогою користувацького інтерфейсу веб-додатку.

Після створення нового правила фільтрації трафіку, веб-додаток відправляє мережевому фільтру інформацію про нове правило, де воно оброблюється і динамічно, під час виконання, завантажується і починає свою роботу.

Під час користування користувачами локальної мережі, міжмережевий фільтр проводить трасування з’єднань користувачів з пристроями в мережі Інтернет. Результати трасування збираються і відправляються мережевим фільтром до веб-додатку, де вони оброблюються та зберігаються у базі даних. Пізніше, при запиті адміністратора, відбувається вибірка цих даних трасування та відображення їх у користувацькому інтерфейсі на спеціальній сторінці.

Варто також оглянути порядок налаштування системи та алгоритми функціонування на більш детальному, технічному рівні.

До комп’ютеру, на якому розвернутий мережевий фільтр, приладнається Wi-Fi адаптер, який пізніше у режимі HotSpot буде виступати у ролі роутера, до нього підключаються користувачі локальної мережі. В ході розробки використовувався Wi-Fi адаптер TL-WN722N. Перед початком користування необхідно встановити відповідні драйвери.

Після налаштування апаратних пристроїв відбувається порівняння доступних мережевих інтерфейсів з очікуваними, що зображені на рис. 2.12.

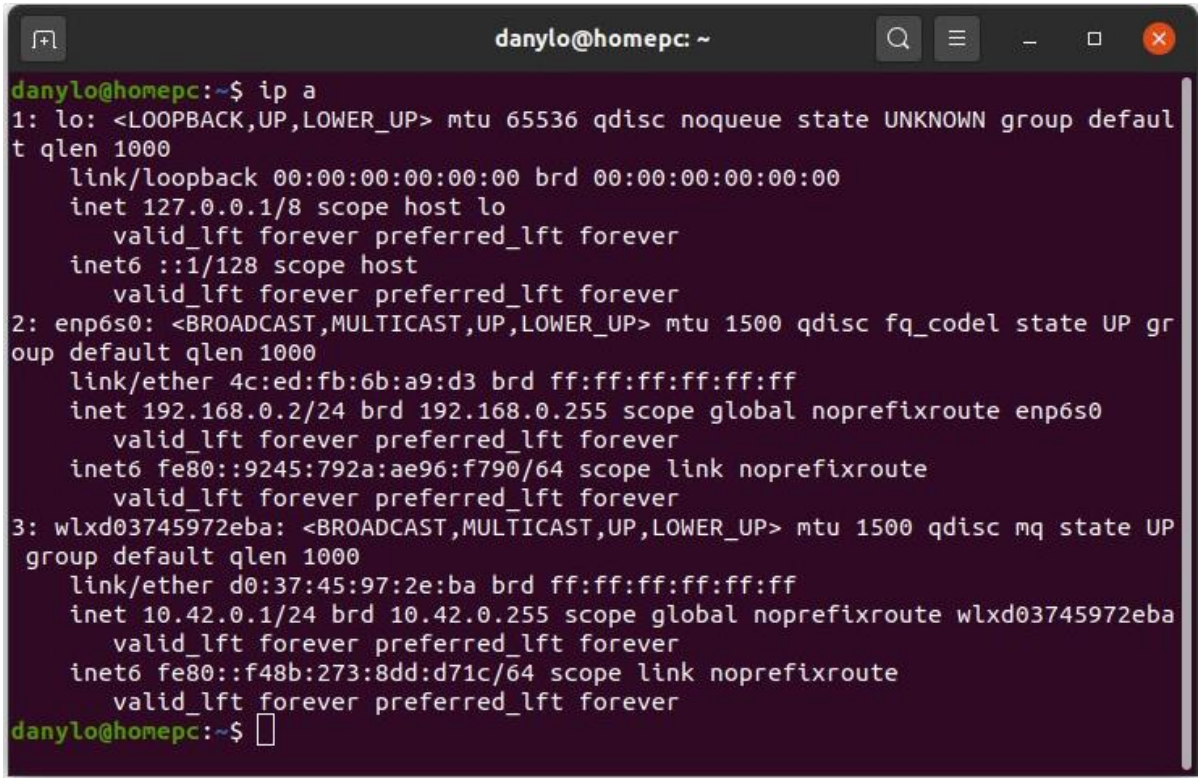

Рис. 2.12. Перелік мережевих інтерфейсів

Згідно з результатом виконання команди, приладнаний Wi-Fi адаптер має порядковий номер 3 та Interface Name wlxd03745972eba. Далі порядковий номер та назва інтерфейсу будуть використовуватись безпосередньо мережевим фільтром на Python.

На цьому етапі апаратні засоби готові до роботи с системою мережевого фільтру.

При старті системи відбувається завантаження вже існуючих правил фільтрації та трасування. Після цього, утиліта запускає умовно нескінчений цикл, в якому будуть послідовно збиратися логи трасування та відправлятися до веб-додатку.

При отримані нового правила фільтрації трафіку, розроблена Linux утиліта виділяє з нього ip-адресу заблокованого ресурсу, та зберігає її в спеціальних таблицях BPF.

На даному етапі для подальшого розуміння принципів роботи розробленої системи необхідно описати використані допоміжні функції BPF, алгоритм

роботи XDP та Traffic Control BPF програм, а також яким чином та завдяки чому відбувається взаємодія python програм з завантаженими програмами у ядро.

Перш за все, взаємодія користувацького простору з програмами у ядрі операційної системи Linux відбувається завдяки так званим BPF таблицям. Таблиці BPF – це структура даних типу асоціативного масиву, що дозволяє зберігати пари ключ та значення, та дозволяє виконувати операції пошуку по ключу, видалення та додавання елементів. Таблиці BPF також можливо використовувати у програмах користувацького простору. Для цього користувацьким програмам необхідно знати файловий дескриптор бажаної таблиці. Насправді, технологія BPF також має власну віртуальну файлову систему, яка монтується за шляхом /sys/fs/bpf. За замовчуванням деякі дистрибутиви не монтують цю файлову систему, тому у таких випадках необхідно виконати наступну команду:

mount -t bpf /sys/fs/bpf /sys/fs/bpf

Усі “файли” в цій файловій системі являються BPF об’єктами (тобто програми чи таблиці), а взаємодія з цими об’єктами відбувається за допомогою системного виклику bpf().

Ядро Linux сприймає ключ та значення BPF таблиць як звичайні бінарні дані, що не мають певного типу.

Для створення таблиці BPF необхідно використовувати багатофункціональний системний виклик bpf, який має наступну сигнатуру:

int bpf(int cmd, union bpf_attr *attr, unsigned int size)

Ця функція в якості аргументів приймає:

– Код операції, яку дана функція має виконати. У даному випадку це BPF_MAP_CREATE.

– Другий аргумент – це bpf_attr, union в термінах мови C, що складається декількох анонімних структур, що також використовуються різними командами системної функції bpf(). Вказані в union bpf_attr атрибути key_size та value_size надалі будуть використовуватися верифікатором під час завантаження програми,

щоб надати ядру гарантії, що при виклику будь-якої команди користувацький код не виходив за межі таблиці.

– Третій аргумент – це розмір union, що вказується користувачем другим аргументом.

Існує декілька можливих кодів команд для системного виклику bpf():

– BPF_MAP_CREATE створює нову таблицю і змушує системний виклик повернути файловий дескриптор, що посилається на щойно створену таблицю;

– BPF_MAP_LOOKUP виконує операцію пошуку за ключом в таблиці й повертає значення;

– BPF_MAP_UPDATE_ELEMENT створює або оновлює елементи, що відповідає вказаному ключу;

– BPF_MAP_DELETE_ELEM знаходить і видаляє елемент, що відповідає вказаному ключу;

– BPF_MAP_GET_NEXT_KEY знаходить елемент, що відповідає заданому ключу і повертає значення наступного елементу;

– BPF_PROG_LAOD змушує верифікатор перевірити вказану нову BPF програму та завантажити її у ядро, після чого повертає новий файловий дескриптор, що відповідає новій програмі.

Якщо системний виклик bpf() закінчився помилкою, то він повертає значення -1. Всього виділяються три причини помилки:

– якщо якийсь з атрибутів має неправильне значення, тоді ядро для системної змінної errno виставляє значення EINVAL;

– якщо користувач, що запускає виконання поточної програми має недостатньо прав, тоді системна змінна errno отримує значення EPERM;

– якщо недостатньо пам’яті для створення таблиці чи нового елементу для неї, errno дорівнює ENOMEM.

```
union bpf_attr {
    struct {    /* Used by BPF_MAP_CREATE */
        __u32         map_type;
        __u32         key_size;    /* size of key in bytes */
        __u32         value_size;  /* size of value in bytes */
        __u32         max_entries; /* maximum number of entries
                                      in a map */
    };

    struct {    /* Used by BPF_MAP_*_ELEM and BPF_MAP_GET_NEXT_KEY
                   commands */
        __u32         map_fd;
        __aligned_u64 key;
        union {
            __aligned_u64 value;
            __aligned_u64 next_key;
        };
        __u64         flags;
    };

    struct {    /* Used by BPF_PROG_LOAD */
        __u32         prog_type;
        __u32         insn_cnt;
        __aligned_u64 insns;      /* 'const struct bpf_insn *' */
        __aligned_u64 license;    /* 'const char *' */
        __u32         log_level;  /* verbosity level of verifier */
        __u32         log_size;   /* size of user buffer */
        __aligned_u64 log_buf;    /* user supplied 'char *'
                                     buffer */
        __u32         kern_version;
                                  /* checked when prog_type=kprobe
                                     (since Linux 4.1) */
    };
} __attribute__((aligned(8)));
```

Рис. 2.13. Структура юніону bpf_attr

Бібліотека Bpf Compiler Collection дозволяє простіше використовувати в програмах на Python таблиці, а також надає можливість використання власних типів BPF таблиць, які будуються на базі стандартних:

– BPF_TABLE має синтаксис BPF_TABLE(table_type, key_type, leaf_type, name, max_entries), де table_type – тип таблиці (наприклад, хеш-таблиця), key_type – тип ключа (наприклад, беззнакове ціле число), leaf_type – тип асоційованого значення (аналогічно до типу ключа), name – ім’я нової таблиці, max_entries – максимальна кількість елементів. Таблиця такого типу є стандартною, найчастіше використовується засобами бібліотеки для створення інших типів таблиць на її базі;

– BPF_HASH – звичайна хеш-таблиця на базі BPF_TAB. Синтаксис: BPF_HASH(name [, key_type [, leaf_type [, size]]]);

– BPF_ARRAY – таблиця, в якій у якості ключа міститься ціле число, таким чином імплементує інтерфейс індексованого масиву. Синтаксис: BPF_ARRAY(name [, leaf_type [, size]]);

– BPF_HISTOGRAM – гістограма. Синтаксис: BPF_HISTOGRAM(name [, key_type [, size]]);

– BPF_STACK_TRACE – таблиця трасування стека. Синтаксис: BPF_STACK_TRACE(name, max_entries);

– BPF_PERF_ARRAY створює perf-масив з максимальною кількістю елементів, вказаних у якості аргумента, і має дорівнювати кількості системних процесорів. Ці таблиці використовуються для отримання лічильників продуктивності апаратних пристроїв. Синтаксис: BPF_PERF_ARRAY(name, max_entries);

– BPF_PROG_ARRAY дозволяє зберігати в собі масив BPF програм. Кожний елемент цього масиву має значення файлового дескриптора або NULL. Масиви цього типу необхідні для виконання “стрибків” з однієї програми до іншої, або tail-call. Синтаксис: BPF_PROG_ARRAY (name, size);

– BPF_DEVMAP створений для зберігання в таблиці елементів ifindex, тобто назву мережевого інтерфейсу пристроїв. Цей тип таблиць може використовуватися лише в програмах типа XDP. Синтаксис: BPF_DEVMAP (name, size);

– BPF_ARRAY_OF_MAPS тип містить в собі таблиці, елементи яких є також таблицями. Метадані внутрішніх таблиць можуть містити будь-який тип таблиць, окрім BPF_MAP_TYPE_PROG_ARRAY. Синтаксис: BPF_ARRAY_OF_MAPS (name, inner_map_name, size).

Також варто розглянути структуру sk_buff, тому що вона використовується в кожній BPF програмі в Traffic Control. sk_buff – це структура, яка є буфером

вхідного чи вихідного пакету і використовується ядром Linux для обробки мережевого трафіку. Дана структура має наступні поля:

– перші два поля мають тип вказівника на структуру sk_buff і ім’я next та prev відповідно. Вони потрібні для надання можливості переходу до наступного та попереднього елементу типу sk_buff, аналогічно до зв’язаного списку;

– третє поле має тип вказівника на структуру sock і вказує на сокет, до якого або з якого надходить пакет;

– наступне поле має тип timeval і вказує на момент в часі коли пакет було отримано чи відправлено сокетом;

– наступні три поля мають тип вказівника на структуру net_device і мають назву dev, input_dev, real_dev. Вони вказують на device пов’язаний з пакетом. Мається цілих три поля для випадків зміни поля dev через інкапсуляцію чи декапсуляцію з віртуальних пристроїв;

– визначення наступних трьох полів зображено на рис. 2.14. Ці поля необхідні для встановлення вказівників на структури, що описують протоколи використані поточним мережевим пакетом;

```
union {
        struct tcphdr   *th;
        struct udphdr   *uh;
        struct icmphdr  *icmph;
        struct igmphdr  *igmph;
        struct iphdr    *ipiph;
        struct ipv6hdr  *ipv6h;
        unsigned char   *raw;
} h;

union {
        struct iphdr    *iph;
        struct ipv6hdr  *ipv6h;
        struct arphdr   *arph;
        unsigned char   *raw;
} nh;

union {
        unsigned char   *raw;
} mac;
```

Рис. 2.14. Структура полів h, nh, mac

– наступне поле має тип вказівника на структуру dst_entry, що описує маршрут пакету до його кінцевої точки;

– наступне поле має тип вказівника на структуру sec_path;

– наступне поле має тип масиву символів і також називається контрольним блоком SKB. Цей масив використовується певними протоколами і драйверами для зберігання приватних для поточного пакету даних. Наприклад, цей масив використвується протоколом TCP для зберігання порядкових номерів пакету і стану ретрансляції фрейму;

– наступні чотири поля мають тип unsigned int: len, data_len, mac_len, csum. Загальна кількість байтів у пакеті зберігається в len. SKB може складатися з лінійного буфера даних і набору з одного чи більше сторінок. Якщо є буфери сторінок, загальна кількість байт в області буфера сторінки встановлюється в змінній data_len. Тому кількість байтів у лінійному буфері може бути розрахована так: skb-> len - skb-> data_len. Поле mac_length містить довжину MAC заголовку. Csum містить контрольну суму пакету;

– sk_buff також містить чотири поля типу вказівника на char: head, data, tail, end. Спочатку поля вказують на місця в буфері, що містять власне дані пакетів, окрім поля end. Останнє поле вказує на структуру skb_shared_info, що починається одразу на наступний байт після ділянки пам’яті доступної для даних пакету.

Було опущено ще безліч полів структури sk_buff тому, що вони не використовуються в розробленому мережевому фільтрі і найчастіше використовується лише ядром Linux.

Після створення нового правила фільтрації мережевого трафіку, це правило відправляється до Python обгортки, в якій відбувається обробка кожного правила певним чином. Так, наприклад, у випадку блокування певного доменного імені в мережі Інтернет, необхідно дізнатися яка ip-адреса відповідає цьому доменному імені шляхом запиту до DNS-серверу. Після цього, запит повертає ip-адресу в

форматі Little-endian, тобто найменш значущий байт зберігається в найменшому адресі. Так як в мережевих протоколах порядок байтів визначений як Big-endian, то для подальшого використання ip-адреси необхідно перевести її в інший формат.

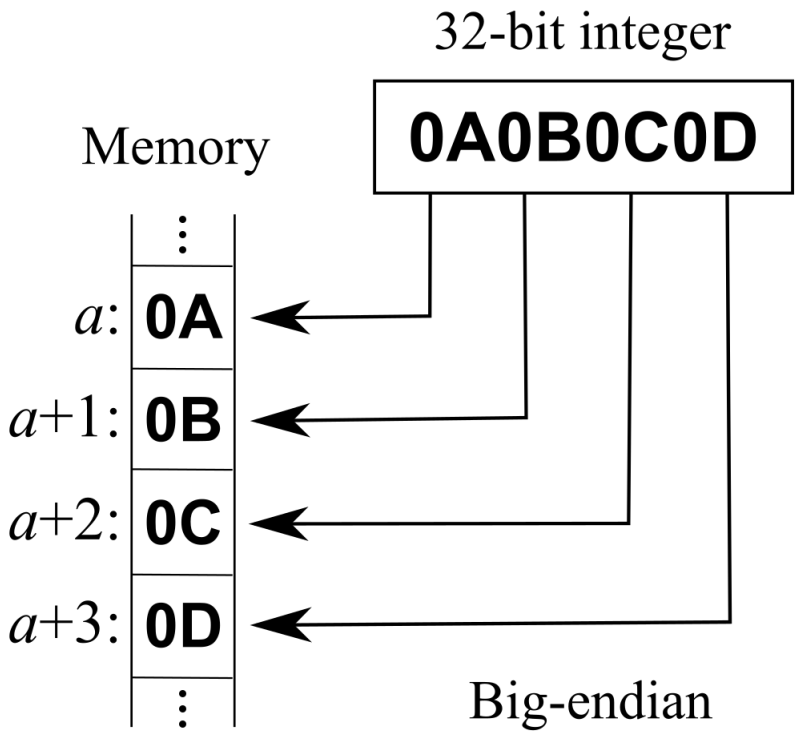

Рис. 2.15. Схематичне зображення формату Big-endian

При надходженні пакету до мережевого інтерфейсу wlxd03745972eba з мережі Інтернет обробка починається з XDP програми. Вказівник на структуру xdp_md передається у якості контексту до XDP програми. У даному випадку xdp_md буде інкапсулювати фрейм Ethernet. Використовуючи цю структуру, BPF програма має провести декапсуляцію фрейму до TCP сегменту, щоб отримати порт і ip-адресу пункту призначення, порівняти їх з комбінаціями порту і ip-адреси, що вказані адміністратором мережі як заборонені. Вказані адміністратором заборонені комбінації містяться в BPF таблицях, тому необхідно виконати пошук по ключу, і у разі вдалого пошуку відкинути вхідний пакет.

Декапсуляція в BPF програмах відбувається наступним чином. Лінійний буфер пакету xdp_md містить в собі послідовно розміщені заголовки фрейму Ethernet, IP пакету та TCP сегменту. Заголовок кожного протоколу у якості payload містить заголовок вищого рівня. Для того, щоб отримати з структури xdp_md потрібні дані, необхідно виконати наступний алгоритм:

1. Використовуючи вказівник data, що вказує на початок фрейму, з структурі xdp_md, привести його до типу ethhdr, що відповідає заголовку фрейму Ethernet.

2. Розрахувати розмір заголовку структури ethhdr, додати його до значення вказівника на початок фрейму Ethernet, і таким чином буде отримано цілочисельне значення для майбутнього вказівника на заголовок наступного протоколу.

3. Необхідно порівняти отримане значення зі значенням вказівника data_end з структури xdp_md, і у разі якщо офсет більше за значення data_end, то обробка пакету має бути зупинена шляхом відкидання, пропуску пакету або повернення помилки. Ця дія необхідна для того, щоб запевнити верифікатор BPF в тому, що програма не виходить за межі доступної для неї пам’яті. Якщо не виконати цю перевірку, верифікатор не зможе завантажити користувацьку програму у ядро і поверне код відповідної помилки.

4. Після вдалої перевірки можна привести отримане раніше цілочисельне значення до вказівника на структуру iphdr. Ця структура відповідає структурі загаловку протоколу IPv4.

5. Тепер для отримання доступу до полів структури iphdr необхідно перевірити знову, чи не виходять адреси цих полів за межі data_end. Для цього необхідно отримати розмір структури iphdr, додати його до значення вказівника на заголовок поточного протоколу, тобто iphdr, і порівняти з data_end. Знову, у разі невдалого порівняння необхідно повернути код дії XDP, і таким чином закінчити обробку пакету. У разі вдалого порівняння, можна отримати доступ до полі структури iphdr.

6. Якщо необхідно провести декапсуляцію до сегменту TCP, тоді потрібно розрахувати справжній розмір поточного заголовку протоколу IP за допомогою поля ihl (Internet Header Length). Ця необхідність зумовлена можливістю наявності IP options в заголовку. Поле ihl визначає загальну довжину заголовку IP у 32-бітних машинних словах. Мінімальне значення поля ihl дорівнює 5 і означає, що мінімальний розмір загаловку дорівнює 20 байтам. Таким чином, розрахунок справжнього розміру в байтах відбувається шляхом множення значення поля ihl на 4.

7. Далі необхідно знов перевірити чи не виходить поточний вказівник за межі data_end. Спочатку потрібно додати до вказівника на заголовок IP актуальне значення розміру цього заголовку і порівняти з data_end. У разі невдалого порівняння алгоритм дій аналогічний тим, що були на попередніх етапах.

8. У разі успіху створюється новий вказівник на заголовок TCP. Далі необхідно провести операції, аналогічні до розрахунку розміру заголовку IP, за виключенням того, що замість поля ihl використовується поле doff (Data Offset) в структурі tcph. І знову виконати порівняння отриманого значення зі значенням вказівника data_end.

9. Далі за адресою, отриманою шляхом додавання розрахованого значення у байтах поля data offset до вказівника на структуру tcph, розміщені дані протоколів прикладного рівня, наприклад HTTP.

Даний алгоритм закодовано за допомогою мови C, а тому для розрахунку розмірів структур заголовків зручно використовувати оператор sizeof. А для отримання відступу (offset) певного поля в середині структури можна використовувати оператор offsetof.

Аналогічний алгоритм необхідно виконувати для кожної програми BPF, якій у якості контексту передається вказівник на структуру xdp_md чи sk_buff.

2.6. Обгрунтування та організація вхідних та вихідних даних програми

Вхідними даними є правила фільтрації мережевого трафіку, які вводяться адміністратором в діалоговому режимі. Дані правила фільтрації і трасування мережевого трафіку також є вихідними даними розробленої системи. Також адміністратор в діалоговому режимі вводить дані користувачів локальної мережі, яким необхідно обмежити доступ до мережі Інтернет. Правила фільтрації мають наступні атрибути: ip-адреса хоста в локальній мережі, ip-адреса хоста в мережі Інтернет, порт та протокол. Журнальні записи трасування локальної мережі мають наступні атрибути: ip-адреса кінцевого призначення, ip-адреса відправника, порт, протокол, час створення журнального запису.

Дані, якими обмінюються веб-додаток і міжмережевий екран мають формат JSON. Дані, якими обмінюються кінцевий користувач з веб-додатком мають формат form-data.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Для роботи програмного продукту використовуються наступні апаратні засоби:

- одноплатний комп’ютер Raspberry Pi 4 Model B 4 GB для розгорнення мережевого фільтру та веб-додатку;
- Wi-Fi адаптер TL-WN722N.

Для доступу до розробленого веб-додатку необхідно мати будь-який пристрій з доступом до локальної мережі, що має встановлений веб-браузер.

2.7.2. Використані програмні засоби

Під час розробки програмного продукту були використані наступні програмні засоби:

– JetBrains Intellij Idea Ultimate Edition;
– JetBrains PyCharm;
– Git, Github;
– PgAdmin 4;
– мережевий драйвер rtl8188eu.

Intellij Idea - це інтегроване середовище розробки для Java. Надає можливості автодоповнення коду, статичного аналізу коду, навігації серед проектів, рефакторінгу, використання вбудованих плагінів і інструментів (наприклад, Maven), полегшує розробку з використанням популярних фреймворків (наприклад, Spring).

Ultimate версія IDE забезпечує деякими додатковими можливостям, наприклад, підтримка JavaScript.

PyCharm - це інтегроване середовище розробки на Python. Загалом, має аналогічний функціонал до Intellij Idea, але для Python і його фреймворків.

Git - це розподілена система контролю версій.

GitHub – це платформа для розміщення Git репозиторіїв, які можуть складатися з початкового коду.

Мережевий драйвер rtl8188eu необхідний для сумісності і правильного функціонування Wi-Fi Hot Spot в Ubuntu 20.04 .

2.7.3. Виклик та завантаження програми

Для того щоб розпочати роботу з міжмережевим фільтром необхідно виконати git clone відповідних репозиторії, або встановити файли проектів з диску, послідовно виконати команди “python3 main.py” і “java -jar webapp.jar”.

Для початку роботи з веб-додатком кінцевому користувачу необхідно за допомогою будь-якого веб-браузера перейти по ip-адресі, що відповідає комп’ютеру, на якому розгорнутий даний веб-додаток і який знаходиться в локальній мережі.

2.7.4. Опис інтерфейсу користувача

Для доступу до розробленого веб-додатку, за допомогою якого відбувається керування, створення та видалення правил фільтрації мережевого трафіку, перегляд журналу трасування пакетів локальної мережі, необхідно авторизуватися в системі на сторінці, зображеній на рис. 2.16.

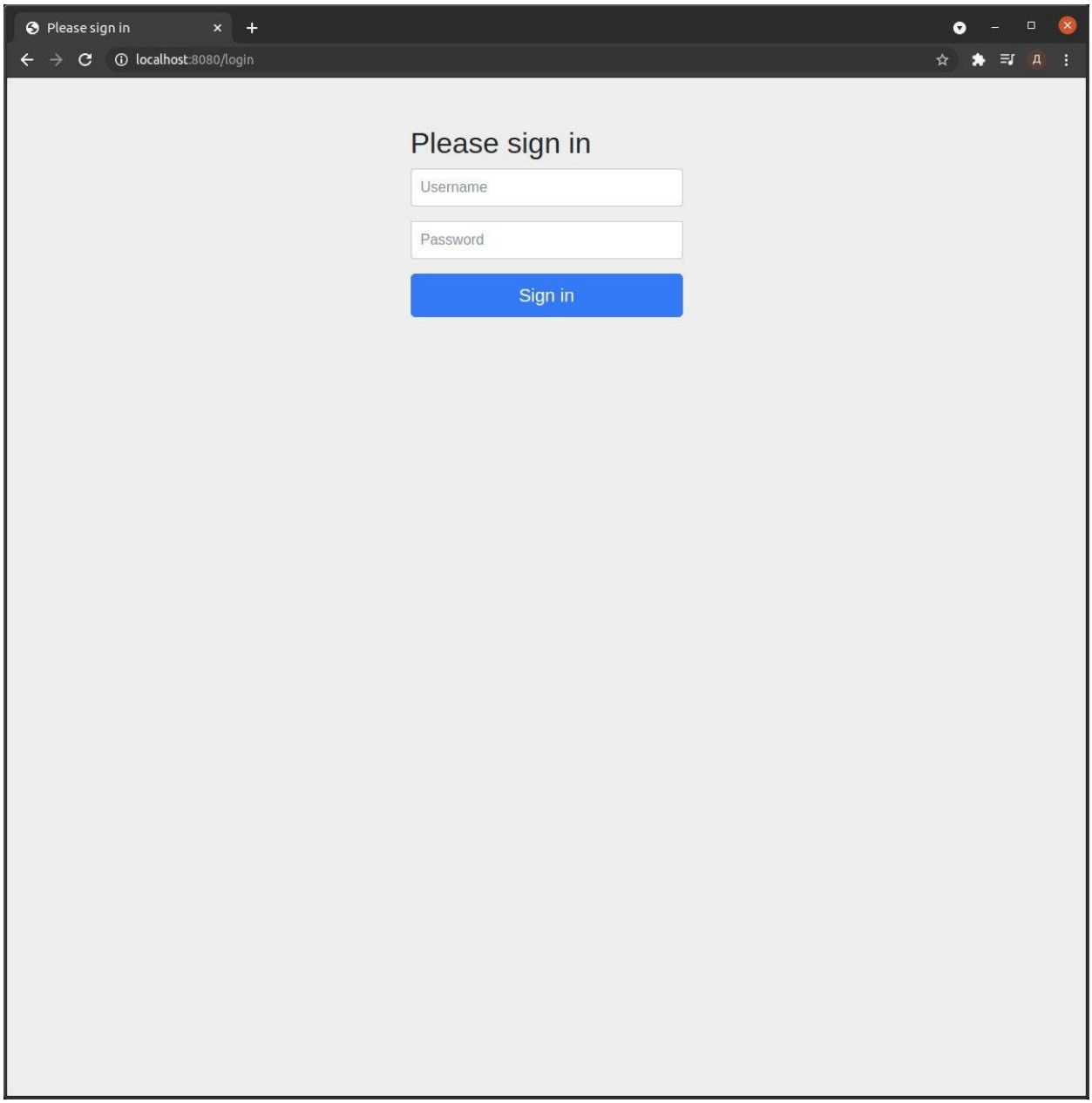

Рис. 2.16. Сторінка авторизації

При невдалій спробі авторизації, якщо неправильний логін чи пароль, відбувається редірект користувача на відповідну сторінку, на якій міститься відповідне повідомлення про помилку.

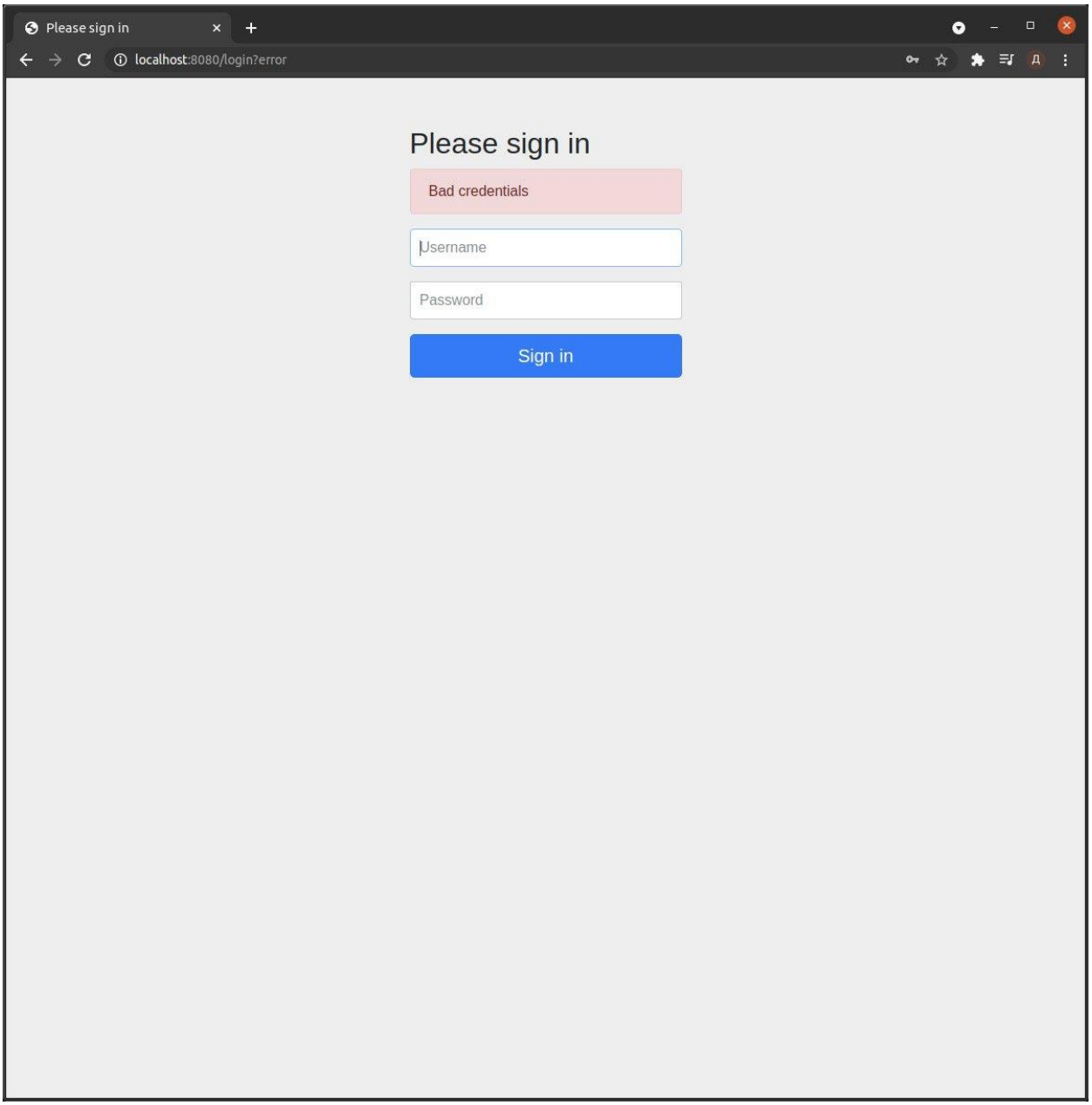

Рис. 2.17. Сторінка авторизації після невдалої спроби авторизації

Після вдалої авторизації користувач має можливість вийти з поточного акаунту, наприклад, з метою авторизації в інший. Але попередньо користувач побачить повідомлення, в якому у нього буде запрошено підтвердити дію.

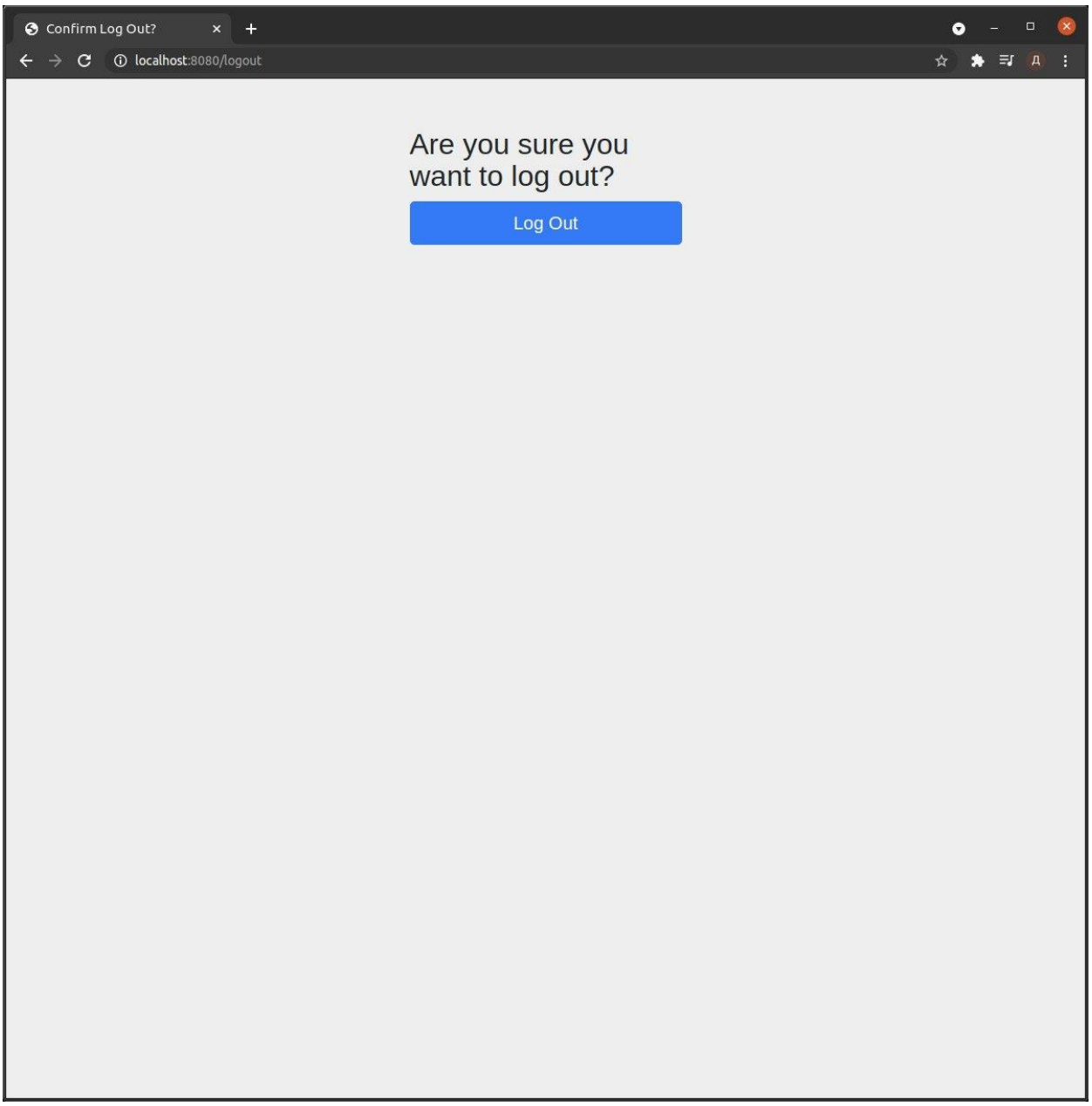

Рис. 2.18. Повідомлення для підтвердження дії

Після виходу с поточного акаунту, користувача знову буде перенаправлено на сторінку авторизації з повідомленням про вдалий вихід.

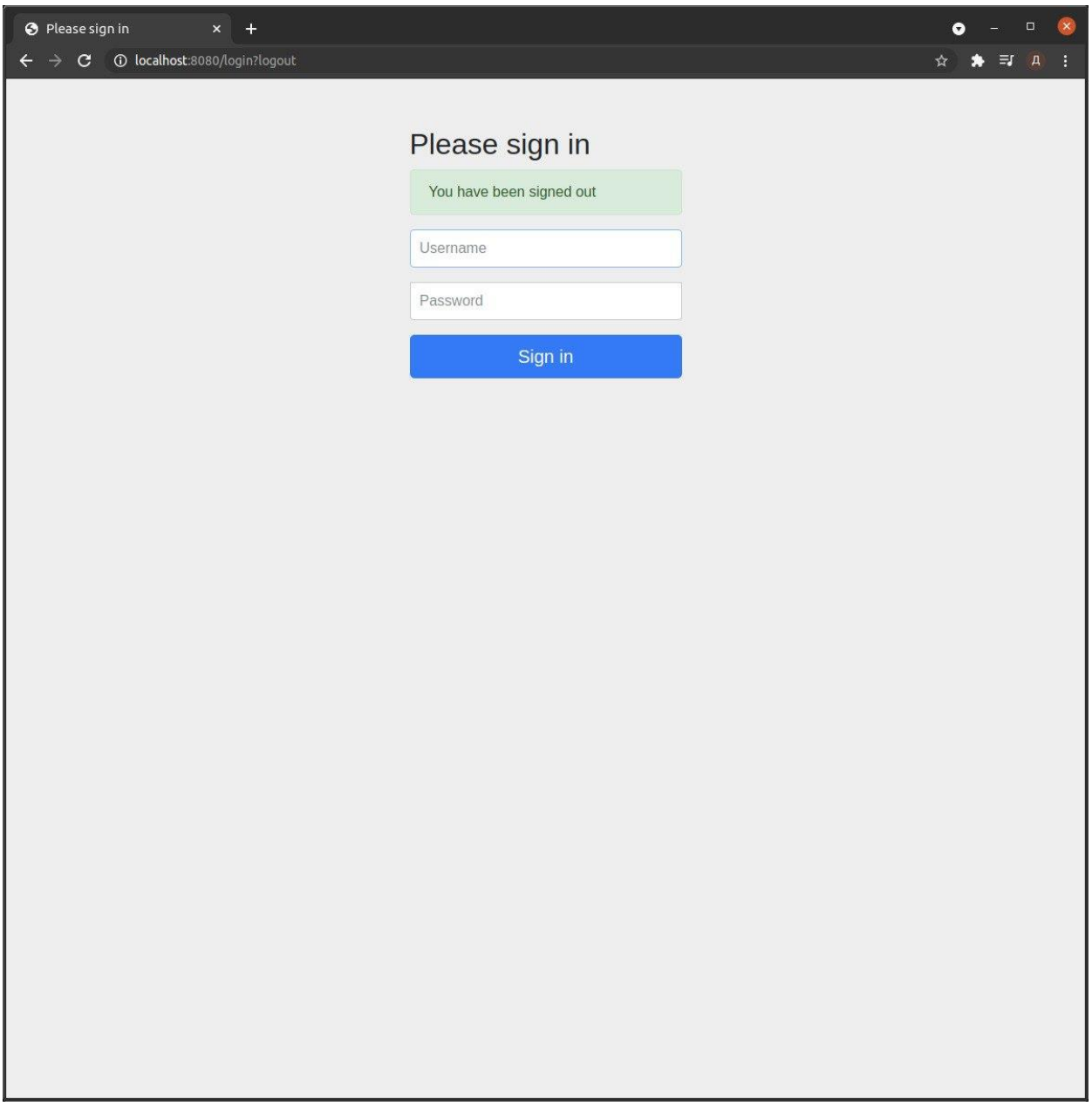

Рис. 2.19. Сторінка авторизації після виходу з акаунту

На кожній сторінці, окрім сторінок авторизації, користувач має панель з посилань для доступу до існуючих функціональностей: перегляд журналу трасування, створення нових правил, перегляд списку заблокованих користувачів локальної мережі і список зареєстрованих акаунтів в програмній системі.

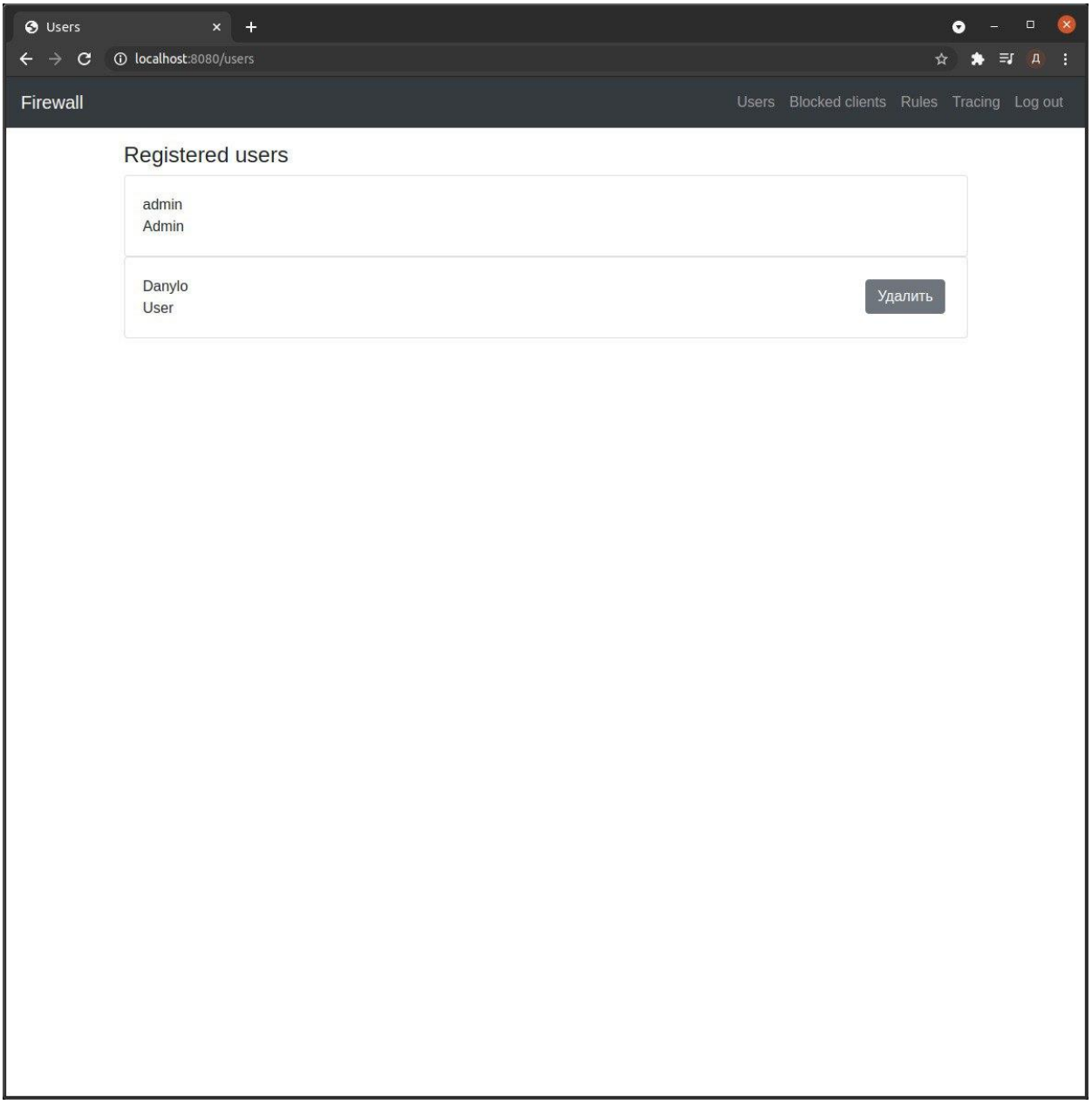

Рис. 2.20. Список зареєстрованих користувачів

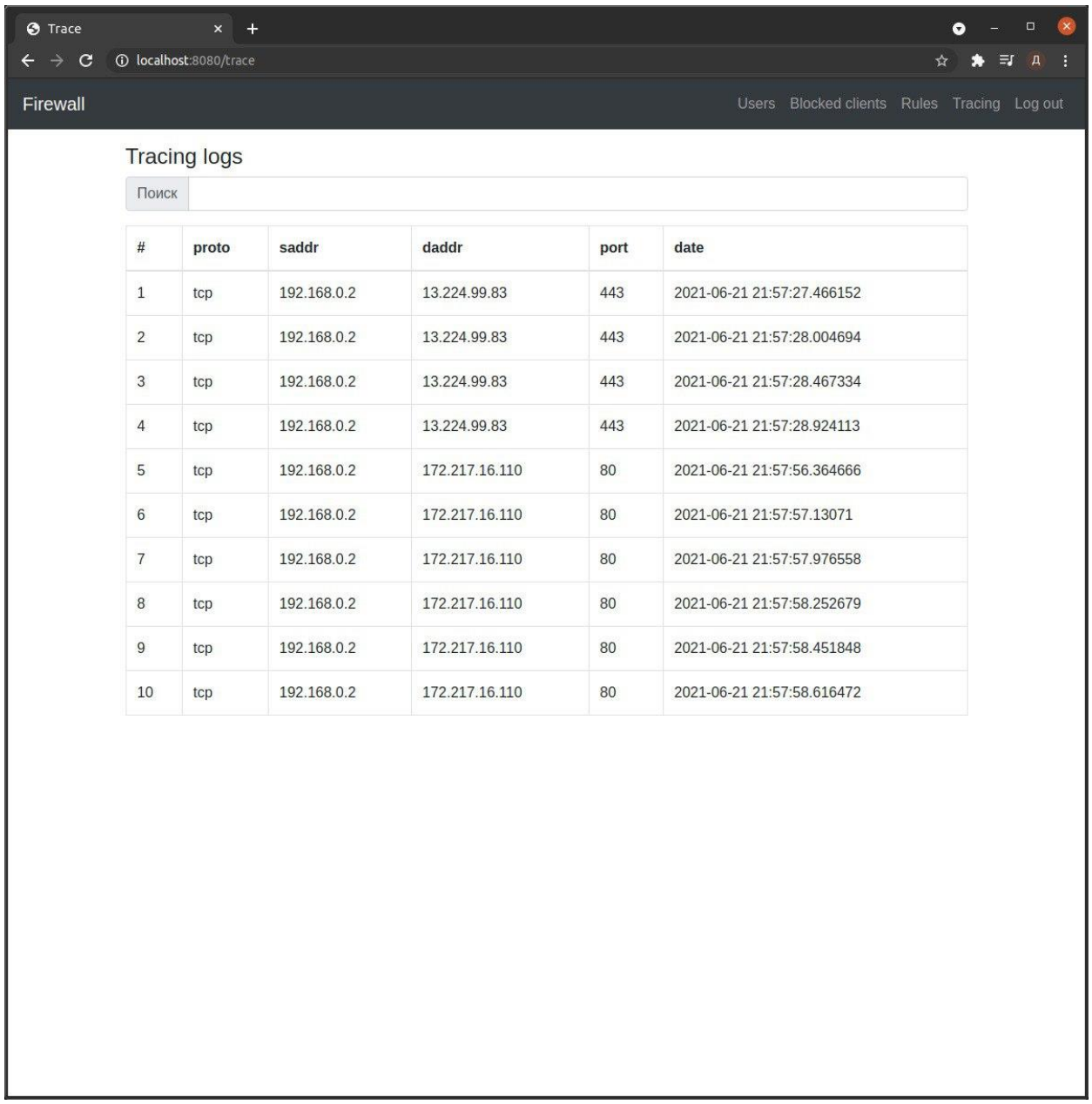

Рис. 2.21. Журнал моніторингу мережі

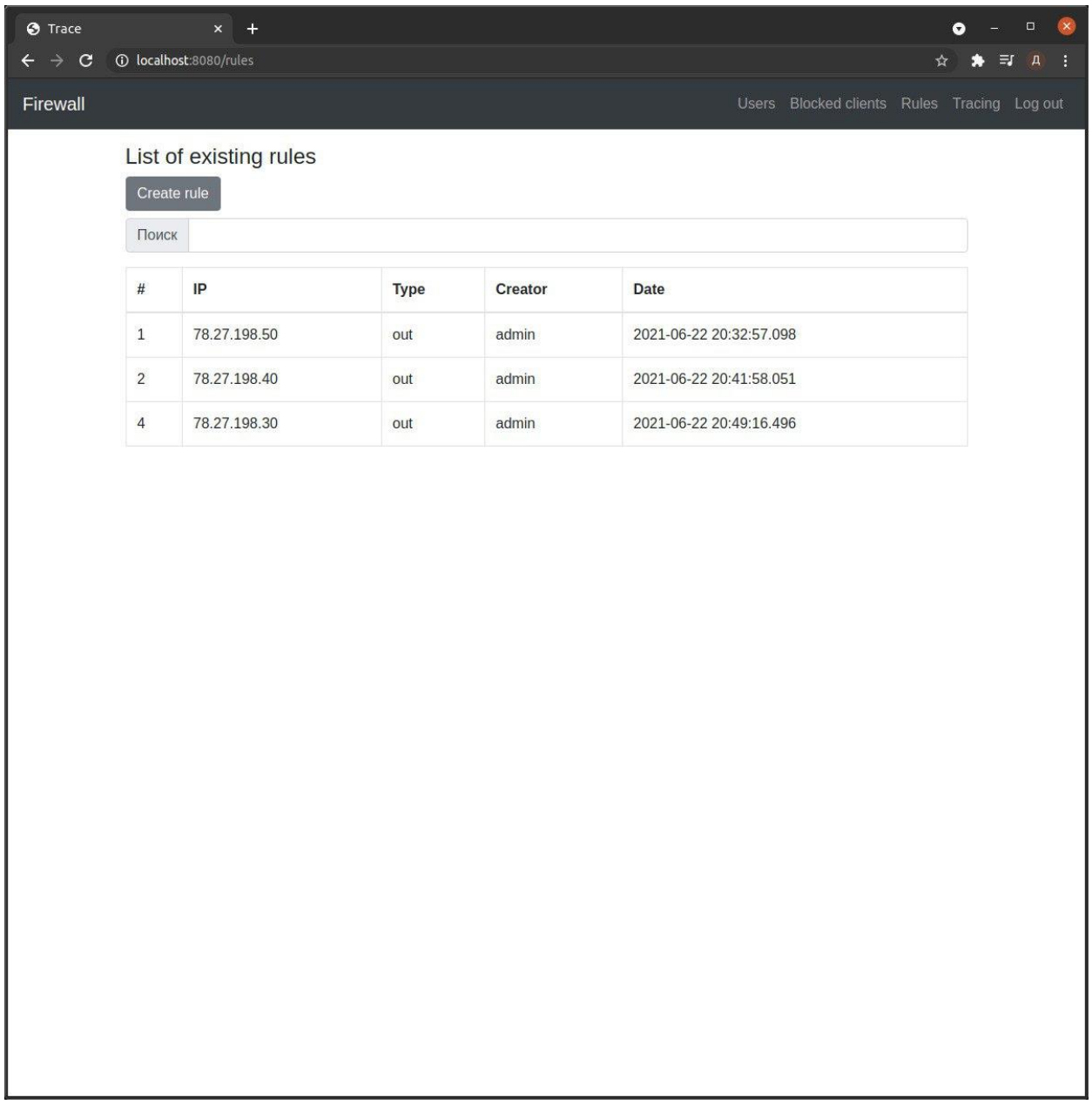

Рис. 2.22. Список існуючих правил фільтрації

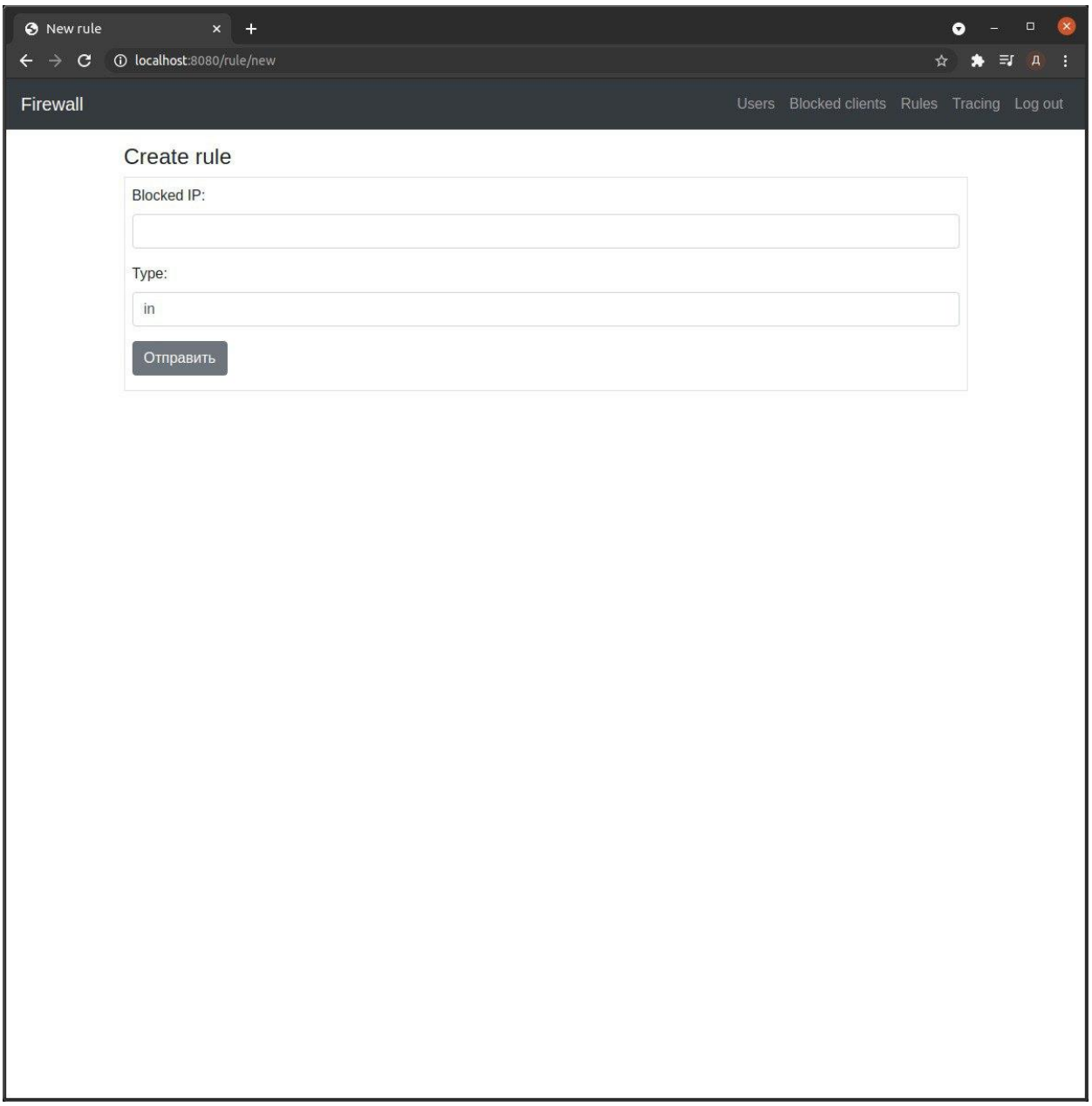

Рис. 2.23. Форма створення правила

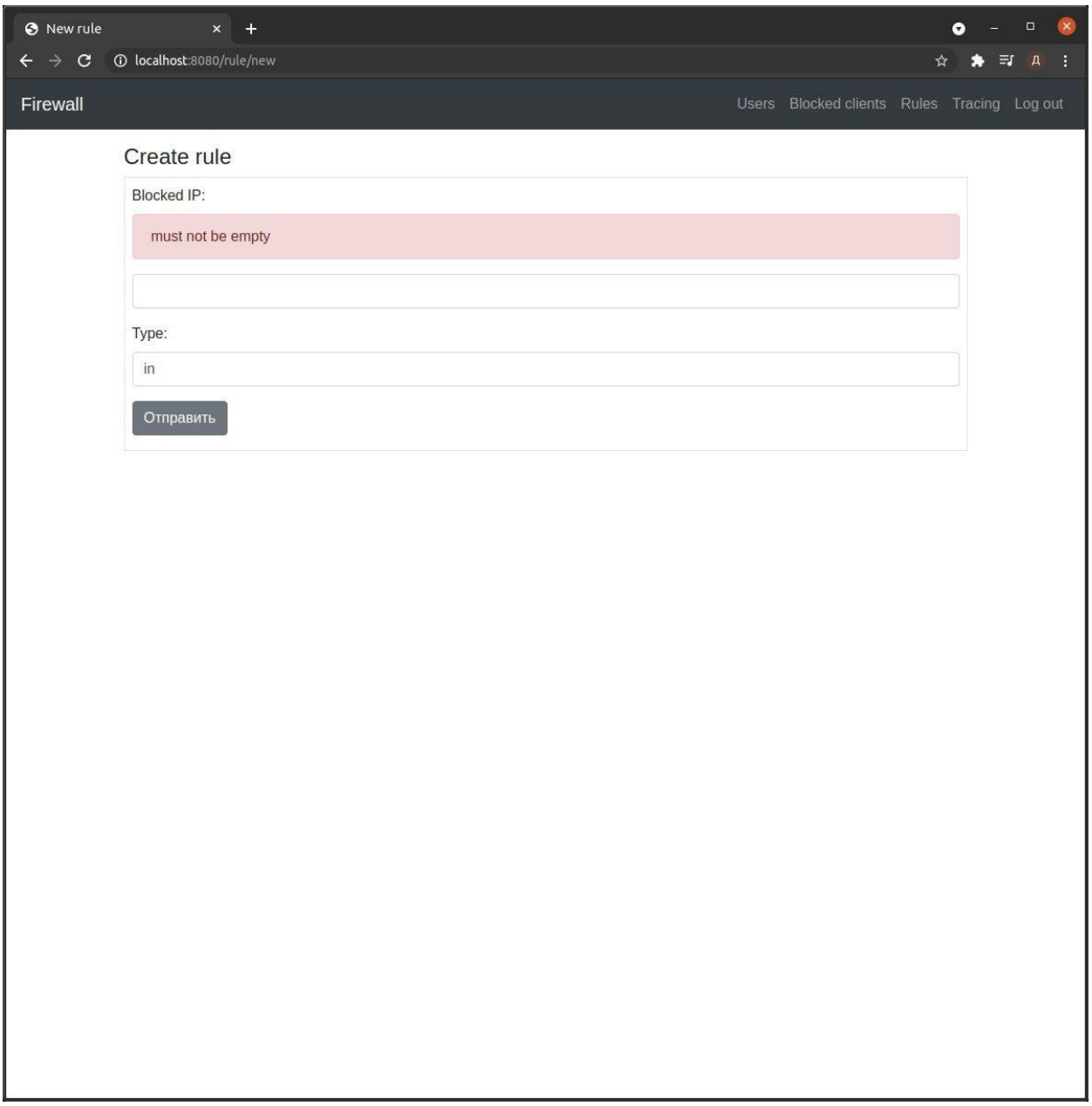

Рис. 2.24. Форма створення правила після некоректного введення даних

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1580;

2. коефіцієнт складності програми – 1,25;

3. коефіцієнт корекції програми в ході її розробки – 0,07;

4. годинна заробітна плата програміста – 139 грн/год;

Середня годинна зарплата програміста була вирахувана виходячи з даних порталу DOU.UA. Середньоукраїнська заробітна плата junior-програміста, складає близько 900 доларів у місяць. При курсі валют НБУ на початок червня 2021 року один американський долар дорівнює 27,1 грн, тому середня зарплата в гривнях дорівнює 24390 грн. При восьмигодинному робочому дні (176 годин в місяць в середньому) середня зарплата за годину буде становити 139 грн.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;

7. вартість машино-години ЕОМ – 16.5 грн/год.

Звичайний комп'ютер споживає 200 Ватт / год, що дорівнює 0,2 кВт / год. Вартість одного кВт на годину від постачальника Yasno на стан середини 2021 року дорівнює 1,68 грн в незалежності від обсягом споживання. Отже вартість електроенергії споживаної комп'ютером за годину дорівнює 0,336 грн. Ціна довгострокової оренди комп'ютера дорівнює 3200 грн на місяць, з чого можна вирахувати, що ціна оренди на годину буде дорівнювати 18,2 грн. Остаточна вартість машино-години ЕОМ дорівнює 18,53 грн за годину.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1+p), \text{ де} \qquad (3.2)$$

Q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1580 \cdot 1{,}25 \cdot (1+0{,}07) = 2113{,}25;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин,} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{2113{,}25 \cdot 1{,}25}{85 \cdot 0{,}8} = 38{,}85, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20\ldots25)\cdot K}; \quad (3.4)$$

$$t_a = \frac{2113,25}{20\cdot 0,8} = 132,08, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot K}; \quad (3.5)$$

$$t_n = \frac{2113,25}{25\cdot 0,8} = 105,66, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4\ldots5)\cdot K}; \quad (3.6)$$

$$t_n = \frac{2113,25}{5\cdot 0,8} = 528,31, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t^к_{отл} = 1,5\cdot t_{отл}; \quad (3.7)$$

$$t^к_{отл} = 1,5\cdot 528,31 = 792,465, \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_д = \frac{Q}{(15\ldots20)\cdot K}; \quad (3.9)$$

$$t_д = \frac{2113,25}{20\cdot 0,8} = 132,06 \text{ людино-годин.}$$

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75\cdot t_{др}; \quad (3.10)$$

$$t_{до} = 0,75\cdot 132,06 = 99,05 \text{ людино-годин.}$$

$$t_д = 77,5192 + 103,359 = 231,1, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 132,08 + 38,85 + 105,66 + 792,465 + 231,1 = 1350,155, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1350,155 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ включають витрати на заробітну плату виконавця програми з/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 1350{,}155 \cdot 139 = 187671{,}545 \text{ грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 792{,}465 \cdot 18{,}53 = 14682{,}376 \text{ грн.}$$

$$К_{ПО} = 187671{,}545 + 14682{,}376 = 202355{,}92 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k - число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$Т = \frac{1350{,}155}{1 \cdot 176} = 7{,}671 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 1350,155 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 7,671 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 202355,92 грн.

ВИСНОВКИ

В даній кваліфікаційній роботі була розроблена програмна система завдяки якій можна проводити фільтрацію та моніторинг мережевого трафіку.

Це програмне забезпечення призначене для використання малим та середнім бізнесом завдяки тому, що було досягнуто зменшення системних вимог разом і з спрощенням у використанні звичайним користувачем. Зменшення системних вимог було реалізовано за рахунок використання новіших технологій на базі операційної системи з відкритим початковим кодом на базі ядра Linux.

Практичне значення полягає у захисті локальних мереж від несанкціонованого доступу, небажаного трафіку і моніторингу мережі.

Під час виконання даної кваліфікаційної роботи було виконано наступні задачі:

− проаналізовано предметну галузь задачі;

− проведено порівняння і аналіз існуючих аналогів;

− спроектовано комплексну програмну систему;

− написано програмний код продукту;

− розроблено рекомендації щодо використання продукту.

Програма створена на базі мови програмування Java з фреймворками Spring, мови Python з використанням бібліотеки BCC, мови C і новітнбох технології extended BPF в ядрі Linux.

Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту 1350,155 людино-годин, проведений підрахунок вартості роботи по створенню програми 202355,92 грн. та розраховано час на його створення 7,671 міс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мартін Калін. Java Web Services: Up and Running, Харків : Наука і техніка, 2009. 320 с.
2. David Calavera and Lorenzo Fontana. Linux Observability with BPF: Advanced Programming for Performance Analysis and Networking. 2019. – 201 с.
3. Michael Kerrisk. The Linux Programming Interface. 2010. – 1512 с.
4. Крейг Уолс. Спрінг у Дії, Шосте Видання, Київ : Видавництво Діалектика, 2021. 520 с
5. David Flanagan. JavaScript: The Definitive Guide Seventh Edition, 2021 – 722 с.
6. Brian Kernighan and Dennis Ritchie. The C Programming Language. 2nd Edition. 1978. 343 с.
7. Кішорі Шаран. Особливості мови програмування Java, . Київ : Видавництво Діалектика, 2018 р. 236 с.
8. Кей С. Хорстманн. Java. Бібліоетка професіонала, том 1. Видавництво Діалектика, 2020 р. 864 с.
9. Кей С. Хорстманн. Java. Бібліоетка професіонала, том 2. Розширені засоби програмування. Видавництво Діалектика, 2021 р. 843 с.
10. JSON Web Token Introductio. URL: https://jwt.io/introduction
11. Веб-сайт НБУ, URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD
12. Веб-сайт «Українська спільнота програмістів», URL: https://jobs.dou.ua/salaries/#period=dec2020&city=all&title=Junior%20Software%20Engineer&language=Java&spec=&exp1=0&exp2=2
13. Веб-сайт компанії орендатора комп'ютерів «ChipChip», URL: https://komputers.com.ua/arenda-kompyutera/
14. PostgreSQL Query Optimization The Ultimate Guide to Building Efficient Queries, 2021 – 310 с.

15. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith, 2019 – 284 с.

16. Євген Моргунов. PostgreSQL. Основи мови SQL. Навчальний посібник, 2019 – 336 с.

17. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 6.050101 «Комп'ютерні науки / І.М. Удовик, Л.М. Коротенко, О.С. Шевцова. Нац. гірн. ун-т. – Д : НТУ «Дніпровська політехніка». - 2018. – 65 с.

18. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності «Комп'ютерні системи» / О.Г. Вагонова, О.Б. Нікітіна, Н.Н. Романюк; М-во освіти і науки України, ДВНЗ «Нац. гірн. ун- т». – Д.: НГУ, 2013. – 11 с.

19. Ірина Бородкіна, Георгій Бородкін. Інженерія програмного забезпечення. Посібник для студентів вищих навчальних закладів, 2018 – 204 с.

20. Юрій Грицюк. Аналіз вимог до програмного забезпечення, 2018 – 456 с.

ДОДАТОК А

КОД ПРОГРАМИ

FirewallApplication.java

```
package com.zink.firewall;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FirewallApplication {

   public static void main(String[] args) {
      SpringApplication.run(FirewallApplication.class, args);
   }

}
```

WebSocketConfig.java

```
package com.zink.firewall.web;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;

@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

   @Override
   public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
      registry.addHandler(getWsHandler(), "/ws");
   }

   @Bean
   public WebSocketHandler getWsHandler() {
      return new WebSocketHandler();
   }
}
```

WebSocketHandler.java

```
package com.zink.firewall.web;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.zink.firewall.dto.TraceDto;
import com.zink.firewall.services.LogService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.socket.TextMessage;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.TextWebSocketHandler;

import java.util.List;

public class WebSocketHandler extends TextWebSocketHandler {

   @Autowired
   private LogService logService;

   @Override
   protected void handleTextMessage(WebSocketSession session, TextMessage message) {
      String json = message.getPayload();
      ObjectMapper objectMapper = new ObjectMapper();
      try {
         List<TraceDto> traceDtos = objectMapper.readValue(json, new
TypeReference<List<TraceDto>>() {
         });
         logService.saveTraces(traceDtos);
         System.out.println(traceDtos);
      } catch (JsonProcessingException e) {
         throw new IllegalStateException("json erorr");
      }
   }
}
```

MainController.java

```
package com.zink.firewall.web;

import com.zink.firewall.dto.RuleDto;
import com.zink.firewall.services.LogService;
import com.zink.firewall.services.RuleService;
import com.zink.firewall.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class MainController {

    @Autowired
    private UserService userService;

    @Autowired
    private LogService logService;

    @Autowired
    private RuleService ruleService;

    @GetMapping("/")
    public String home(Model model) {
        return "index";
    }

    @GetMapping("/users")
    public String users(Model model) {
        model.addAttribute("users", userService.getAllUsers());
        return "users";
    }

    @GetMapping("/trace")
    public String trace(Model model) {
        model.addAttribute("traces", logService.getAllTrace());
        return "trace";
    }

    @GetMapping("/rule/new")
    public String createRule(Model model) {
        return "create_rule";
    }

    @PostMapping("rule/new")
    public String createRule(Authentication auth, @Validated @ModelAttribute("rule")
RuleDto rule, BindingResult br, Model model) {
        if (br.hasErrors()) {
            return "create_rule";
        }
```

```
        rule.setCreator(auth.getName());
        ruleService.createRule(rule);
        return "redirect:/rules";
    }

    @GetMapping("/rules")
    public String rules(Model model) {
        model.addAttribute("rules", ruleService.getAllRules());
        return "rules";
    }
}
```

SecurityConfig.java

```
package com.zink.firewall.web;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurer
Adapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/**").hasAnyRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .formLogin();
    }

    @Bean
    public BCryptPasswordEncoder encoder() {
        return new BCryptPasswordEncoder(12);
    }

}
```

LogService.java

```
package com.zink.firewall.web;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurer
Adapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
            .authorizeRequests()
            .antMatchers("/**").hasAnyRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .formLogin();
   }

   @Bean
   public BCryptPasswordEncoder encoder() {
      return new BCryptPasswordEncoder(12);
   }

}
```

RuleService.java

```
package com.zink.firewall.services;

import com.zink.firewall.dao.entities.Rule;
import com.zink.firewall.dao.enums.Type;
import com.zink.firewall.dao.repos.RuleRepository;
import com.zink.firewall.dto.RuleDto;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

@Service
```

```
public class RuleService {

   @Autowired
   private UserService userService;

   @Autowired
   private RuleRepository ruleRepository;

   public void createRule(RuleDto ruleDto) {
      Rule rule = new Rule();
      rule.setIp(ruleDto.getIp());
      rule.setDate(new Date());
      rule.setType(Type.from(ruleDto.getType()));
      rule.setCreator(userService.getUserByName(ruleDto.getCreator()));
      ruleRepository.save(rule);
   }

   public List<RuleDto> getAllRules() {
      return
ruleRepository.findAll().stream().map(RuleDto::new).collect(Collectors.toList());
   }

}
```

UserService.java

```
package com.zink.firewall.services;

import com.zink.firewall.dao.entities.User;
import com.zink.firewall.dao.repos.UserRepository;
import com.zink.firewall.dto.UserDto;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class UserService {

   @Autowired
   private UserRepository userRepository;

   public List<UserDto> getAllUsers() {
      List<User> users = userRepository.findAll();
      return users.stream().map(u -> {
         UserDto dto = new UserDto();
         dto.setId(u.getId());
```

```
            dto.setRole(u.getRole());
            dto.setUsername(u.getUsername());
            return dto;
        }).collect(Collectors.toList());
    }

    public User getUserByName(String username) {
        return userRepository.findByUsername(username).get();
    }

}
```

MyUserDetailsService.java

```
package com.zink.firewall.services;

import com.zink.firewall.dao.entities.User;
import com.zink.firewall.dao.repos.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collection;
import java.util.Collections;

@Service("userDetailsService")
public class MyUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User currentUser = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("Нет пользователя с
таким именем: " + username));
        Collection<GrantedAuthority> authorityList = Collections
                .singletonList(new
SimpleGrantedAuthority(currentUser.getRole().getRoleName()));
        return                                                                    new
org.springframework.security.core.userdetails.User(currentUser.getUsername(),
                currentUser.getPassword(), authorityList);
```

```
    }

}
```

User.java

```
package com.zink.firewall.dao.entities;

import com.sun.istack.NotNull;
import com.zink.firewall.dao.enums.Role;
import javax.persistence.*;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @Column(nullable = false, unique = true, length = 12)
    private String username;

    @NotNull
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(length = 10)
    private Role role;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String name) {
        this.username = name;
    }

    public String getPassword() {
```

```
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Role getRole() {
        return role;
    }

    public void setRole(Role role) {
        this.role = role;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + username + '\'' +
                ", password='" + password + '\'' +
                ", role=" + role +
                '}';
    }
}
```

Rule.java

```
package com.zink.firewall.dao.entities;

import com.zink.firewall.dao.enums.Type;

import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name = "rules")
public class Rule {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 15)
    private String ip;
    private Date date;
    @Enumerated(EnumType.STRING)
```

```
    @Column(length = 3)
    private Type type;
    @ManyToOne
    private User creator;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public Type getType() {
        return type;
    }

    public void setType(Type type) {
        this.type = type;
    }

    public User getCreator() {
        return creator;
    }

    public void setCreator(User creator) {
        this.creator = creator;
    }
}
```

Trace.java

```
package com.zink.firewall.dao.entities;

import javax.persistence.*;
import java.util.Date;


@Entity
@Table(name = "traces")
public class Trace {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String saddr;
   private String daddr;
   private String dport;
   private String proto;
   private Date date;

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public String getSaddr() {
      return saddr;
   }

   public void setSaddr(String saddr) {
      this.saddr = saddr;
   }

   public String getDaddr() {
      return daddr;
   }

   public void setDaddr(String daddr) {
      this.daddr = daddr;
   }

   public String getDport() {
      return dport;
```

```
    }

    public void setDport(String dport) {
        this.dport = dport;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getProto() {
        return proto;
    }

    public void setProto(String proto) {
        this.proto = proto;
    }

    @Override
    public String toString() {
        return "Trace{" +
                "id=" + id +
                ", proto='" + proto + '\'' +
                ", saddr='" + saddr + '\'' +
                ", daddr='" + daddr + '\'' +
                ", dport='" + dport + '\'' +
                ", date=" + date +
                '}';
    }
}
```

redirection.c

```
#define KBUILD_MODNAME "program"
#include <linux/bpf.h>
#include <linux/if_ether.h>
#include <linux/in.h>
#include <linux/ip.h>
#include <linux/pkt_cls.h>
#include <linux/tcp.h>

BPF_ARRAY(iplist, uint32_t, 256);
```

```
#if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
#define __bpf_htons(x) __builtin_bswap16(x)
#define __bpf_constant_htons(x) ___constant_swab16(x)
#elif __BYTE_ORDER__ == __ORDER_BIG_ENDIAN__
#define __bpf_htons(x) (x)
#define __bpf_constant_htons(x) (x)
#else
#error "Fix compiler's __BYTE_ORDER__"
#endif

#define bpf_htons(x) \
  (__builtin_constant_p(x) ? __bpf_constant_htons(x) : __bpf_htons(x))

#define IP_CSUM_OFF offsetof(struct iphdr, check)



static inline void modify_http_body(struct __sk_buff *skb);

static inline bool adjust_skb(struct __sk_buff *skb);



int redirection(struct __sk_buff *skb) {
    void *data = (void*)(long)skb->data;
    void *data_end = (void*)(long)skb->data_end;

    struct ethhdr *ethh = data;
    struct iphdr *iph;

    __be32 daddr;
    uint32_t *blocked_addr;

    __u64 ihl = sizeof(*iph);
    __u64 ip_offset = sizeof(*ethh);

    if (data + ip_offset + ihl > data_end) {
        return TC_ACT_SHOT;
    }

    // может ли быть не IP?
    iph = ip_offset + (void*) (long) ethh;
    daddr = iph->daddr;

    bpf_trace_printk("daddr: %d\n", daddr);

    for (int i = 0; i < 256; i++) {
        int index = i;
```

```
        blocked_addr = iplist.lookup(&index);
        if (blocked_addr) {
            if (daddr == *blocked_addr) {
                if (adjust_skb(skb)) {
                    modify_http_body(skb);
                    return TC_ACT_OK;
                }
            }
        } else {
            break;
        }
    }

    return TC_ACT_OK;
}

static inline bool adjust_skb(struct __sk_buff *skb) {
    void *data = (void*)(long) skb->data;
    void *data_end = (void*)(long) skb->data_end;

    struct ethhdr *ethh = data;
    struct iphdr *iph;

    __u64 ip_offset = sizeof(*ethh);

    if (data + ip_offset + sizeof(*iph) > data_end) {
        return false;
    }

    iph = data + ip_offset;


    if (iph->protocol == IPPROTO_TCP) {
        return false;
    }

    __u32 ip_hlen = iph->ihl << 2;
    struct tcphdr *tcph = (void*)(long) iph + ip_hlen;
    if ( (void*)(long) tcph + sizeof(*tcph) > data_end) {
        return false;
    }

    __u32 tcp_hlen = tcph->doff << 2;
    __u32 poffset = ETH_HLEN + ip_hlen + tcp_hlen;
    __u32 plength = iph->tot_len - ip_hlen - tcp_hlen;


    unsigned long r[7];
```

```
    if (plength >= 7) {
        int i = 0;
        for (int i = 0; i < 7; i++) {
            r[i] = load_byte(skb, poffset + i);
        }
    }

    bool isMatch = false;

    if ((r[0] == 'G') && (r[1] == 'E') && (r[2] == 'T')) {
        isMatch = true;
    }
    if ((r[0] == 'P') && (r[1] == 'O') && (r[2] == 'S') && (r[3] == 'T')) {
        isMatch = true;
    }
    if ((r[0] == 'P') && (r[1] == 'U') && (r[2] == 'T')) {
        isMatch = true;
    }
    if ((r[0] == 'D') && (r[1] == 'E') && (r[2] == 'L') && (r[3] == 'E') && (r[4] == 'T') &&
(r[5] == 'E')) {
        isMatch = true;
    }
    if ((r[0] == 'H') && (r[1] == 'E') && (r[2] == 'A') && (r[3] == 'D')) {
        isMatch = true;
    }
    if ((r[0] == 'H') && (r[1] == 'T') && (r[2] == 'T') && (r[3] == 'P')) {
        isMatch = true;
    }

    if (isMatch) {
        __u16 http_substitution_length = 66;
        __u32 pdiff = http_substitution_length - plength;
        return 0 == bpf_skb_change_tail(skb, pdiff, 0);
    }

    bpf_trace_printk("not http");
    return false;
}

static inline void modify_http_body(struct __sk_buff *skb) {
    void *data = (void*)(long)skb->data;
    void *data_end = (void*)(long)skb->data_end;
    struct ethhdr *ethh = data;
    struct iphdr *iph;

    __u64 ip_offset = sizeof(*ethh);
    if (data + ip_offset + sizeof(*iph) > data_end) {
        return;
```

```
    }

    iph = (void*) ethh + sizeof(*ethh);
    __u32 ip_hlen = iph->ihl << 2;
    struct tcphdr *tcph = (void*)(long) iph + ip_hlen;

    if ((void*) iph + sizeof(*tcph) > data_end) {
        bpf_trace_printk("modify_http_body 2nd check failed\n");
        return;
    }

    __u16 http_substitution_length = 66;
    __u32 tcp_hlen = tcph->doff << 2;
    __u32 poffset = ETH_HLEN + ip_hlen + tcp_hlen;
    __u32 plength = iph->tot_len - ip_hlen - tcp_hlen;

    __u32 daddr_offset = (void*)(long) iph - data + offsetof(struct iphdr, daddr);
    __u32 new_daddr = 16787978; // 10.42.0.1
    __u32 old_daddr = iph->daddr;
    iph->daddr = new_daddr;

    bpf_l3_csum_replace(skb, IP_CSUM_OFF, old_daddr, new_daddr, 4);

    char http_substitution[] =
        "GET /forbidden HTTP/1.1\r\nHost: 10.42.0.1\r\nAccept: */*\r\n\r\n";
    bpf_skb_store_bytes(skb, poffset, http_substitution, http_substitution_length, 0);
}
```

ДОДАТОК Б

Відгук керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему: «Розробка програмного забезпечення для фільтрації та трасування трафіку з використанням технології BPF в Linux»
студента групи 121-17-1 Зіньковського Данила Олександровича

Керівник економічного розділу **О.Г. Вагонова**

Зав. каф. ПЕП та ПУ, д.е.н.

ДОДАТОК В

Перелік файлів на диску

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Зіньковський.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Зіньковський.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Зіньковський.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Зіньковський.ppt	Презентація кваліфікаційної роботи