МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

**САГАН МАКСИМ ЯРОСЛАВОВИЧ**

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
д. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2022р.

# РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ЗАХИСТУ ВІД DDOS АТАК НА ОСНОВІ ЛОГІВ NGINX

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Антонов Ю. С., доцент кафедри
інформаційних технологій
к. ф-м. н., доцент
_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2022

Зміст

# АННОТАЦІЯ

**Саган М.** Розробка інтелектуальної системи захисту від DDoS атак на основі логів nginx. Спеціальність 122 “Комп'ютерні науки”, Освітня програма “Data science”. Донецький національний університет імені Василя Стуса. 2022

У кваліфікаційній (магістерській) роботі розроблено додаток для захисту веб-серверу на основі логів, які створюються веб-сервером, за час роботи, та частотним аналізом входження користувача в частотний діапазон. Показані кроки розробки, отримані результати, та перспективи подальшого розвитку програмного продукту.

Ключові слова: DDoS, nginx, кібер-безпека, аналіз даних, хмарні обчислення, AWS.

# ABSTRACT

**Sahan M.** Development of an intelligent system of protection against DDoS attacks based on nginx logs. Specialty 122 "Computer science", Educational program "Data science". Donetsk National University named after Vasyl Stus. 2022

In the qualification (master's) thesis, an application was developed for the protection of the web server based on the logs created by the web server, during operation, and frequency analysis of the user's entry into the frequency range. Development steps, obtained results, and prospects for further development of the software product are shown.

Keywords: DDoS, nginx, cyber security, data analysis, cloud computing, AWS.

# ВСТУП

На даний момент сфера IT розвивається надзвичайно швидкими темпами, нові проекти з’являються кожного дня. Завдяки хмарним провайдерам, кожен може отримати можливість створювати та розгортати свої рішення в короткий час та за розумні гроші. Хмарні сервіси звільняють користувачів від необхідності обслуговування власної інфраструктури, необхідності оплати роботи спеціалістів, адміністраторів, та позбавляють необхідності вивчати предмет самостійно. Все сказане вище значно знижує поріг входу. На даний момент лідером у цій сфері є платформа AWS. Платформа Amazon Web Services (AWS) надає понад 200 повнофункціональних послуг із центрів обробки даних, розташованих по всьому світу, і є найповнішою у світі хмарною платформою. Веб-сервіси Amazon — це онлайн-платформа, яка надає масштабовані та економічно ефективні рішення для хмарних обчислень. AWS — це широко поширена хмарна платформа, яка пропонує кілька операцій на вимогу, як-от обчислювальна потужність, зберігання бази даних, доставка контенту тощо, щоб допомогти компаніям масштабуватись і рости.

Хмарні рішення надають надзвичайну гнучкість при виборі ресурсів, але іноді, через це страждають характеристики компонентів. Так диски, які надає амазон, мають дуже низький показник IOPS (операції введення/виведення за секунду)

Разом із цим доволі гостро стає проблема захисту цих ресурсів від потенціальних нападників, які використовують як і специфічні дефекти системи для атаки, такими випадками займаються спеціалісти з кібербезпеки, так і більш прості, але не менш дієві DOS\DDOS атаки.

# РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ ТА РІШЕНЬ

## 1.1 Види кібератак

- **Програми-вимагачі (шифрувальники).** Шифрують всю інформацію на девайсах вашої компанії, що може призвести до повної зупинку бізнесу. Інколи така інформація навіть не підлягає відновленню.
- **Інсайдерські атаки**. Це найскладніший вид кіберзагроз, адже вони пов'язані з людським фактором. Інсайдером може бути співробітник, якому ви довіряєте. Він може зашкодити компанії як цілеспрямовано, так і випадково. Такий вид кібератаки складно передбачити.
- **Фішинг.** Одна з найефективніших атак. Полягає в розсилці кібер-зловмисником листів зі шкідливими файлами або посиланнями. Відкриваючи такі вкладення, людина заражає свій ПК. З цього розпочинається проникнення в мережу організації. Навіть досвідчені люди іноді потрапляють у цю пастку. Тому, якщо ви отримали підозрілого листа, краще взагалі його не відкривати.
- **Цільові кібератаки, DDoS-атаки** – це атаки на обчислювальну систему з метою довести її до відмови. Зловмисники прагнуть створити такі умови, щоб обмежити або взагалі заблокувати для користувачів системи доступ до системних ресурсів.

## 1.2 Специфіка захисту веб серверів

Для веб-серверів Nginx - рекомендуються такі заходи захисту:

1. Необхідно проводити регулярні оновлення серверного програмного забезпечення. Це один з найпростіших і дієвих способів захисту. Рекомендується перевіряти наявність оновлень не рідше 1 разу на 1-2 тижні.

2. При установці серверного ПО проводиться його діагностика і задаються налаштування за замовчуванням. Ці установки можуть бути

«лазом» для зловмисників. Після перевірки працездатності сервера слід видаляти налаштування за замовчуванням. Це ще одна нескладна захід із забезпечення безпеки серверів з ОС Linux.

3. Обговоріть з колегами, які саме модулі вам необхідні для роботи, інші непотрібні модулі закоментуйте. У разі необхідності ви завжди зможете повернути модуль, який закоментувавши.

4. Рекомендується відключити всі доступи до сайтів по IP-адресою. Зловмисники сканують саме їх, а користувачі «ходять» по доменному імені.

5. Обов'язково захистіть сервера від DoS і DDoS-атак. Докладні інструкції щодо захисту від даного виду атак можна знайти у відкритому доступі в інтернеті. У разі ж покупки, всі деталі захисту повинні пояснювати компанії-продавця. Цей захід безпеки актуальна для всіх типів серверів: як Linux, так і Windows.

6. Необхідно вести запис логів, щоб в разі виникнення помилок або якийсь нештатної ситуації можна було простежити, джерело. Вести подібні журнали 8 необхідно завжди і це один з найважливіших елементів забезпечення захисту [8].

Linux — це член сімейства Unix-подібних ОС з відкритим кодом на основі ядра Linux, спочатку ядра операційної системи. випущений 17 вересня 1991 року Лінусом Торвальдсом. Linux зазвичай упаковується в дистрибутив Linux. Дистрибутиви включають ядро Linux і допоміжне системне ПЗ та бібліотеки, багато з яких взяті з проекту GNU. Досить багато дистрибутивів Linux додають слово «Linux» до назви, але Free Software Foundation використовує назву «GNU/Linux», щоб підкреслити важливий вклад ПЗ GNU, що викликає певні протиріччя в людей та приводить до суперечок [9].

## 1.3 Сучасні тенденції у захисті

Сьогодні в центрі уваги інформаційній безпеці насамперед люди, так як причиною майже 80% всіх інцидентів є людський фактор. Звичайний рядовий співробітник або фахівець, який володіє інформаційними ресурсами компанії, може завдати непоправної шкоди. Кіберзлочинці добираються до необхідної їм конфіденційної інформації шляхом довготривалих націлених атак (APT). APT - термін кібербезпеки, що означає супротивника, що володіє досить сучасним рівнем спеціалізованих навичок та знань і значними ресурсами, які дають змогу створювати загрозу небезпечних кібератак.

Термін спочатку використовувався для опису кібернападів на військові організації, але не обмежений військовою сферою. Атака APT перевершує звичайні кіберзагрози, оскільки орієнтується на злам конкретної цілі та підготовка відбувається на підставі інформації про неї, що досліджується та збирається протягом досить тривалого часу. APT здійснює злам цільової інфраструктури використовуючи експлуатації методів «соціальної інженерії». та програмних вразливостей.

Станом на 2022 рік, не вироблено абсолютних методів протидії загрозам класу APT, вони продовжують розвиватися. Виявлення цільової атаки потребує ретельного аналізу подій безпеки за тривалий термін. Відмінним аспектом 9 наслідків атаки є відсутність гарантії повного відновлення та подальшої безпеки [10].

В середньому виявлення даної атаки відбувається через 200 днів, найчастіше зараження відбувається з вини співробітників, які недостатньо обізнані в області безпеки. Так як 85% дій циклічні і повторюються щодня, співробітник може навіть і не здогадуватися, що став причиною зараження всієї організації [11].

Одна з компаній, які надають послуги з кіберзахисту є Cisco. Cisco - американська транснаціональна компанія, що розробляє та продає

мережеве обладнання, призначене в основному для великих організацій та телекомунікаційних підприємств.

Одна з найбільших у світі компаній, що спеціалізуються у галузі високих технологій. Спочатку займалася лише корпоративними маршрутизаторами.

Однією з особливостей бізнес моделі компанії стала багаторівнева розгалужена система сертифікації інженерів з комп'ютерних мереж. Завдяки тому, що іспити цієї системи перевіряють знання і навички не тільки продукції Cisco, а й протоколів та мережевих технологій, багато організацій, що працюють на мережевому обладнанні інших фірм, підкреслюють цінність професійних сертифікатів Cisco. Також слід зазначити, що сертифікація на рівні експерта (CCIE) є однією з найвідоміших та найшанованіших у комп'ютерній індустрії. Це сертифікати, які є найбільш широко визнаними і шановними сертифікатами у галузі, сертифікати експертів Cisco свідчать світу, що ви знаєте, про що говорите [12][13].

Діяльність компанії постійно розширюється та охоплює все більше нових галузей IT-галузі. Cisco пропонує системи безпеки, телефонію на базі Інтернету, хмарні системи та багато інших рішень для спільної роботи та корпоративних мереж. Бере участь у розвитку Інтернету речей (IoT), проводить освітні програми та пропонує одну з найбільш затребуваних багаторівневих та ретельних систем сертифікації інженерів комунікаційних технологій [14].

IoT - Інтернет речей, відноситься до мільйонів та навіть мільярдів фізичних пристроїв по всьому світі, які прямо зараз підключені до мережі Інтернет, які збирають та обмінюються даними. Завдяки появі дуже дешевих чіпів та широкому поширенню бездротових мереж, можна перетворити будь що, від чогось маленького, як пігулка, до чогось великого, як корабель чи літак, в частину IoT. З’єднання цих об’єктів і додавання до них різноманітних датчиків дає високий рівень цифрового

інтелекту пристроям, які в іншому випадку були б безглуздими, дозволяючи їм надавати та передавати дані в режимі реального часу без підтримки чи контролю людьми. Інтернет речей робить весь сучасний світ більш розумним та чутливим, об’єднуючи цифровий і фізичний світи [15].

## 1.4 DoS атаки

Атака DOS - це спроба перевантажити онлайн-сервіс (веб-сайт) трафіком. Мета - порушити веб-сайт або мережу, щоб не допустити законних користувачів до доступу до послуги.

Атака DOS зазвичай запускається з однієї машини, на відміну від DDOS-атаки, яка запускається з декількох машин.

Ось гарна метафора.

Зобразіть торговий центр, де недавній інцидент озброїв активістів тварин. Ці активісти на тварин (нелегітимний рух) переповнюють вхід, щоб перекрити покупців (законний трафік) проникнути в приміщення.

Покупці не можуть потрапити до магазинів, а магазини втрачають гроші.

Це в значній мірі схожа на атаку DOS, метафорично кажучи.

## 1.5 DDoS атаки

DDOS-атаки зазвичай гірші, ніж DOS-атаки. Вони запускаються з декількох комп'ютерів. Задіяні машини можуть налічувати сотні тисяч і більше.

Звичайно, ці машини не належать нападнику. Ці машини зазвичай додаються до мережі хакера за допомогою шкідливих програм. Ця група машин також відома як ботнет[81].

Напад DDOS особливо важко захищати, оскільки дуже важко визначити законний трафік від трафіку нападника[82].

Існує багато різних DDOS-атак, таких як затоплення HTTP або SYN.

Затоплення HTTP - це лише практика надсилання на сервер тисячі і тисячі запитів, намагаючись перекрити його.

Потоп SYN заповнює мережу TCP непідтвердженими пакетами даних. Це може мати серйозні наслідки і навіть може вплинути на користувачів, не пов’язаних із передбачуваною жертвою.

1.4 Сучасні способи протидії кібератакам (у тому числі програмні засоби)

Для багатьох видів кібер атак існують протоколи протидії, так, наприклад при загрозі атак методом соціальної інженерії, проводять додаткові навчання для персоналу. При загрозі атак на вразливості ПЗ практикують своєчасний патчинг та обмеження доступу тільки з довірених джерел. Для протидії DOS/DDOS використовують як обмеження доступу до ресурсу, так і фільтри безпеки.

## 1.6 Огляд аналогів

### 1.6.1 Fail2ban

Fail2ban — це фреймворк програмного забезпечення для запобігання вторгненням. Написаний мовою програмування Python, він розроблений для запобігання атакам грубої сили. Він може працювати в системах POSIX, які мають інтерфейс до системи керування пакетами або локально встановленого брандмауера, наприклад iptables або TCP Wrapper. Fail2ban працює шляхом моніторингу файлів журналу (наприклад, /var/log/auth.log, /var/log/apache/access.log тощо) для вибраних записів і запуску сценаріїв на їх основі. Найчастіше це використовується для блокування вибраних IP-адрес, які можуть належати хостам, які намагаються порушити безпеку системи. Він може заборонити будь-яку IP-адресу хоста, який робить занадто багато спроб входу або виконує будь-яку іншу небажану дію протягом періоду часу, визначеного адміністратором. Включає підтримку

IPv4 і IPv6. За бажанням можна налаштувати довші заборони для кривдників-рецидивістів, які постійно повертаються. Fail2ban зазвичай налаштований на розблокування заблокованого хосту протягом певного періоду, щоб не «заблокувати» будь-які справжні з'єднання, які могли бути тимчасово неправильно налаштовані. Однак, часу розблокування в кілька хвилин зазвичай достатньо, щоб зупинити мережеве з'єднання, затоплене зловмисними з'єднаннями, а також зменшити ймовірність успішної атаки за словником.

### 1.6.2 iptables

iptables — це утиліта для користувача, яка дозволяє системному адміністратору налаштовувати правила фільтрації IP-пакетів брандмауера ядра Linux, реалізованого як різні модулі Netfilter. Фільтри організовано в різних таблицях, які містять ланцюжки правил обробки пакетів мережевого трафіку. Для різних протоколів зараз використовуються різні модулі ядра та програми; iptables застосовується до IPv4, ip6tables до IPv6, arptables до ARP, а ebtables до кадрів Ethernet.

### 1.6.3 Cloudflare WAF

Cloudflare WAF — це продукт безпеки, який, як і традиційний брандмауер, фільтрує потенційно зловмисний мережевий трафік. На відміну від традиційних брандмауерів, хмарні брандмауери розміщені в хмарі. Ця хмарна модель брандмауерів також називається брандмауер як послуга (FWaaS). Хмарні брандмауери створюють віртуальний бар'єр навколо хмарних платформ, інфраструктури та додатків так само, як традиційні брандмауери утворюють бар'єр навколо внутрішньої мережі організації. Хмарні брандмауери також можуть захистити локальну інфраструктуру.

## 1.7 Огляд обраного веб сервера

NGINX — це програмне забезпечення з відкритим вихідним кодом для веб-обслуговування, зворотного проксі-сервера, кешування, балансування навантаження, потокового передавання медіа тощо. Він починався як веб-сервер, розроблений для максимальної продуктивності та стабільності. Окрім можливостей HTTP-сервера, NGINX також може функціонувати як проксі-сервер для електронної пошти (IMAP, POP3 і SMTP), а також як зворотний проксі та балансувальник навантаження для серверів HTTP, TCP і UDP.

### 1.7.1 Передісторія

Ігор Сисоєв спочатку написав NGINX для вирішення проблеми C10K, термін, введений у 1999 році для опису труднощів, з якими стикаються існуючі веб-сервери при обробці великої кількості (10K) одночасних з’єднань (C). Завдяки своїй асинхронній архітектурі, керованій подіями, NGINX зробив революцію в роботі серверів у високопродуктивних контекстах і став найшвидшим доступним веб-сервером.

Відкривши проект у 2004 році та спостерігаючи за експоненціальним зростанням його використання, Сисоєв став співзасновником NGINX, Inc. для підтримки подальшого розвитку NGINX і просування NGINX Plus як комерційного продукту з додатковими функціями, призначеними для корпоративних клієнтів. NGINX, Inc. стала частиною F5, Inc. у 2019 році. Сьогодні NGINX і NGINX Plus можуть обробляти сотні тисяч одночасних з’єднань і обслуговувати більше найбільш завантажених сайтів Інтернету, ніж будь-який інший сервер.

### 1.7.2 NGINX як веб-сервер

Метою NGINX було створення найшвидшого веб-сервера, і підтримка цієї досконалості все ще є основною метою проекту. NGINX

стабільно перемагає Apache та інші сервери в тестах вимірювання продуктивності веб-серверів. Однак після початкового випуску NGINX веб-сайти розширилися від простих HTML-сторінок до динамічного багатогранного вмісту. NGINX виріс разом із цим і тепер підтримує всі компоненти сучасного Інтернету, включаючи WebSocket, HTTP/2, gRPC і потокове передавання кількох відео форматів (HDS, HLS, RTMP та інші).

### 1.7.3 NGINX Не тільки як веб-сервер

Хоча NGINX став відомим як найшвидший веб-сервер, масштабована базова архітектура виявилася ідеальною для багатьох веб-завдань, окрім обслуговування вмісту. Оскільки він може обробляти велику кількість з’єднань, NGINX зазвичай використовується як зворотний проксі-сервер і балансувальник навантаження для керування вхідним трафіком і розподілу його на повільніші висхідні сервери – від застарілих серверів баз даних до мікросервісів.

NGINX також часто розміщується між клієнтами та другим веб-сервером, щоб служити термінатором SSL/TLS або веб-прискорювачем. Діючи як посередник, NGINX ефективно вирішує завдання, які можуть уповільнити ваш веб-сервер, наприклад узгодження SSL/TLS або стиснення та кешування вмісту для підвищення продуктивності. Динамічні сайти, створені з використанням будь-чого від Node.js до PHP, зазвичай розгортають NGINX як кеш вмісту та зворотний проксі, щоб зменшити навантаження на сервери додатків і максимально ефективно використовувати базове обладнання.

## 1.8 Хмарний провайдер

Хмарний провайдер, це той хто дає в оренду, або продає обчислювальні ресурси, при цьому частина обслуговування останніх залишається його відповідальністю.

### 1.8.1 Хто такий постачальник хмарних послуг

Постачальник хмарних послуг — це компанія з інформаційних технологій (IT), яка надає своїм клієнтам обчислювальні ресурси через Інтернет і надає їх на вимогу. CSP добре підходять для організацій і окремих осіб, які не хочуть брати на себе відповідальність за встановлення програмного забезпечення, апаратного забезпечення чи мережевих ресурсів і їх обслуговування до кінця їхнього життєвого циклу.

Постачальники хмарних послуг часто класифікуються за типом ресурсу, який вони надають:

- Програмне забезпечення як послуга (SaaS) – цей тип CSP надає клієнтам готові програмні додатки, доступ до яких здійснюється через веб-браузер або інтерфейс прикладної програми (API).
- Інфраструктура як послуга (IaaS) – цей тип CSP надає клієнтам доступ до API та інших ресурсів, які полегшують створення віртуального центру обробки даних, здатного підтримувати робочі навантаження в багатохмарних середовищах.
- Платформа як послуга (PaaS) – цей тип CSP розширює можливості IaaS і SaaS, надаючи командам розробників доступ до бібліотек та інших програмних інструментів, які підтримують створення та розміщення власних хмарних програм.

### 1.8.2 AWS

Що означає Amazon Web Services (AWS)?

Amazon Web Services (AWS) — це комплексна служба віддаленого обчислення, яка надає інфраструктуру хмарних обчислень через Інтернет із сховищем, пропускною здатністю та налаштованою підтримкою інтерфейсів прикладного програмування (API).

Запущений у 2006 році, AWS надається піонером концепції хмарних рішень Amazon Inc. Внутрішнє управління IT-ресурсами Amazon створило AWS, який розширився та перетворився на інноваційного та економічно ефективного постачальника хмарних рішень.

Amazon запустив AWS під час ранньої перехідної фази хмарних обчислень. Перед запуском Amazon перебудував свою інфраструктуру, щоб консолідувати потужність серверів і сховище після того, як зрозумів, що їхні хост-сервери завантажені приблизно на 50 відсотків. AWS знаходиться в тій самій інфраструктурі, що й інші веб-ресурси Amazon, наприклад Webstore.

Amazon комплектує AWS масштабованими та практично необмеженими обчислювальними ресурсами, сховищем і пропускною здатністю. AWS використовує модель ціноутворення за передплатою за користуванням або оплатою за те, що використовуєш.

Послуги AWS включають:

- Amazon Elastic Computer Cloud (EC2)
- Проста служба зберігання Amazon (Amazon S3)
- Amazon CloudFront
- Служба реляційної бази даних Amazon (Amazon RDS)
- Служба простих сповіщень Amazon (Amazon SNS)
- Проста служба черги Amazon (Amazon SQS)
- Віртуальна приватна хмара Amazon (Amazon VPC)

Amazon EC2 і Amazon S3 — це дві основні служби «Інфраструктура як послуга» (IaaS), якими користуються розробники рішень хмарних додатків у всьому світі.

### 1.8.3 Диски в AWS (Elastic Block Store)

Amazon Elastic Block Store (EBS) – це сервіс, який пропонує Amazon, який зберігає інформацію для екземплярів Amazon Elastic Compute Cloud (EC2). Amazon EBS пропонує постійне блокове зберігання в хмарній системі Amazon Web Services (AWS). Це більша частина AWS, яка побудована на нових моделях хмарних обчислень і найсучасніших корпоративних сервісних архітектурах.

По суті, Amazon Elastic Block Store пропонує важливі переваги, які є частиною віртуалізованих або абстрактних архітектур. Він пропонує захист від збою компонентів, а також резервування та резервне копіювання з високою доступністю та, згідно з його назвою, еластичністю. EBS також пропонує клієнтам масштабованість і цінову гнучкість.

Незважаючи на те, що Amazon Elastic Block Store є прикладом використання потужності хмари для зберігання, це не панацея. Деякі системні адміністратори мають занепокоєння щодо того, як EBS абстрагує зберігання даних і, за словами одного блогера, «порушує принцип меж» — ідея полягає в тому, що без фізичного дискового сховища системи можуть мати проблеми із затримкою або бути важко доступними. виправляти помилки, навіть якщо вони можуть досягти вищих тестів продуктивності. Зрештою, для багатьох інженерів, які визнають різноманітні плюси та мінуси надсилання даних у дуже диверсифіковане та сильно розділене середовище зберігання, те, наскільки далеко зайти з концепціями зберігання постачальників, є компромісом.

### 1.8.4 IOPS

IOPS (операції введення/виведення за секунду) — це стандартна одиниця вимірювання максимальної кількості зчитувань і записів у несуміжні місця зберігання. IOPS вимовляється як EYE-OPS.

Постачальники систем зберігання даних часто посилаються на IOPS для характеристики продуктивності твердотільних накопичувачів (SSD), жорстких дисків (HDD) і мереж зберігання даних. Однак число IOPS не є фактичним еталонним показником, і цифри, рекламовані постачальниками, можуть не відповідати реальній продуктивності.

Разом зі швидкістю передачі даних, яка визначає швидкість передачі даних із суміжних місць зберігання, IOPS можна використовувати для вимірювання продуктивності сховища. Хоча швидкість передачі вимірюється в байтах, IOPS вимірюється як ціле число.

Як вимірювання, IOPS можна порівняти з обертами за хвилину (об/хв) двигуна автомобіля. Якщо автомобіль знаходиться в нейтральному положенні, стверджувати, що двигун здатний обертатися зі швидкістю 10 000 об/хв у цей момент, немає сенсу. Без урахування розміру блоку даних (або розміру вводу/виводу), активності читання/запису або потоку вводу/виводу, IOPS як окреме вимірювання мало що говорить.

### 1.9 Історія регулярних виразів

Витоки регулярних виразів лежать у теорії автоматів та теорії формальних мов. Ці області вивчають обчислювальні моделі (автомати) та способи опису та класифікації формальних мов. У 1940-х роках. Уоррен Маккалок і Уолтер Піттс описали нервову систему, використовуючи простий автомат як модель нейрона. Математик Стівен Кліні пізніше описав ці моделі, використовуючи свою систему математичних позначень, названу «регулярні множини». Кен Томпсон вбудував їх у редактор QED, а потім у редактор expr, awk, vi, Perl.Регулярні вирази в Perl та Tcl походять від реалізації, написаної Генрі Спенсером. Філіп Хейзел розробив бібліотеку англ. Perl-compatible regular expressions (Perlсумісні регулярні вирази), яка використовується в багатьох сучасних інструментах, таких як Apache.

Регулярні вирази складаються з констант та операторів, які визначають безліч рядків та безлічі операцій на них відповідно. На цьому кінцевому алфавіті Σ визначено такі константи:

● (порожня множина) ∅.

● (порожній рядок (англ.)) ε позначає рядок, що не містить жодного символу. Еквівалентно "". 25

● (Рядок) «a», де a - символ алфавіту Σ, позначає рядок, що складається з одного цього символу. та наступні операції:

● (зчеплення, конкатенація) RS означає безліч {αβ | α ∈ R & β ∈ S}. Наприклад, {"boy", "girl"} {"friend", "cott"} = {"boyfriend", "girlfriend", "boycott", "girlcott"}.

● (Диз'юнкція, перерахування) R | S означає об'єднання R і S.

● (замикання Кліні, зірка Кліні) R* позначає мінімальне надмножина множини R, яка містить ε і замкнута щодо конкатенації. Це безліч всіх рядків, отриманих конкатенацією нуля або більше рядків з R. Наприклад, {«Go», «Ukraine»}* = {ε, «Go», «Ukraine», «GoGo», «GoUkraine», «UkraineGo» , "UkraineUkraine", "GoGoGo", "GoGoUkraine", "GoUkraineGo", …} [34].

# РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ

## 2.1 Вступ

На даний момент сфера IT розвивається надзвичайно швидкими темпами, нові проекти з’являються кожного дня. Разом із цим доволі гостро стає проблема захисту цих ресурсів від потенціальних нападників, які використовують як і специфічні дефекти системи для атаки, такими випадками займаються спеціалісти з кібербезпеки, так і більш прості, але не менш дієві DOS\DDOS атаки.

Атака на відмову в обслуговуванні, розподілена атака на відмову в обслуговуванні (англ. DoS attack, DDoS attack, (Distributed) Denial-of-service attack) — напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена.

Одним із найпоширеніших методів нападу є насичення атакованого комп'ютера або мережевого устаткування великою кількістю зовнішніх запитів (часто безглуздих або неправильно сформульованих) таким чином атаковане устаткування не може відповісти користувачам, або відповідає настільки повільно, що стає фактично недоступним. Взагалі відмова сервісу здійснюється:

- примусом атакованого устаткування до зупинки роботи програмного забезпечення/устаткування або до витрат наявних ресурсів, внаслідок чого устаткування не може продовжувати роботу;
- заняттям комунікаційних каналів між користувачами і атакованим устаткуванням, внаслідок чого якість сполучення перестає відповідати вимогам.

Якщо атака відбувається одночасно з великої кількості IP-адрес, то її називають розподіленою (англ. Distributed Denial-of-Service — DDoS).

Такі атаки наносять шкоду у вигляді втрати потоку клієнтів для сервісу та додаткових витрат на інфраструктуру.

## 2.2 Аналіз предметної області

На даний момент існує досить багато засобів для захисту від ддос атак.

- Фаєрволи за рівнем

Найпростіший, проте, такий що підходить тільки для специфічних продуктів спосіб, це використовувати фільтрацію по вайт-лісту, тобто надавати доступ до системи тільки завідомо легітимним користувачам. Такий спосіб часто використовується при захисті різноманітних адміністративних ресурсів, проте не підходить для ресурсів із загальним доступом.

Наступним типом фаєрволів є такі що перевіряють наскільки багато запитів генерує клієнт (частотне відсічення), так наприклад, якщо при нормальній роботі один клієнт генерує потік у 10 запитів на секунду то значення у 100 запитів буд підозрілим. Підтипом таких фаєрволів також являються фаєрволи із відсіченням по помилці, тобто, якщо від користувача приходить, наприклад, багато запитів про неправильні логін та пароль, то швидше за все він намагається підібрати пароль. Деякі із таких фаєрволів уміють блокувати користувачів поступово, тобто, при першому порушенні користувач блокується на 60 секунд, а після, наприклад, 10 таких випадків, настає перманенте блокування. Всі вказані вище функції реалізують такі системи як fail2ban або fail2ban.

Наступним кроком у галузі захисту від атак стали так звані фаєрволи додатків, які зазвичай передбачають взаємодію із користувачем. Популярним в цьому сегменті є CLoudFlare DDoS protection, режим роботи при якому користувачам перед входом на сайт пропонується пройти капчу, і таким чином довести що він не робот.

- Фаєрволи за інфраструктурним типом розміщення

Фаєрвол може розміщуватись як на Вашій інфраструктурі, так і бути стороннім сервісом. Фаєрвол як сторонній сервіс дуже просто інтегрувати та обслуговувати, але майже неможливо контролювати, а власний сервіс, навпаки, важче інтегрувати і підтримувати, але всі тонкі налаштування та масштабування залишаються підконтрольними Вам.

- Фаєрволи за мережевим типом розміщення

Фаєрвол може розташовуватись як на хості, де працює ПП, так і в комутаторах перед хостом. Як правило, фаєрволи на хостах мають набагато простіший функціонал.

- Фаєрволи за типом перехоплення.

Відрізняються тим, на якому рівні мережевої моделі він працює, зазвичай це 4, або 7 рівень. Фаєрволи канального рівня(4) це фаєрволи які використовують для блокування кількість пакетів в секунду та іп-адресу. Фаєрволи додатків, це ПП що передбачають взаємодію з контентом на рівні HTTP/S (як наприклад показ капчі)

## 2.3 Постановка задачі

Необхідно створити легкий в плані споживання ресурсів продукт, що дозволить ефективно обробляти логи веб-сервера нджинкс та реагувати на загрози. Продукт повинен мінімально впливати на роботу сервера, в плані споживання його ресурсів та створення затримок. Необхідно спроектувати продукт таким чином, щоб він міг ефективно працювати із повільними дисками, або уникнути їх використання взагалі. Джерелом логів буде веб сервер/зворотній проксі NGINX, так як він являє собою швидкий і максимально продуктивний програмний продукт, який прекрасно працює на широкому спектрі засобів, та може обробляти великі обсяги трафіку. Даний веб сервер широко використовується у високонавантажених

системах для роздачі корисного трафіку, або як проксі. Також його можна дуже тонко налаштувати що безсумнівно є плюсом.

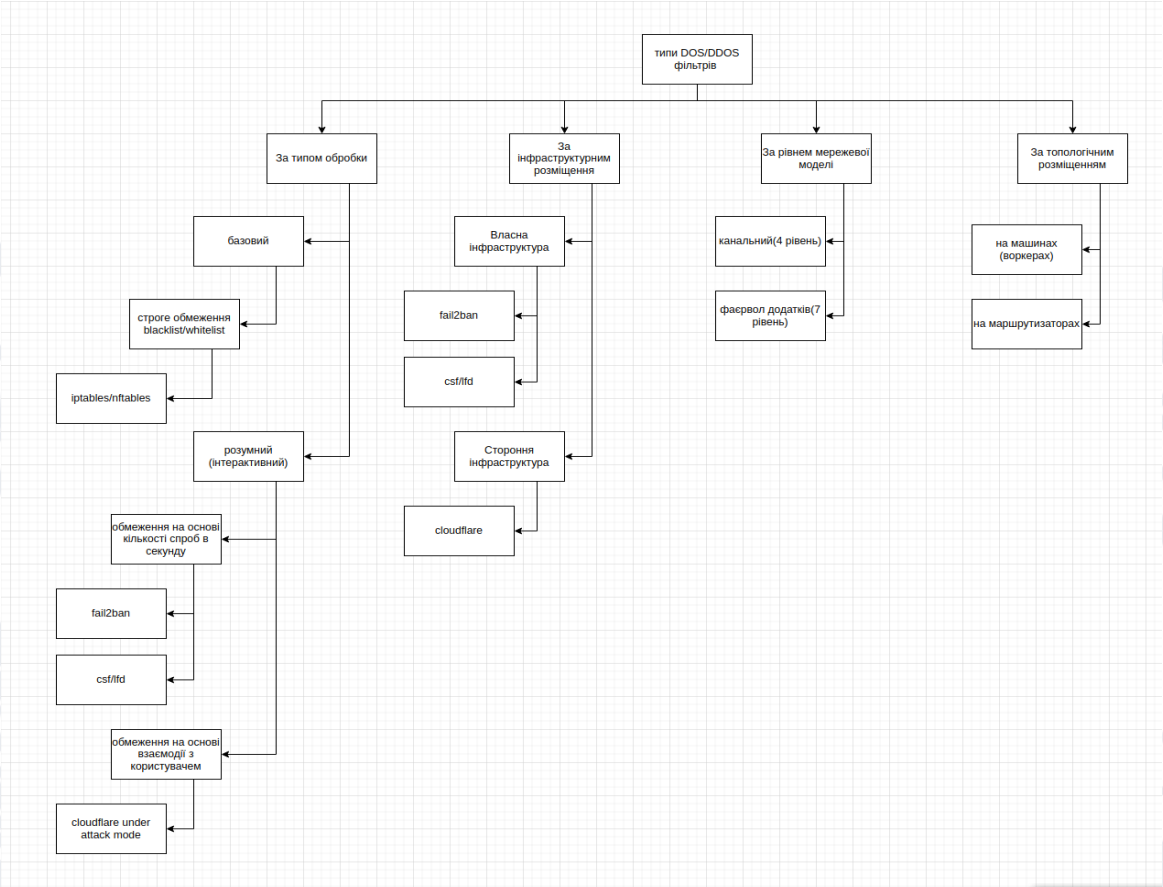



Рис. 2.1 - топологія фаєрволів

## 2.4 специфіка вхідних даних

Вхідними даними для програми послужить лог запитів, який створюється та наповнюється веб сервером. Із цих даних сервер буде отримувати інформацію про ip-адресу користувача, та час надходження запиту. як уже було сказано раніше, обраний веб-сервер можна тонко налаштувати, це також відноситься до формату логів. Окрім формату логів, Буде розглянуто можливості розміщення логів у різних місцях(як от ОЗУ, або файл на диску машини). У загальному, лог являє собою послідовність рядків, розділену символом переходу на нову строку (\n).

## 2.5 специфіка середовища запуску

Оскільки програмний продукт, який буде створено, орієнтований на хмарні системи, то для розробки і тестування буде обрано хмарну інфраструктуру. В даному випадку обрано хмарного провайдера AWS.

### 2.5.1 AWS

Платформа Amazon Web Services (AWS) надає понад 200 повнофункціональних послуг із центрів обробки даних, розташованих по всьому світу, і є найповнішою у світі хмарною платформою. Веб-сервіси Amazon — це онлайн-платформа, яка надає масштабовані та економічно ефективні рішення для хмарних обчислень. AWS — це широко поширена хмарна платформа, яка пропонує кілька операцій на вимогу, як-от обчислювальна потужність, зберігання бази даних, доставка контенту тощо, щоб допомогти компаніям масштабуватись і рости.

### 2.5.2 AWS Диски

AWS Elastic Block Store (EBS) — це рішення Amazon для зберігання на блочному рівні, яке використовується з хмарним сервісом EC2 для зберігання постійних даних. Це означає, що дані зберігаються на серверах AWS EBS, навіть якщо екземпляри EC2 вимкнено. EBS пропонує таку ж високу доступність в межах вибраної зони доступності, дозволяючи користувачам масштабувати ємність сховища за низькою моделлю ціноутворення на основі підписки. Томи даних можна динамічно приєднувати, від’єднувати та масштабувати за допомогою будь-якого екземпляра EC2, як і фізичний блоковий накопичувач. Як надзвичайно надійний хмарний сервіс, пропозиція EBS гарантує 99,999% доступності. Даний тип дисків пропонує низький рівень IOPS, та доволі непогану пропускну здатність (в районі 125 мб/с при середніх розмірах диску)

### 2.5.3 IOPS

IOPS (операції введення/виведення за секунду) — це стандартна одиниця вимірювання максимальної кількості зчитувань і записів у несуміжні місця зберігання. IOPS вимовляється як EYE-OPS.

Постачальники систем зберігання даних часто посилаються на IOPS для характеристики продуктивності твердотільних накопичувачів (SSD), жорстких дисків (HDD) і мереж зберігання даних. Однак число IOPS не є фактичним еталонним показником, і цифри, рекламовані постачальниками, можуть не відповідати реальній продуктивності.

Разом зі швидкістю передачі даних, яка визначає швидкість передачі даних із суміжних місць зберігання, IOPS можна використовувати для вимірювання продуктивності сховища. Хоча швидкість передачі вимірюється в байтах, IOPS вимірюється як ціле число.

Як вимірювання, IOPS можна порівняти з обертами за хвилину (об/хв) двигуна автомобіля. Якщо автомобіль знаходиться в нейтральному положенні, стверджувати, що двигун здатний обертатися зі швидкістю 10 000 об/хв у цей момент, немає сенсу. Без урахування розміру блоку даних (або розміру вводу/виводу), активності читання/запису або потоку вводу/виводу, IOPS як окреме вимірювання мало що говорить.

### 2.5.4 затримка та пропускна спроможність

Пропускна здатність вимірює, скільки одиниць інформації може обробити система за певний період часу. Це може стосуватися кількості операцій введення-виведення за секунду, але зазвичай вимірюється в байтах за секунду. Самі по собі IOPS і пропускна здатність не можуть забезпечити точне вимірювання продуктивності.

Затримка вимірює час між надсиланням запиту та отриманням відповіді. Що стосується IOPS, затримка – це міра часу, який потрібен для виконання одного запиту введення-виведення з точки зору програми. Хоча

це не дає повної картини, поєднання вимірювань затримки, IOPS і пропускної здатності може допомогти оцінити продуктивність.

### 2.5.5 Проблеми низького показника IOPS

Якщо не брати до уваги диски з високим рівнем IOPS, за які необхідно доплачувати (зазвичай такі диски використовуються для баз даних) звичайні диски, які пропонує амазон мають доволі низький показник IOPS, приблизно 300 для диску середнього розміру, тоді як ССД накопичувачі у сучасних комп'ютерах пропонують 500000 і більше IOPS. Такий рівень може стати проблемою при частому читанні/записі на диск, як у випадку із записом логів веб-сервером, так і в випадку із читанням логу для його аналізу фільтром. Це може призвести як до зменшення продуктивності веб сервера, так і до того що сканер не зможе своєчасно реагувати на загрози.

## 2.6 Основні критерії оптимізації

Під час дослідження найближчого аналогу програмного продукту (fail2ban), стало зрозуміло, що більшість процесорного часу витрачається на парсинг файлу з логами за допомогою регулярних виразів.

```
$ python -m cProfile -s cumtime /usr/bin/fail2ban-regex nginx.log '^<ADDR>'

...

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.001    0.001   52.734   52.734 fail2ban-regex:26(<module>)
        1    0.000    0.000   52.640   52.640 fail2banregex.py:784(exec_command_line)
        1    0.000    0.000   52.638   52.638 fail2banregex.py:719(start)
        1    0.411    0.411   52.636   52.636 fail2banregex.py:571(process)
   100000    0.789    0.000   50.909    0.001 fail2banregex.py:448(testRegex)
   100000    0.689    0.000   49.839    0.000 filter.py:601(processLine)
   100000    3.741    0.000   39.767    0.000 datedetector.py:321(matchTime)
  1595350    0.945    0.000   34.731    0.000 datetemplate.py:157(matchDate)
  1695573   33.933    0.000   33.933    0.000 {method 'search' of '_sre.SRE_Pattern'
   100000    0.205    0.000    5.583    0.000 datedetector.py:469(getTime)
   100000    0.187    0.000    5.305    0.000 datetemplate.py:323(getDate)
   100000    1.826    0.000    4.858    0.000 strptime.py:172(reGroupDictStrptime)
   100000    1.284    0.000    3.519    0.000 filter.py:811(findFailure)
  1400000    0.564    0.000    1.479    0.000 utf_8.py:15(decode)
200000/100000    0.615    0.000    1.303    0.000 strptime.py:143(zone2offset)
   100000    0.154    0.000    1.100    0.000 strptime.py:124(validateTimeZone)
   100000    0.422    0.000    0.965    0.000 {method 'index' of 'list' objects
```

Рис. 2.2 розподіл часу виконання

### 2.6.1 Що таке регулярні вирази

Регулярний вираз — це послідовність символів, яка визначає шаблон пошуку в тексті. Зазвичай такі шаблони використовуються алгоритмами пошуку рядків для операцій «знайти» або «знайти та замінити» над рядками або для перевірки вхідних даних. Методи регулярних виразів розроблені в теоретичній інформатиці та теорії формальної мови.

Концепція регулярних виразів почалася в 1950-х роках, коли американський математик Стівен Коул Кліні формалізував концепцію регулярної мови. Вони увійшли в загальне використання з утилітами обробки тексту Unix. Різні синтаксиси для написання регулярних виразів

існували з 1980-х років, один із яких був стандартом POSIX, а інший, широко використовуваний, був синтаксисом Perl.

### 2.6.2 Швидкодія регулярних виразів

Регулярні вирази являють собою кінечні автомати, тобто в роботі перевіряють різні варіанти, навіть якщо користувач використовує однаковий формат даних постійно. Дана технологія прекрасно підходить для пошуку та аналізу неоднорідного тексту, та пошуку певних патернів. Але в нашому випадку, коли структура тексту повністю визначена, від регулярних виразів можна відмовитись на користь більш швидких методів пошуку.

### 2.6.3 Проблема читання даних при низькому IOPS

Як уже було сказано раніше, при частому читанні/записі на диск, як у випадку із записом логів веб-сервером, так і в випадку із читанням логу для його аналізу фільтром. Це може призвести як до зменшення продуктивності веб сервера, так і до того що сканер не зможе своєчасно реагувати на загрози. Диски пропонують нам низький IOPS, при високій пропускній здатності, це підштовхує нас спробувати читати файл блоками, і зберігати ці блоки в ОЗУ. Альтернативним варіантом є відмова від використання диску, та запис логу одразу в ОЗУ, яка є набагато швидшою в порівнянні навіть з найшвидшими дисками.

# РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Вибір мови програмування для реалізації

Даний підрозділ присвячений вибору мови програмування для реалізації програмного продукту.

### 3.1.1 Вимоги до мови програмування

Оскільки основною задачею, яку необхідно вирішити, є не власна продуктивність готового програмного продукту, а проблема неможливості достатньо часто читати з апаратного накопичувача системи, до мови програмування не постає вимога в високій утилізованості (продуктивності на такт). Обрана мова повинна бути поширена серед мережевих інженерів, дев-опс спеціалістів, та адміністраторів, тобто серед тих, кому потрібно буде взаємодіяти із програмним продуктом, який буде результатом розробки. Це означає що програмний код буде для них зрозумілим і, за необхідності, вище вказані спеціалісти зможуть без особливих проблем відлагодити програму, або внести в неї свої корективи, також це сприятиме кращому розумінню принципів роботи програмного продукту. Мова повинна сприяти швидкі та легкій розробці програмного продукту, її текст повинен бути максимально зрозумілим. Також бажано щоб мова містила велику кількість потрібного нам базового функціоналу ‘із коробки’, що дозволить покращити портативність.

### 3.1.2 Python

Python — це інтерпретована мова програмування високого рівня загального призначення, яка широко використовується для розробки веб-сайтів, аналізу даних і автоматизації.

Python є мовою загального призначення, що означає, що він універсальний і може використовуватися для програмування багатьох різних типів функцій. Оскільки це інтерпретована мова, вона виключає

необхідність компіляції коду перед виконанням, а оскільки це мова програмування високого рівня, Python здатний абстрагувати деталі від коду. Насправді Python так багато уваги приділяє абстракції, що його код може зрозуміти більшість програмістів-початківців.

Код Python, як правило, короткий, і в порівнянні з компільованими мовами, такими як C і C++, він виконує програми повільніше. Його зручність робить його популярною мовою для громадянських розробників, які працюють з алгоритмами машинного навчання в програмних додатках із низьким кодом без коду (LCNC).

Python має простий синтаксис і відомий своєю великою спільнотою, яка активно сприяє зростанню вибору програмних модулів і бібліотек. Початкову розробку Python очолив Гвідо ван Россум наприкінці 1980-х років. Сьогодні Python керує Python Software Foundation

Плюси python:

- Простий у використанні та вивченні: для початківців Python простий у використанні. Це мова програмування високого рівня, а її синтаксис схожий на англійську мову. Ці причини роблять мову легкою для вивчення та адаптації. Порівняно з Java і C, у Python те саме завдання можна виконати за допомогою меншої кількості рядків коду. Завдяки легкому освоєнню принципи Python можна виконувати швидше порівняно з іншими мовами.
- Підвищення продуктивності: Python є дуже продуктивною мовою. Проста природа Python допомагає розробникам зосередитися на вирішенні проблем у ньому. Щоб зрозуміти синтаксис і поведінку мови програмування, користувачам не потрібно витрачати години, тому виконується більше роботи.
- Гнучкість: ця мова є дуже гнучкою, і тому вона дозволяє користувачеві пробувати нові речі. Користувачі можуть розробляти нові види програм, використовуючи мову програмування Python. Мова не обмежує

користувача спробувати щось інше. Інші мови програмування не надають такої гнучкості та свободи, тому Python є кращим у цих питаннях.

- Велика бібліотека: Python надає користувачеві величезну бібліотеку. Стандартна бібліотека Python величезна, і майже кожна функція, яку потрібно виконати, доступна в її бібліотеці. Це тому, що він має величезну підтримку спільноти та корпоративного спонсорства. Під час роботи з Python користувачі не використовують зовнішні бібліотеки.
- Спільнота підтримки: Мова Python була створена багато років тому, і тому вона має зрілу спільноту, яка може підтримувати будь-який тип розробника, починаючи від рівня початківців до рівня експертів. Для мови програмування Python доступно достатньо посібників, підручників та документації, які допомагають розробникам швидше та краще зрозуміти мову. Завдяки спільноті підтримки Python швидко розвивається порівняно з іншими мовами.

Мінуси python:

- Швидкість: порівняно з Java або C швидкість Python нижча. Python — це інтерпретована мова, яка динамічно типізується. Для виконання коду кожен рядок коду має бути чітко впорядкований, оскільки мова інтерпретується. Це займає багато часу, а отже, уповільнює процес виконання. Динамічна структура Python також уповільнює його швидкість, тому що під час виконання коду необхідно виконати зайву роботу. Тому у тих випадках, коли потрібне швидке прискорення, Python використовується не дуже часто.
- Споживання пам'яті: Python має дуже високе споживання пам'яті. Це тому, що він гнучкий до типів даних. Він використовує великий обсяг пам'яті. Python не є гарним вибором для завдань, де користувач хоче оптимізувати пам’ять, тобто це мова, яка потребує інтенсивного використання пам’яті.
- Розробка для мобільних пристроїв: Python сильний у серверних платформах і настільних комп’ютерах, а отже, це фантастична мова

серверного програмування. Але це не підходить для мобільної розробки. Для мобільної розробки Python є крихкою мовою. Через це Python не має багато вбудованих програм для мобільних пристроїв, тому що він не економить пам’ять і має тривалу потужність для обробки. Carbonnelle — вбудована програма в Python.

- Доступ до бази даних: Python забезпечує просте програмування. Однак, коли він взаємодіє з базою даних, виникають деякі проблеми. У порівнянні з такими технологіями, як JDBC і ODBC, які є досить відомими, рівень доступу до бази даних мови програмування Python є примітивним і недостатньо розвиненим. Великі підприємства, яким зазвичай потрібна плавна взаємодія зі складними застарілими даними, не віддають перевагу використанню Python.
- Помилки виконання: користувачі Python згадували різні проблеми, з якими вони зіткнулися під час розробки мови. Оскільки мова Python динамічно типізована, тип даних змінної може бути змінений у будь-який час. Тому його потрібно тестувати частіше, а також є помилки в мові, яка відображається під час виконання.
- Простота: Python є простою та легкою у використанні мовою програмування, що також є недоліком мови. Користувачі Python настільки звикають до його легкого синтаксису та великої бібліотеки, що стикаються з проблемами під час вивчення інших мов програмування. Деякі користувачі також вважають, що коди Java непотрібні через їх складність. Тому Python має дуже вразливий характер, і користувачі починають сприймати все легковажно.

### 3.1.3 Причини для вибору мови програмування

У даному випадку основним критерієм вибору мови пайтон стало те, що вона дуже розповсюджена серед мережевих інженерів, дев-опс спеціалістів, та адміністраторів, тобто серед тих, кому потрібно буде взаємодіяти із програмним продуктом, який буде результатом розробки.

Це означає що програмний код буде для них зрозумілим і, за необхідності, вище вказані спеціалісти зможуть без особливих проблем відлагодити програму, або внести в неї свої корективи, також це сприятиме кращому розумінню принципів роботи програмного продукту.

У даному випадку необхідно в більшій мірі вирішити проблему пов'язану з низьким рівнем швидкодії апаратного забезпечення системи, ніж продуктивністю системи в цілому, тому нижча швидкість роботи інтерпретованої мови програмування пайтон не стане проблемою.

### 3.2 Тестування різних методів читання логів

Для того аби визначитись із методом читання файлу з диску були обрані та протестовані наступні методи:

- Потокове читання з диску (як контрольний показник)
- Читання з UNIX FIFO стеку
- Читання з диску блоками(чанками)
- Читання з оперативної пам'яті

#### 3.2.1 Потокове читання з диску.

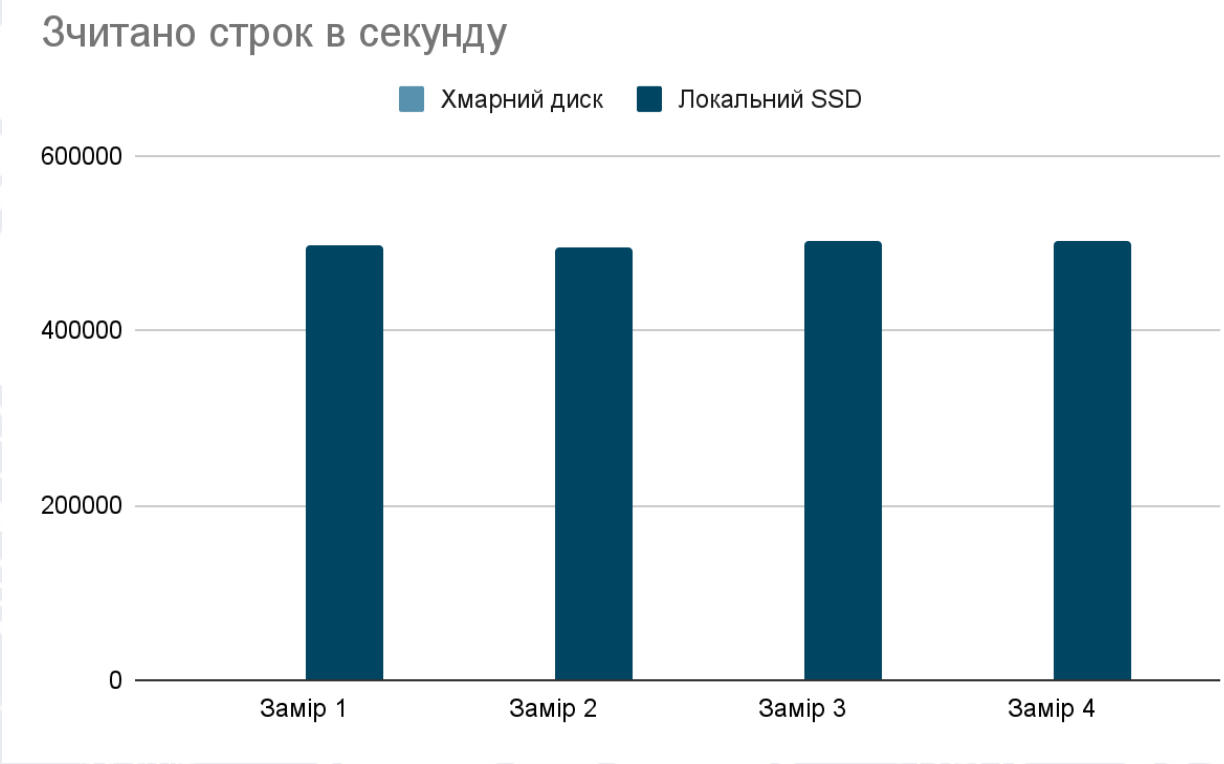
Потокове читання, це найпростіший метод читання з диску, коли файл відкривається на читання і постійно перевіряється на наявність нових рядків для читання. Проблема такого методу в середовищі малої кількості IOPS в тому що дані можна забирати не частіше ніж цей показник. Даний метод був взятий і протестований виключно з ціллю порівняти результати інших методів і не буде розглядатись як кандидат на використання. Для замірів були обрані 2 середовища, аби показати важливість дискової підсистеми в швидкості читання файлу.

Хмарний диск: AWS GP2 SSD (400 IOPS 125mb/s)

Локальний SSD: Samsung evo 980 (500000 IOPS 3500mb/s)

Таблиця 3.1 Потокове читання з диску

| | Хмарний диск | Локальний SSD |
|---|---|---|
| Замір 1 | 518 | 498248 |
| Замір 2 | 563 | 495254 |
| Замір 3 | 502 | 502859 |
| Замір 4 | 524 | 501664 |




Діаграма 3.1 Потокове читання з диску

Як результат заміру ми можемо спостерігати наскільки важливий показник iops при читанні файлу, та наскільки низький цей показник у хмарного накопичувача. Міра, яка була використана при замірах, це кількість зчитаних рядків за секунду. Насправді в буденних задачах достатньо рівня в приблизно 500 рядків за секунду, адже не кожен сервер обробляє 500 запитів в секунду.

### 3.2.2 Читання з диску блоками

При потоковому читанні з диску ми постійно звертаємось до області пам'яті, після чого звільняємо дескриптор (обробник зони пам’яті).

Оскільки це відбувається при читанні кожного рядка, то ми не можемо читати швидше ніж показник кількості IOPS (теоретично) що, за виключенням інших факторів, і показують попередні тести. Аби вирішити це питання, можна спробувати читати дані з диску блоками, Тобто, відкривши дескриптор (обробник зони пам’яті)один раз, ми починаємо послідовно зчитувати дані з диску, не виснажуючи показник IOPS, а впираючись у показник пропускної можливості диску(який є доволі великим у ABC дисків, у порівнянні з  IOPS) Суб'єкти тесту такі ж як і в попередньому випадку.

Хмарний диск: AWS GP2 SSD (400 IOPS 125mb/s)

Локальний SSD: Samsung evo 980 (500000 IOPS 3500mb/s)

Таблиця 3.2 Читання з диску блоками

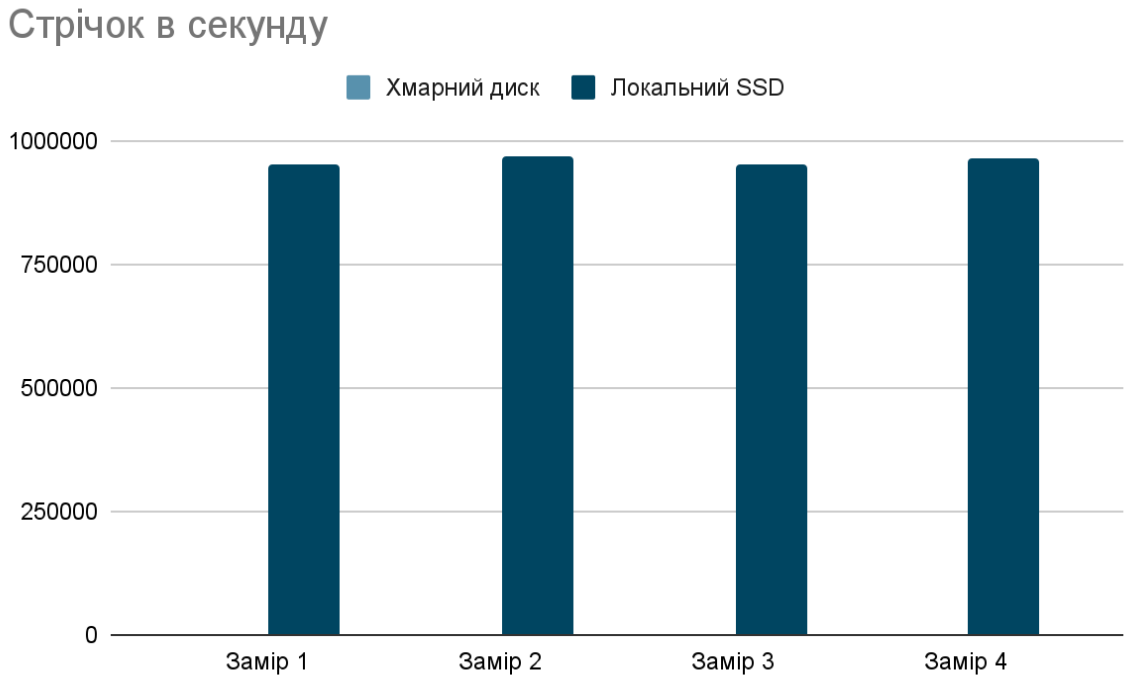
| | Хмарний диск | Локальний SSD |
|---|---|---|
| Замір 1 | 1631 | 953112 |
| Замір 2 | 1602 | 968247 |
| Замір 3 | 1615 | 952658 |
| Замір 4 | 1593 | 962354 |

Як можна побачити, читання блоками позитивно відобразилося на кількості зчитаних і відповідно оброблених строк в секунду. Для хмарного ССД показник виріс більш ніж на 300%. І хоча цей покажчик все ще далекий від показників локального диску, слід зауважити і різницю в пропускній здатності досліджуваних дисків(125 МБ/С проти 3500 МБ/С).

### 3.2.3 Читання з оперативної пам'яті.

На лінукс та юнікс машинах, за замовчуванням є можливість працювати з пам'яттю, як з диском. У таких системах частина ОЗУ підмонтовується як диск

за шляхом /dev/shm. Це дає нам змогу перевірити як відбуватиметься обробка, при читанні напряму з оперативної пам’яті.



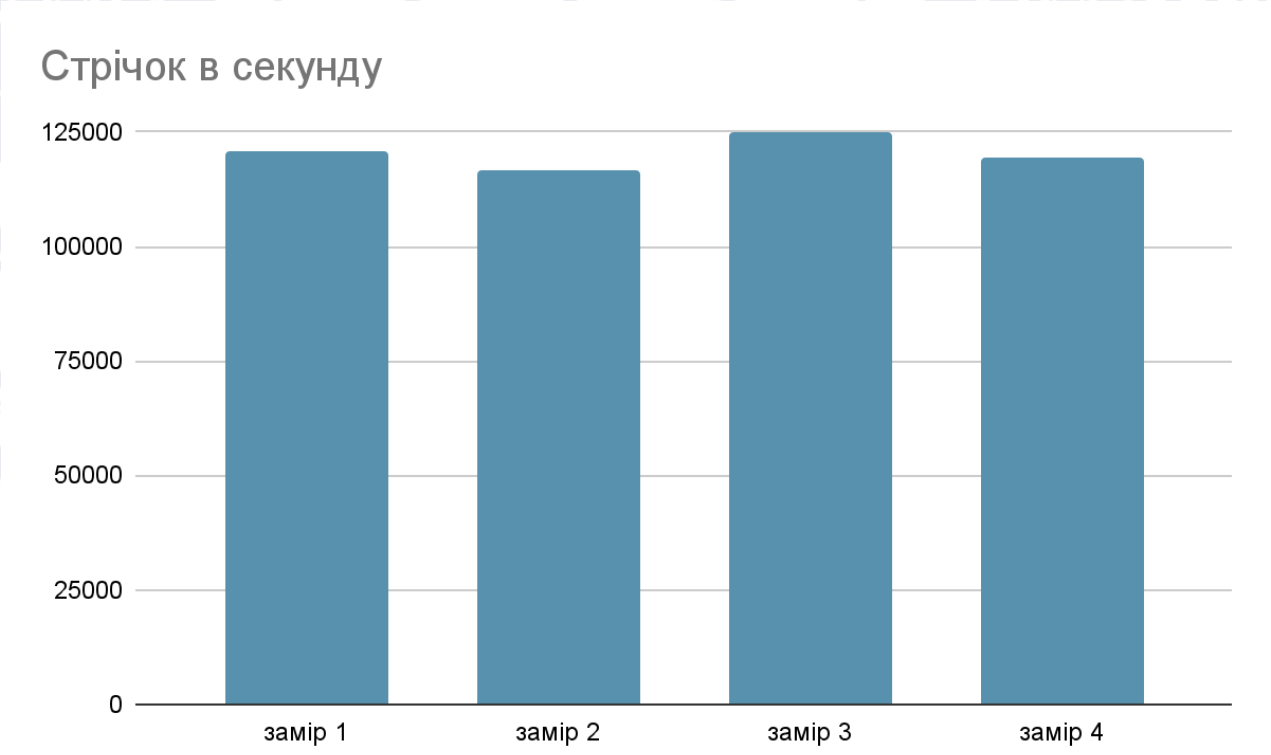

Діаграма 3.2 Читання з диску блоками

Також є спосіб записувати логи напряму в пам’ять, без використання примонтованих віртуальних директорій, але він потребує значних змін у веб-сервері, тому був відкинутий в даній роботі. Як відомо, оперативна пам'ять має вражаючу швидкість запису/читання та швидкість доступу до даних. Даний замір буде проведено тільки для хмарного екземпляру, оскільки на таких швидкостях в гру вступає процесор, і його швидкість, тому показники з двох систем можуть бути оманливі.

Таблиця 3.3 Читання з оперативної пам'яті.

| | |
|---|---|
| замір 1 | 120584 |
| замір 2 | 116854 |
| замір 3 | 124859 |
| замір 4 | 119254 |

Як можна побачити швидкість читання з оперативної пам’яті в порівнянні з хмарним диском вражає, хоча і є меншою ніж локальний ССД, причиною тому є швидкість процесора в хмарній машині, і на локальному стенді(1 ядро один потік проти 6 ядер 12 потоків), як і було сказано вище, в таких обставинах роль процесора виростає. Хоча робота з ОЗУ замість диску дала приріст у більш ніж 20000%, запис в оперативну пам’ять суттєво впливає на ресурси системи, тобто, пам’ять зайнята файлом логу може бути використана для більш корисних операцій. Найбезпечніший варіант, коли програмний продукт по якійсь причині вийде з ладу і перестане вичитувати лог з оперативної пам’яті, при цьому веб-сервер, який продовжує генерувати логи призведе до зависання серверу, шляхом переповнення пам'яті. Додаткові заміри показали що під навантаженням 100000 запитів у секунду, навіть з коротким форматом логів, веб-сервер генерує до 50 МБ логів кожних 5 секунд. Таким чином розміщення логів в ОЗУ не є безпечним, не дивлячись на всі його переваги.



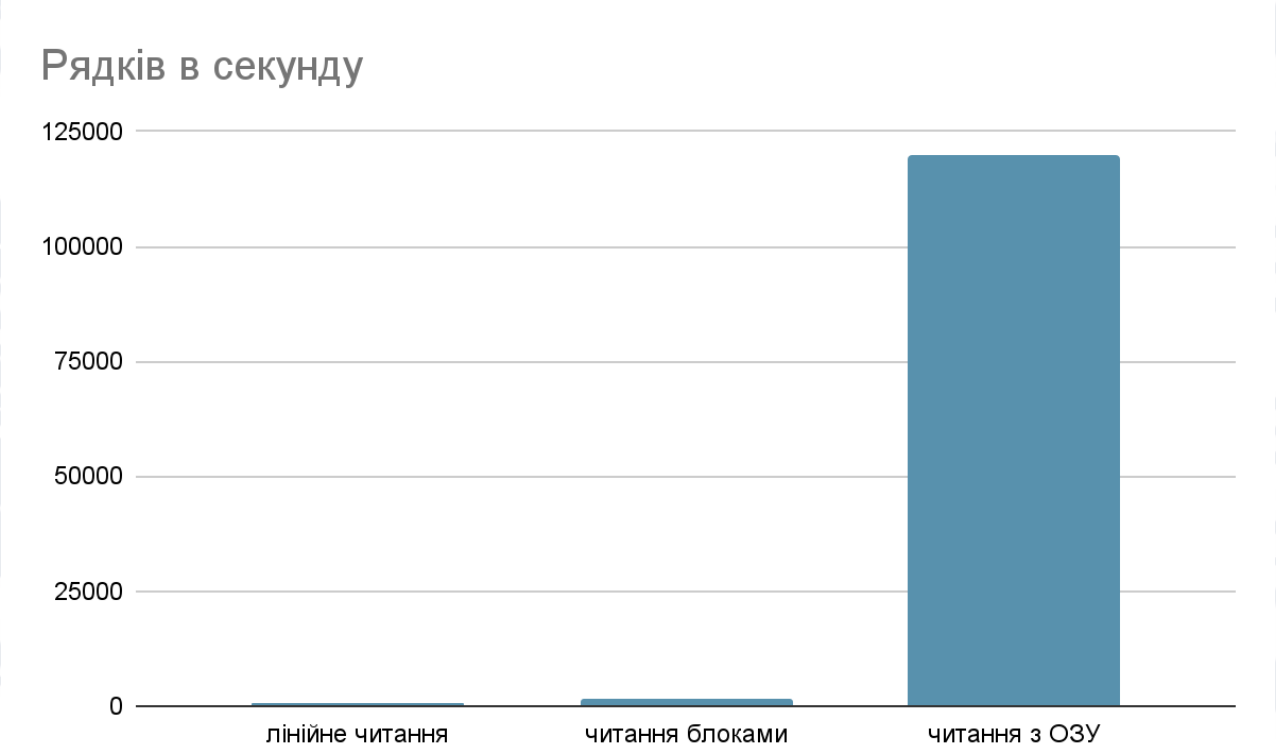

Діаграма 3.3 Читання з оперативної пам'яті.

### 3.2.4 Висновок із тестування

Як висновок із проведених тестів приводжу нижче порівняльну таблицю зі всіма видами читання, які брали участь в експерименті.

Таблиця 3.4 Висновок із тестування

| | Хмарний диск |
|---|---|
| лінійне читання | 500 |
| читання блоками | 1600 |
| читання з ОЗУ | 120000 |




Діаграма 3.4 Висновок із тестування

Таким чином, не зважаючи на те що читання з ОЗУ показало себе найкраще, читання блоками було обране як компромісний варіант між швидкістю та безпекою.

## 3.3 Тестування варіантів  парсингу та форматів логів

Данний підрозділ описує тестування різних варіантів парсингу.

### 3.3.1 парсинг звичайного логу із використанням регулярних виразів

Звичайний підхід, коли програма підлаштовується під формат логів веб-серверу, використовуючи регулярні вирази. Даний підхід використовується у розглянутому аналозі Fail2ban. Даний підхід є найбільш гнучким і простим у реалізації, але разом з тим і найбільш повільним, через те що регулярні вирази не є оптимальними при парсингу тексту структура якого заздалегідь точно відома. для прикладу був взятий звичайний лог веб-серверу.

### 3.3.2 парсинг звичайного логу за роздільником в тексті

Підхід при якому ми використовуємо ключові символи для виокремлення даних у лозі із заздалегідь відомою структурою, даний варіант значно швидше регулярних виразів але все ще не ідеальний, оскільки розгорнутий формат логів потребує додаткових кроків при видобутку інформації.

### 3.3.3 парсинг модифікованого логу із використанням регулярних виразів

Звичайний формат логів містить багато корисної при наладці інформації (шлях запиту, юзер агент користувача, розмір запиту в байтах, і т.д), проте не вся ця інформація корисна при роботі розробленого програмного продукту, Для його роботи потрібен лише час надходження запиту та іп адреса користувача. Тому, доцільно змінити формат логів, аби пришвидшити роботу модуля парсингу. Модифікований лог містить тільки необхідний нам, час надходження та іп адресу.

### 3.3.4 парсинг модифікованого логу з роздільником

Тепер, коли ми маємо мінімізований лог, ми можемо спробувати використати для його парсингу просте розділення за спец символом.

### 3.3.5 Результати тестування

Було проведено тестування усіх наведених вище способів парсингу логів, результати представлені у таблиці та графіку, метод парсингу звичайного логу із використанням регулярних виразів був взятий за 100% при виведенні результатів, як такий, що його необхідно покращити.

легенда:

1 - парсинг звичайного логу із використанням регулярних виразів

2 - парсинг звичайного логу за роздільником в тексті

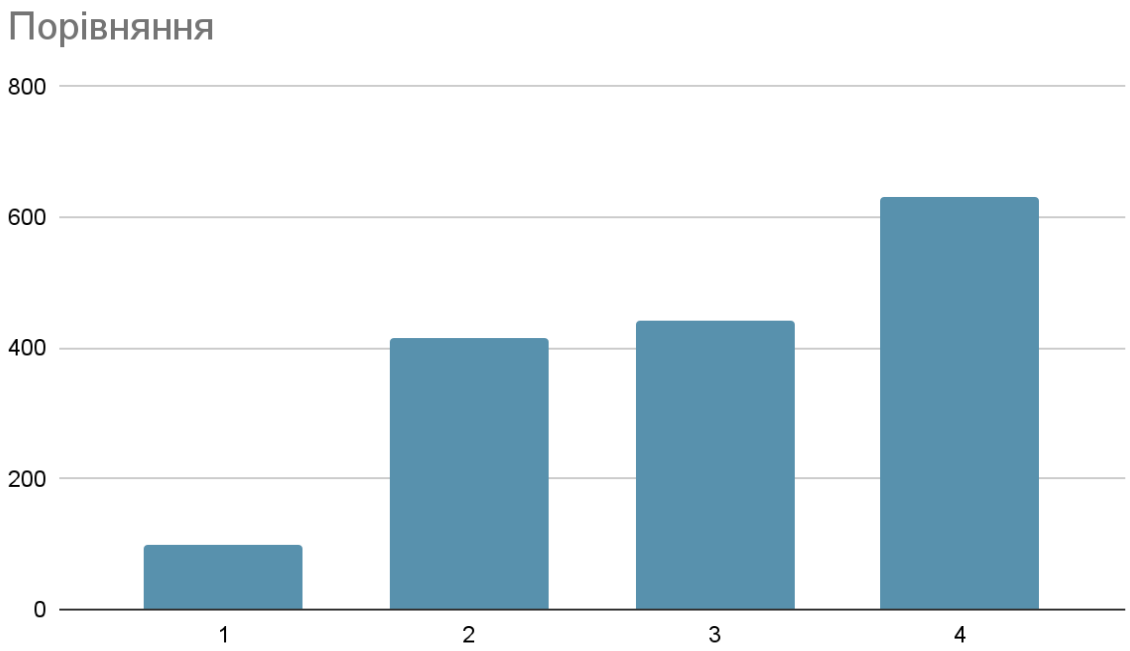
3 - парсинг модифікованого логу із використанням регулярних виразів

4 - парсинг модифікованого логу за роздільником

Таблиця 3.5 Результати тестування

| | продуктивність % |
|---|---|
| 1 | 100 |
| 2 | 416 |
| 3 | 441 |
| 4 | 632 |

Як можна побачити з графіку, зміна формату логів дозволяє полегшити роботу як регулярним виразам(за рахунок зниження кількості варіантів) так і парсингу з роздільником(за рахунок зменшення кількості операцій). Зміна формату логу для веб-сервера нджинкс не є проблемою, та відбувається доволі легко. За результатами тестування було обрано метод парсингу модифікованого логу з роздільником, який показав ефективність вищу більш ніж у 6 разів(в умовах тестованої системи).




Діаграма 3.5 Результати тестування

## 3.4 Пошук способу блокування та розблокування цілей

### 3.4.1 Iptables

iptables — це утиліта для користувача, яка дозволяє системному адміністратору налаштовувати правила фільтрації IP-пакетів брандмауера ядра Linux, реалізованого як різні модулі Netfilter. Фільтри організовано в різних таблицях, які містять ланцюжки правил обробки пакетів мережевого трафіку. Для різних протоколів зараз використовуються різні модулі ядра та програми; iptables застосовується до IPv4, ip6tables до IPv6, arptables до ARP, а ebtables до кадрів Ethernet. Використання даної утиліти дозволить проводити блокування на вищому рівні з урахуванням протоколів, портів і т.д. Утиліта використовує ланцюжки, це дозволить зібрати дані про заблоковані раніше адреси, та дозволить уникнути використання сторонньої бази даних, зменшуючи власне дублювання даних, та роблячи продукт менш залежним і більш портативним.

### 3.4.2 блекхол роут

Спосіб блокування який полягає в додаванні запису в таблиці маршрутизації, який вказує на те що іп адреса яку необхідно заблокувати, нікуди не веде, і таким чином не дає встановити підключення. Менш ресурсозатратний і більш примітивний спосіб блокування, який, проте, не дає можливості зручно отримати інформацію про вже заблоковані адреси, що змушує додавати зовнішню БД до програмного продукту.

### 3.4.3 Висновок

У результаті було обрано метод блокування iptables, як більш зручний, та менш ресурсозатратний, у тому плані що для нього не потрібно створювати окрему БД, це економить час при розробці, та зменшить дублювання даних. Також це дозволить запобігти розсинхронізації даних, та дозволить підтримувати single-source-of-truth (єдине джерело правди), тобто не буде необхідності додатково слідкувати за актуальністю даних у сторонній БД.

## 3.5 Модулі додатку

Додаток складається із трьох структурних модулів:

- Джерело даних, включає в себе функціонал читання файлу з диску, та його парсингом, тобто передає підготовлені дані на вхід програми.
- Модель, модель аналізує дані та на основі проведеного аналізу робить запит до модуля блокування на, власне, блокування адреси
- Модуль блокування, отримує запити від моделі, та проводить блокування адрес

### 3.5.1 Класи програми

- Singleton - клас який гарантує що у нього буде лише один інстанс(екземпляр). Наслідується іншими класами.

- BanMethod - Абстрактний клас, містить визначення методів необхідних для створення методу блокування, дозволить контрольовано розширяти методи блокування власними. Потребує від потомків наявності методів ban і unban.
- IptablesBlockMethod - клас, наслідник класу BanMethod, реалізує функціонал по роботі з інтерфейсом iptables, для звернень до апі використовується бібліотека iptc. даний клас при ініціалізації створює необхідний ланцюжок в таблиці фільтру, та створює правило перенаправлення в основному ланцюжку. Якщо на момент ініціалізації ланцюжок існує, то класс завантажить всі записи с нього до кешу, додавання і видалення з кешу відбуваються синхронно з зверненням до апі, що дозволяє підтримувати їх у синхронному стані. При блокуванні адреси, відправляється запит на додавання правила в iptables, при успішному блокуванні додається запис у кеш. При видаленні відбувається зворотній процес, відправляється запит на видалення правила з ланцюжка і видаляється з кешу. Кеш реалізований за допомогою хеш таблиці, це дозволяє зробити додавання в таблицю та запити на елемент по імені за O(1), тобто миттєво, в термінах інформаційних технологій.
- ChunkReader - Клас, який виконує читання з диску блоками (чанками). Даний клас при ініціалізації може читати вказану кількість ліній з кінця. Також, даний клас слідкує за позицією в файлі, з якої читав востаннє. Клас містить буфер заданого розміру, в якому тримає блок (чанк) даних, як тільки буфер стає порожнім більш ніж на 50%, класс вичитає з файлу новий блок даних. Клас реалізований у вигляді безкінечного ітератора, як тільки файл закінчився ітерація повертає None це дозволяє визначити кінець файлу і прийняти міри.
- SplitParser - Клас, який використовує композицію з попереднім класом (ChunkReader) і виконує парсинг рядків які були зчитані. Клас реалізований у вигляді безкінечного ітератора, як тільки файл закінчився

ітерація ChunkReader повертає None, клас теж поверне None, це дозволяє визначити кінець файлу і прийняти міри.

- Config - Клас динамічної конфігурації. Клас пропонує 3 рівні конфігурації, конфігурація за допомогою змінних оточення, конфігурація шляхом зміни файлу .env, якщо ключ не був знайдений в попередніх шарах, він буде взятий зі значень за замовчуванням, інакше поверне помилку. Пріоритет запиту шарів наступний: змінні оточення, файл env, змінні за замовчуванням. Клас є потомком класу Singleton, аби гарантувати що не буде створено декілька інстансів конфігурації.
- Model - Клас, який використовує композицію із класами IptablesBlockMethod, SplitParser, та Config. Клас реалізує аналіз даних та прийняття рішення щодо блокування користувача, на основі даних отриманих від класу SplitParser. Блокування відбувається шляхом виклику функцій класу IptablesBlockMethod. Клас Config використовується аби реалізувати можливість гнучкого керування і параметризації програмного продукту. Рішення приймаються на основі частоти, з якою певна адреса з'являється у логах за певний період часу. Одиницею гранулярності являється період часу встановлений користувачем.

### 3.5.2 Опис класів програми, їх функцій та даних

- Singleton - не містить даних чи методів
- BanMethod
    - ban - абстрактний метод, необхідний для реалізації потомками способу блокування
    - unban - абстрактний метод, необхідний для реалізації потомками способу розблокування

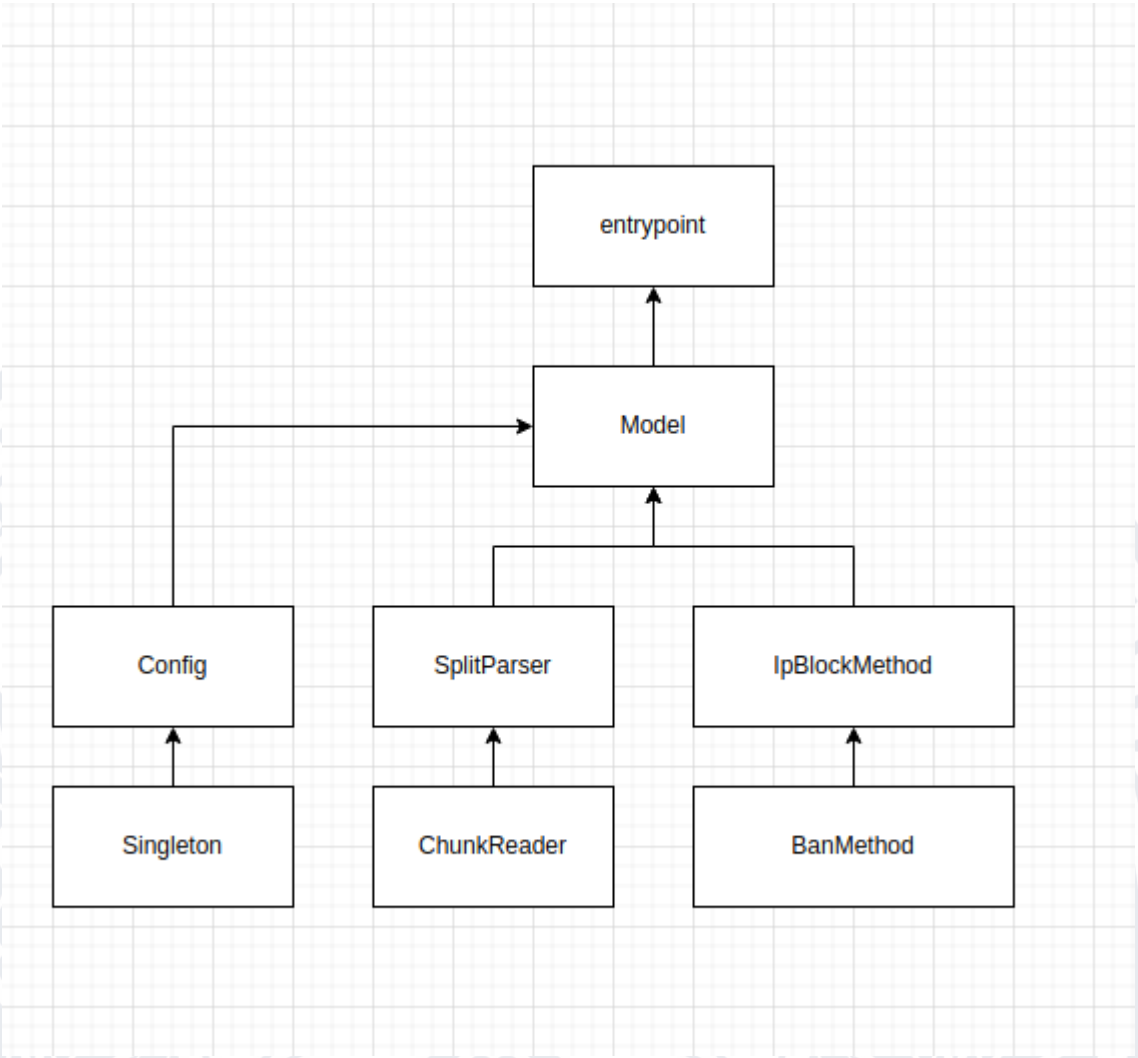



Рис 3.2 Класи програми

- IptablesBlockMethod
  - __init__ - метод ініціалізації. Створює необхідний ланцюжок в таблиці фільтру, та створює правило перенаправлення в основному ланцюжку. Якщо на момент ініціалізації ланцюжок існує, то класс завантажить всі записи с нього до кешу, додавання і видалення з кешу відбуваються синхронно з зверненням до апі, що дозволяє підтримувати їх у синхронному стані.
  - export_chain_rules_to_datastore - Метод експорту даних з існуючого ланцюжка в кеш, при доступі до апі спирається на бібліотеку iptc.
  - add_ip_to_datastore – додає іп в кеш
  - remove_ip_from_datastore - видаляє іп з кешу

- is_ip_in_datastore - перевіряє чи присутній іп в кеші
- ban - Відправляє запит на додавання правила в iptables, при успішному блокуванні додає запис у кеш
- unban - Відправляє запит на видалення правила з iptables ланцюжка, при успішному розблокуванні видаляє запис із кешу.
- flush - очистити ланцюжок із заблокованими адресами, при успішному очищені також очистить кеш.
- cleanup - Метод, що очистить ланцюжки, та вадалить правила і інші об'єкти створені при виклику ініціалізації.
- ip_datastore - хеш-таблиця, являє собою кеш

- ChunkReader
  - __init__ - метод, при ініціалізації перевірить доступність файлу для читання та прочитає n строк з кінця файлу
  - _tail - внутрішній метод для читання рядків файлу з кінця
  - load_chunk - метод який вичитує блок тексту в буфер
  - __iter__ - метод який поверне об'єкт ітератора
  - __next__ - Вбудований метод ітератора, викличе метод next
  - next - метод який поверне наступний об'єкт з буфера (строку), також метод викличе load_chun, якщо буфер вичерпано на 50% і більше. якщо буфер пустий і в файлі немає нових записів, поверне None
  - buffered_lines - змінна буферу рядків
- SplitParser
  - __iter__ - метод який поверне об'єкт ітератора
  - __next__ - метод який, запросить у об'єкта ChunkReader наступну строку, распарсить її і поверне.
- Config
  - __init__ - метод ініціалізації, запросить дані з файлу конфігурації
  - get - метод, який дістає значення, використовуючи 3 рівні конфігурації, конфігурація за допомогою змінних оточення, конфігурація шляхом зміни файлу .env, якщо ключ не був знайдений

в попередніх шарах, він буде взятий зі значень за замовчуванням, інакше поверне помилку. Пріоритет запиту шарів наступний: змінні оточення, файл env, змінні за замовчуванням

- __getitem__ - вбудований метод, дає можливість отримувати значення з конфігу так, якби конфіг був словником (об’єкт[ключ])

- Model
  - __init__ - метод ініціалізації
  - calculate_frequency_and_ban - отримує данні з об’єкту SplitParser, визначає частоту звернень за вказаний період.
  - filter_ips - сканує сховище іп-адрес, банить адреси, які перевищили частоту зазначену в конфігурації
  - frequency_datastore - словник, ключ це іп адреса, значення, кількість знаходжень у лозі за відрізок часу

### 3.5.3 Опис додаткових функцій

В додатку реалізована функція логування подій, яка дозволить виявити некоректну роботу, у разі потреби, та леко відладити помилку, рівень виводу логів регулюється в конфігурації. Можна обрати 5 рівнів виводу логу

- Debug- вивід абсолютно всіх логів в програмному продукті(найбільш детальний рівень)
- Info - вивід логів, що відображають процес роботи програми
- Warning - вивід логів, починаючи з рівня попереджень
- Error - вивід логів починаючи з тих що сигналізують про помилки, які, проте, не завжди призводять до завершення роботи програми
- Critical - вивід тільки найкритичніших повідомлень

Приклад логу:

```
2022-10-14 15:33:13 |DEBUG    | reader: end of file reached
2022-10-14 15:33:13 |DEBUG    | parser: datasource is empty
2022-10-14 15:33:13 |INFO     | __main__: end of log file reached. will wait
0.5 seconds
```

```
2022-10-14 15:33:13 |INFO     | reader: file chunk loaded successfully, end
position is 1583846446
2022-10-14 15:33:13 |DEBUG    | reader: end of file reached
2022-10-14 15:33:13 |DEBUG    | parser: datasource is empty
2022-10-14 15:33:13 |INFO     | __main__: end of log file reached. will wait
0.5 seconds
2022-10-14 15:33:14 |INFO     | reader: file chunk loaded successfully, end
position is 1583846446
2022-10-14 15:33:14 |DEBUG    | reader: end of file reached
2022-10-14 15:33:14 |DEBUG    | parser: datasource is empty
2022-10-14 15:33:14 |INFO     | __main__: end of time sample reached. filtering
results
2022-10-14 15:33:14 |INFO     | __main__: IP 172.17.0.1 due to ban
2022-10-14 15:33:14 |WARNING  | iptables: 172.17.0.1 already in datastore but
tried to ban
2022-10-14 15:33:14 |INFO     | __main__: IP 172.17.0.1 banned
2022-10-14 15:33:14 |INFO     | __main__: end of log file reached. will wait
0.5 seconds
```

### 3.5.4 Методи конфігурування додатку

Додаток може бути конфігурований 2-ма способами, через файл .env, приклад такого файлу ви можете побачити нижче, та через змінні оточення, що може бути зручно при динамічному конфігуруванні, та при використанні контейнерів у статичній (іммутабельній) інфраструктурі.

src > .env

```
1  LOG_LEVEL=DEBUG
2  LOG_FILE=nginx_ddos_protection.log
3
4  SLEEP_ON_EMPTY_LOG_SECONDS=0.5
5  RATELIMIT=300
6  RATELIMIT_PERIOD_SECONDS=1
7
8  SOURCE_LOG_FILE=nginx_test_conf/log/access.log
```

Рис. 3.3 приклад файлу конфігурації

## 3.6 Модель взаємодії

Даний підрозділ описує модель взаємодії компонентів системи між собою, та з користувачем.

### 3.6.1 Модель взаємодії з користувачем

Взаємодія з користувачем відбувається шляхом конфігурування в один бік, та переглядом виводу на екрані в інший, як показано на рисунку 3.6.1.1

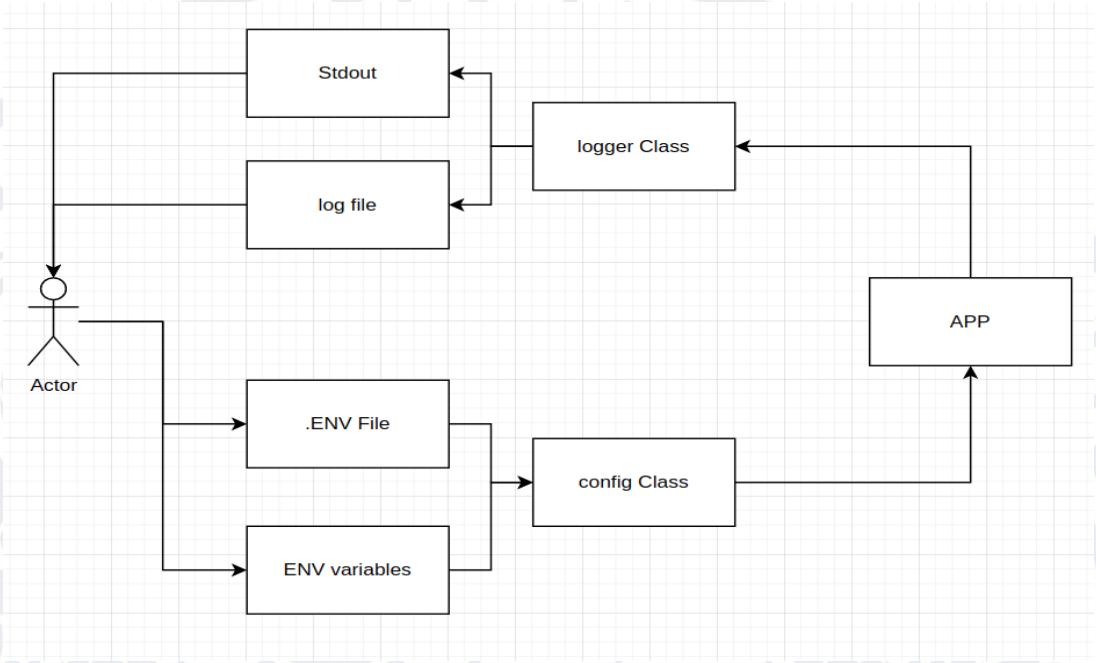



Рисунок 3.4 взаємодія користувача з програмою

### 3.6.2 Модель взаємодії з веб-сервером

Модель взаємодії програми з веб-сервером базується на тому, що веб-сервер створює записи в логах, а програмний продукт їх читає. Веб-сервер не повинен знати і залежати від програмного продукту, це зветься слабкою зв'язністю і підвищує надійність роботи програмного комплексу. Схема такої взаємодії зображена на рисунку 3.6.2.1.

### 3.6.3 Діаграма потоків даних

Діаграма потоку даних відображає більше повну картину, оскільки показує одразу взаємодію програмного продукту, як з користувачем, так із зовнішнім комплексом(веб-сервером). Дана діаграма зображена на рисунку 3.6.3.1

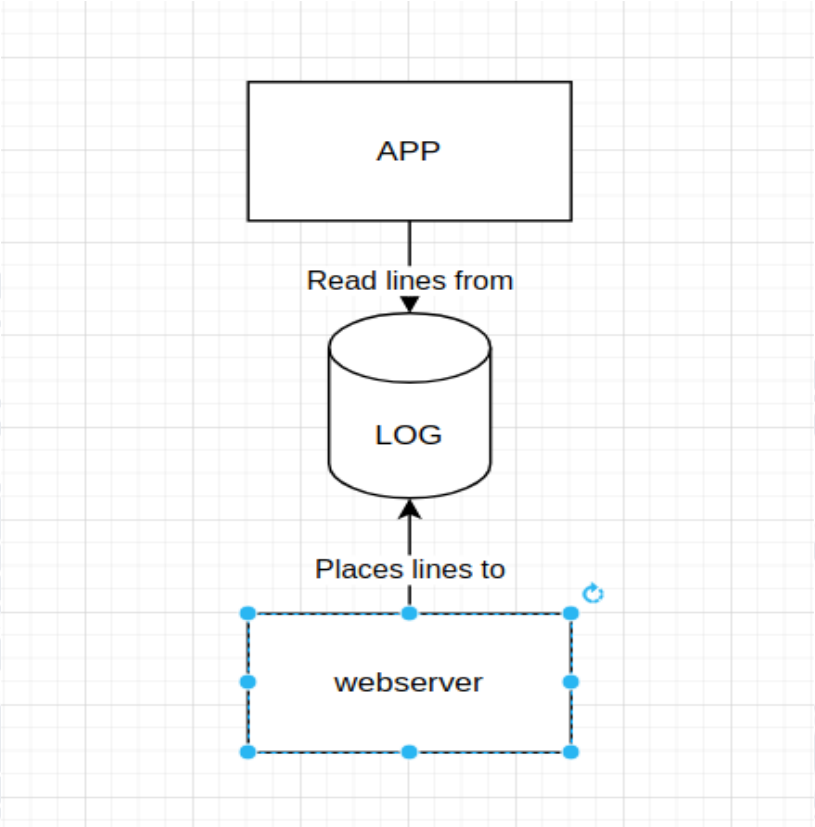



Рисунок 3.5 Модель взаємодії з веб-сервером

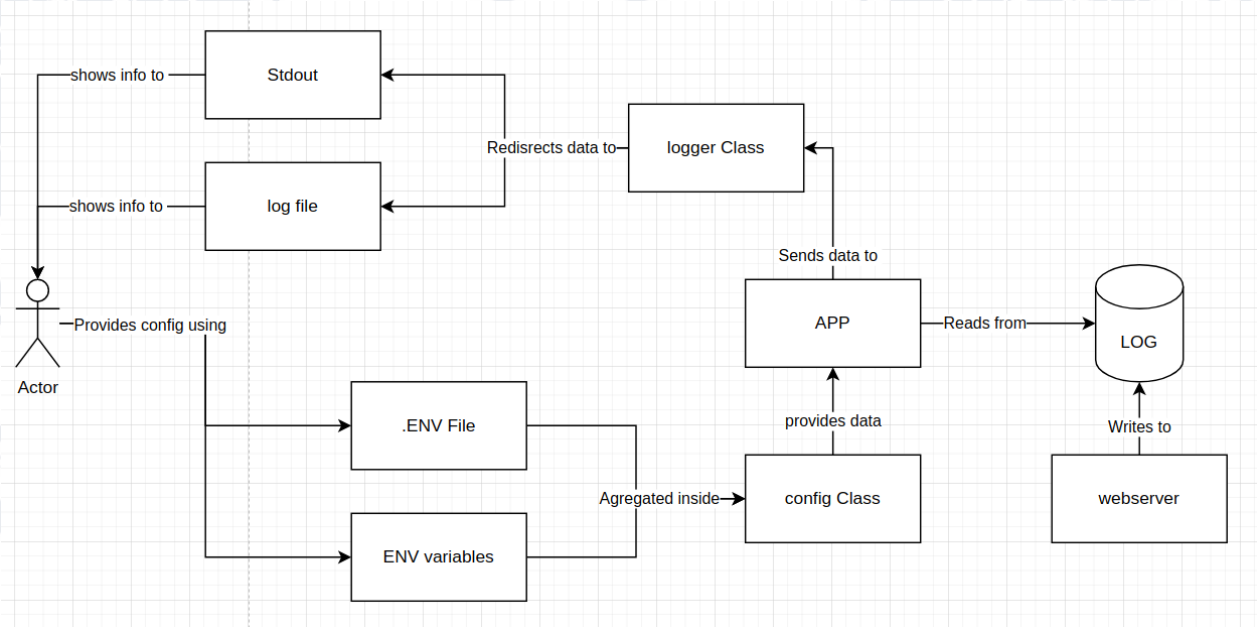



Рисунок 3.6 Діаграма потоків даних

## 3.7 Зміни в конфігурації веб-серверу

### 3.7.1 Тестування різних підходів запису на диск

Окрім ситуації з швидкістю читання на дисках з низьким IOPS, можуть також виникнути проблеми із частим записом на диск. Тому, було проведено дослідження того, як різні методи запису на диск впливають на продуктивність роботи веб-сервера. Для досліду було обрано мінімальну конфігурацію, аби операції читання і обробки контенту мінімально впливали на показники, таким чином ми піднімемо роль запису логів. Необхідно буде зробити декілька опорних тестів, для того аби параметри конкретної системи не впливали на показники. Такими тестами було обрано запис на диск без змін у конфігурації, а взагалі без запису на диск (з вимкненим логом). Тести традиційно проводитимуться на двох системах, хмарному інстансі ABC, та локальній машині. Для того аби створити навантаження на систему буде використано утиліту WRK [54].

### 3.7.2 Тестування запису на диск без змін (хмарний інстанс)

Дане тестування включає в себе навантаження веб-сервера нджинкс, та заміру пропускної спроможності останнього. контентом буде слугувати дефолтна сторінка веб-серверу. Файл конфігурації веб-сервера буде приведено нижче. В наступних тестах буде змінюватись тільки директива access_log, тому, тільки вона і буде включена в наступних тестах.

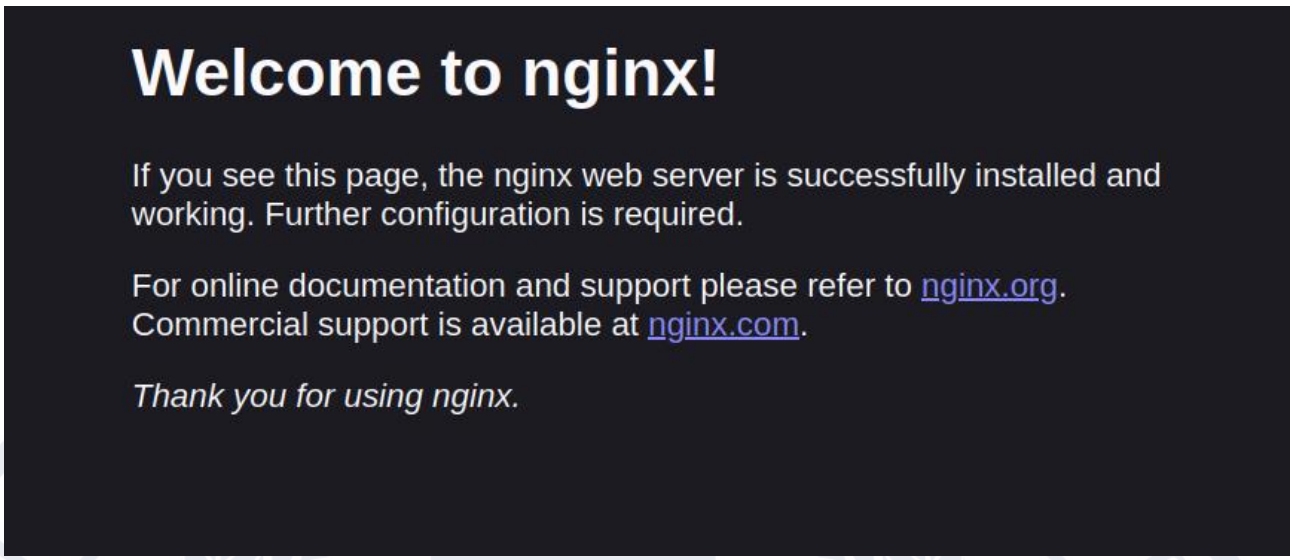



Рисунок 3.7 дефолтна сторінка веб-серверу

Конфігурація веб-сервера:

```
“log_format short '$remote_addr|$msec';
server {
        listen          80;
        listen  [::]:80;
        server_name  localhost;
        access_log  /etc/nginx/conf.d/log/access.log short;
        location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
        }
}”
```

Умови досліду для піддослідної машини:

- aws ebs ssd 3000 iops storage
- 1 ядро процесора (intel xeon)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80

- кількість одночасних підключень 700

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 6650 запитів у секунду.

Примітка: було використано формат логу, узгоджений в попередніх розділах.

### 3.7.3 Тестування пропускної спроможності веб-серверу із вимкненим логуванням. (хмарний інстанс)

Після того, як було встановлено “нижню планку”, тобто базовий рівень продуктивності, необхідно провести досліди, та встановити верхню границю для нашої системи, таким чином ми матимемо на що орієнтуватись в наступних дослідах.

Умови досліду для піддослідної машини:

- диск не використовується
- 1 ядро процесора (intel xeon)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log off;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому:

- 10500 запитів у секунду.

Як можна побачити відключення логу має певний ненульовий вплив на продуктивність системи. Тобто вплив параметрів диска на продуктивність не нульовий, і стає більш суттєвим при зменшенні складності логічної обробки даних веб-сервером.

### 3.7.4 Тестування пропускної спроможності веб-серверу із логуванням у стандартний вивід (хмарний інстанс)

Оскільки технологія контейнеризації в даний момент стрімко набирає оберти, видається доцільним перевірити і даний варіант роботи веб-серверу в контексті виводу логу. За правилами “програми 12 факторів” [55] весь вивід логів повинен відбуватись у стандартний вивід (/dev/stdout), та у потік виводу помилок (/dev/stderr).

Умови досліду для піддослідної машини:

- Вивід у стандартний потік виводу із показом на дисплеї
- 1 ядро процесора (intel xeon)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log /dev/stdout;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 1900 запитів у секунду.

Як видно з результатів, вивід у потік стандартного виводу значно сповільнює роботу веб-серверу. В основному це пов'язано з тим, що дані одразу виводяться на екран, тобто відбувається блокуючий запис. Під час експлуатації вивід не йде одразу на дисплей, а обробляється демоном ядра контейнеризації.

### 3.7.5 Тестування пропускної спроможності веб-серверу із логуванням у стандартний вивід без показу на дисплеї (хмарний інстанс)

Як було казано в результатах попереднього тестування (розділ 3.7.4), під час експлуатації вивід не йде відразу на дисплей, а обробляється

демоном ядра контейнеризації. Тому було вирішено провести тест більш приближений до умов експлуатації.

Умови досліду для піддослідної машини:

- Вивід у стандартний потік виводу без показу на екрані
- 1 ядро процесора (intel xeon)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log /dev/stdout;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 3400 запитів у секунду.

Як видно із результатів тесту, запуск в такому режимі приносить додатково до 40% продуктивності.

### 3.7.6 Тестування пропускної спроможності веб-серверу із логуванням у оперативну пам’ять (хмарний інстанс)

Оскільки серед методів читання логу був присутній варіант із читанням логів із оперативної пам’яті, необхідно “навчити” веб-сервер записувати їх туди, та протестувати характеристики швидкості такого запису. На лінукс та юнікс машинах, за замовчуванням є можливість працювати з пам'яттю, як з диском. У таких системах частина ОЗУ підмонтовується як диск за шляхом /dev/shm. Це дає нам змогу перевірити як відбуватиметься обробка, при читанні напряму з оперативної пам’яті. Також є спосіб записувати логи напряму в пам’ять, без використання примонтованих віртуальних директорій, але він потребує значних змін у веб-сервері, тому був відкинутий в даній роботі. Як відомо, оперативна

пам'ять має вражаючу швидкість запису/читання та швидкість доступу до даних.

Умови досліду для піддослідної машини:

- Вивід у файл в оперативній пам'яті
- 1 ядро процесора (intel xeon)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log /dev/shm;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 9600 запитів у секунду.

Як видно, запис у оперативну пам'ять є майже настільки ж ефективним, як і відсутність логу взагалі (90% ефективності).

### 3.7.7 Тестування пропускної спроможності веб-серверу із логуванням у файл в режимі запису блоками (хмарний інстанс)

Після вивчення документації хмарного веб-сервера було знайдено спосіб запису логів у файл блоками, тобто, спочатку веб сервер накопичує певну кількість логів у пам'яті, і тільки після цього записує їх в файл.

Умови досліду для піддослідної машини:

- aws ebs ssd 3000 iops storage (запис блоками)
- розмір чанку 64 кб
- 1 ядро процесора (intel xeon)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт

- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log  /etc/nginx/conf.d/log/access.log short buffer=64k;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 10400 запитів у секунду.

Як видно із результатів, запис блоками майже не чинить впливу на продуктивність веб-серверу, все через те, що відпадає необхідність у постійній блокуючій операції, а саме запису на диск для кожного рядка. Запис блоків відбувається асинхронно, тому не впливає на роботу веб-сервера.

### 3.7.8 Тестування запису на диск без змін (локальна машина)

Для більшої вибірки даних, і відповідно більш достовірних результатів, тестування аналогічне розділам 3.7.2-3.7.7 було проведено на локальній машині. Тестування  у розділах 3.7.8-3.7.13 буде повторювати умови тестування із розділів 3.7.2-3.7.7, за винятком ресурсів, тестованої машини, у якої більш швидкий диск, та потужніший процесор.

Умови досліду для піддослідної машини:

- local ebs ssd 500000 iops storage
- 6 ядер процесора (intel core i7)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log /etc/nginx/conf.d/log/access.log short;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 88000 запитів у секунду.

Даний приріст в основному зумовлений різницею в характеристиках систем, кінцевий результат необхідно рахувати у відсотках в рамках прогонів систем з однаковими характеристиками.

### 3.7.9 Тестування пропускної спроможності веб-серверу із вимкненим логуванням. (локальна машина)

Умови досліду для піддослідної машини:

- диск не використовується
- 6 ядер процесора (intel core i7)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log  off;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 115000 запитів у секунду.

Як і в тестуваннях з хмарною системою, даний варіант показує потужний приріст.

### 3.7.10 Тестування пропускної спроможності веб-серверу із логуванням у стандартний вивід (локальна машина)

Умови досліду для піддослідної машини:

- вивід у стандартний потік виводу з показом на дисплеї
- 6 ядер процесора (intel core i7)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log  /dev/stdout;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 46000 запитів у секунду. Як і в тестуваннях з хмарною системою, даний варіант показує падіння пропускної спроможності.

### 3.7.11 Тестування пропускної спроможності веб-серверу з логуванням у стандартний вивід без показу на дисплеї (локальна машина)

Умови досліду для піддослідної машини:

- вивід у стандартний потік виводу без показу на дисплеї
- 6 ядер процесора (intel core i7)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log  /dev/stdout;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 54000 запитів у секунду.

Як і в тестуваннях з хмарною системою, даний варіант показує приріст, при вимкненні показу логів на дисплей, та узгоджується з результатами відповідного тестування.

### 3.7.12 Тестування пропускної спроможності веб-серверу з логуванням у оперативну пам’ять (локальна машина)

Умови досліду для піддослідної машини:

- файл в оперативній пам’яті
- 6 ядер процесора (intel core i7)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log /dev/shm;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 97000 запитів у секунду.

Як і в тестуваннях з хмарною системою, даний варіант показує потужний приріст, та узгоджується з результатами відповідного тестування.

### 3.7.13 Тестування пропускної спроможності веб-серверу із логуванням у файл в режимі запису блоками (локальна машина)

Умови досліду для піддослідної машини:

- запис на диск блоками
- розмір блоку 64 кб
- 6 ядер процесора (intel core i7)
- контент являє собою сторінку за замовчуванням

Умови для машини, яка здійснює запити:

- 6 ядер процесора
- ширина каналу 1 гігабіт
- кількість потоків 80
- кількість одночасних підключень 700

Змінена директива запису логів: “access_log  /etc/nginx/conf.d/log/access.log short buffer=64k;”

За результатами 10 замірів було отримано результат із наступною кількістю запитів у секунду в середньому 110000 запитів у секунду.

Виглядає неймовірним, що даний замір показує результат запису, більший ніж результати запису в оперативну пам’ять, та, як уже було сказано справа в блокуючому читанні, тобто для кожного рядка логу відбувається запис, а при буферизації логи накопичуються. При використанні формату логу узгодженого в попередніх розділах, та розміру буферу в 64 кб, запис відбувається в кілька тисяч разів рідше, ніж, при відсутності буферу.

### 3.7.14 Висновки з тестування та агреговані результати

Нижче представлені таблиці з результатами тестів, у відсотковому відношенні, за 100% було взято результати запису на диск без змін. В даних тестах, також, видно, що чим менша потужність системи. тим більш явний вплив режиму запису на продуктивність.

Таблиця 3.6 порівняння способів запису на диск

| Тип запису | Хмарна машина | Локальна система |
|---|---|---|
| запис без змін | 100,00% | 100,00% |
| запис зі вимкненими логами | 157,00% | 130,00% |
| запис у стандартний вивід із показом | 28,00% | 52,00% |
| запис у стандартний вивід без показу | 51,00% | 61,00% |
| запис у файл в оперативній пам'яті | 138,00% | 110,00% |
| запис у файл блоками | 156,00% | 125,00% |


Діаграма 3.6 порівняння способів запису на диск

Як можна бачити із результатів тестування, запис на диск блоками є набагато більш ефективним, ніж, запис без буферизації, і навіть ніж запис у оперативну пам’ять.

### 3.8 Тестування програми та заміри

Для перевірки роботи програми, було обрано примітивний сценарій з двома атакуючими машинами із адресами 172.17.0.1 та 172.17.0.1, вони являють собою віртуальні машини, встановлені на тій же фізичній машині що сервіс.

Сценарій відповідає типовому в такій ситуації, тобто, ми маємо певний об’єм логів, який був накопичений до запуску даного програмного продукту, а також ми маємо ситуацію, коли користувач проводить атаку під час роботи продукту та веб-сервера. Далі буде приведено текстовий лог програми і дані пояснення до нього, часові відмітки було прибрано для зручності читання.

```
|DEBUG   | iptables: got FILTER table
|INFO    | iptables: custom chain already exists
|DEBUG   | iptables: exported rules to in-memory datastore
```

```
|INFO     | reader: 100000 tailed successfully
|INFO     | reader: succesfully opened file and tail 100000 lines
|INFO     | __main__: model initialized
```

Тут ми можемо бачити початок ініціалізації програми. було отримано інформацію із-іптейблз, перевірено та завантажено лог-файл.

```
|DEBUG    | __main__: IP 172.17.0.1 added to datastore first time
|INFO     | __main__: end of time sample reached. filtering results
|DEBUG    | __main__: IP 172.17.0.2 added to datastore first time
|INFO     | __main__: end of time sample reached. filtering results
|INFO     | __main__: IP 172.17.0.1 due to ban
|DEBUG    | iptables: 172.17.0.1 added to datastore
|INFO     | iptables: 172.17.0.1 banned by iptables
|INFO     | __main__: IP 172.17.0.1 banned
|DEBUG    | __main__: IP 172.17.0.2 added to datastore first time
```

Тут видно процесс обробки завантажених із файлу рядків, які були створені веб-сервером до того як программа була запущена. За сценарієм 172.17.0.1 почав атаку ще до того як аналізатор було запущено, а 172.17.0.2 емулює звичайні неагресивні запити. Як видно з логу, 172.17.0.1 було заблоковано ще в першому проході, а 172.17.0.2 хоч і був замічений в обох проходах, не був заблокований, оскільки не проводив агресивних запитів.

```
|INFO     | reader: file chunk loaded successfully, end position is 1583846446
|DEBUG    | reader: end of file reached
|DEBUG    | parser: datasource is empty
|INFO     | __main__: end of log file reached. will wait 0.5 seconds
|INFO     | reader: file chunk loaded successfully, end position is 1583846446
|DEBUG    | reader: end of file reached
|DEBUG    | parser: datasource is empty
|INFO     | __main__: end of log file reached. will wait 0.5 seconds
|INFO     | reader: file chunk loaded successfully, end position is 1583846446
|INFO     | __main__: end of time sample reached. filtering results
|INFO     | __main__: IP 172.17.0.2 due to ban
|INFO     | __main__: IP 172.17.0.2 banned
|INFO     | __main__: end of log file reached. will wait 0.5 seconds
```

Тут ми можемо побачити, що певний час не відбувалось запитів і програма повідомляє що при завантаженні чанка даних не отримала нових рядків. Після чого, як і заплановано за сценарієм, проходить сплеск запитів, при останньому

зчитуванні аналізатор отримує ці рядки і за результатами обробки блокує 172.17.0.2.

### 3.9 Майбутній розвиток програми

Серед шляхів розвитку програми можна виділити два основних напрямки, а саме, вдосконалення уже реалізованого функціоналу, та впровадження нового.

Серед аспектів для вдосконалення можна виділити наступні:

- Удосконалити систему підрахунку частоти - на даний момент підрахунок відбувається у кінці кожного циклу рівного періоду обчислення, що зветься гранулярністю, тобто мінімальним періодом, за який програма може виконати дії. Даний процес можна реалізувати краще, якщо перевіряти кількість входжень в датасет постійно, а не в кінці періоду обчислення. Так наприклад, якщо встановлено поріг у 100 запитів на хвилину на одну адресу, а користувач зробив уже 1000, його буде заблоковано тільки по завершенню періоду обчислення.
- Додати багатопотоковість - різні модулі програми можна запускати у різних потоках центрального процесора, аби покращити паралельність, та утилізацію ресурсів.
- Змінити мову програмування - незважаючи на те що буде втрачено гнучкість, реалізація продукту на іншій, компільованій мові програмування може додати ще трохи продуктивності.

Серед аспектів, які можна додатково реалізувати, можна виділити наступні:

- Додати класи для блокування - це дозволить використати інші методи блокування.
- Додати класи парсера - це дозволить працювати з іншими форматами логів.

# ВИСНОВОК

У даній роботі було проведено низку досліджень та експериментів, того, як програмні комплекси взаємодіють з апаратним забезпеченням. Дослідження того, як апаратні характеристики системи впливають продуктивність програмного коду, а також, як конфігурація програмної системи дозволяє краще розкрити потенціал апаратної системи. На основі дослідження були виявлені взаємозв'язки того, як дискова підсистема впливає на швидкість роботи та аналізу даних, також проаналізовано те як різні методі запису та читання можуть впливати на продуктивність одного і того ж апаратного забезпечення. Було розроблено ефективній в роботі та легкий у впровадженні спосіб запису даних на диск веб-сервером, який при цьому є безпечним у плані перевантаження ОЗУ. Також було застосовано результати досліджень для розробки та реалізації алгоритму читання даних з диску, це дозволило оптимізувати програмне забезпечення для аналізу. Була спроектована та реалізована програмна система, яка дозволяє захистити сервер від атак на перевантаження мережевої шини. Керуючись результатами тестування було внесено оптимізацію, яка вигідно виділяє продукт серед аналогів, хоча і дещо втрачає в гнучкості. Код програмного продукту уже доступний для використання та доповнення в GitHub репозиторії за адресою

https://github.com/RuStyC0der/nginx_ddos_protection.git

# СПИСОК ЛІТЕРАТУРИ


1. IOPS URL: https://habr.com/ru/post/164325/
2. IOPS URL: https://ru.wikipedia.org/wiki/IOPS
3. IOPS URL: https://aws.amazon.com/ebs/provisioned-iops/
4. IOPS URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html
5. IOPS URL: https://aws.amazon.com/ru/ebs/pricing/
6. IOPS URL: https://www.techopedia.com/definition/2709/inputoutput-operations-per-second-iops
7. IOPS URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-io-characteristics.html
8. AWS URL: https://uk.wikipedia.org/wiki/Amazon_Web_Services
9. AWS URL: https://aws.amazon.com/ru/
10. AWS URL: https://www.techopedia.com/definition/26426/amazon-web-services-aws
11. Хмарний провайдер URL: https://www.techtarget.com/searchitchannel/definition/cloud-service-provider-cloud-provider
12. Хмарний провайдер URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-cloud-provider/
13. Хмарний провайдер URL: https://www.techopedia.com/definition/133/cloud-provider
14. Веб-сервер URL: https://www.techtarget.com/whatis/definition/Web-server
15. Веб-сервер URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server
16. Веб-сервер URL: https://economictimes.indiatimes.com/definition/web-server
17. Веб-сервер URL: https://en.wikipedia.org/wiki/Web_server
18. Веб-сервер URL: https://www.tutorialspoint.com/internet_technologies/web_servers.htm
19. Нджинкс URL: https://www.nginx.com/resources/glossary/nginx/

20. Нджинкс URL: https://kinsta.com/knowledgebase/what-is-nginx/
21. Нджинкс URL: https://en.wikipedia.org/wiki/Nginx
22. Нджинкс URL: https://nginx.org/en/
23. Нджинкс URL: https://www.hostinger.com/tutorials/what-is-nginx
24. Нджинкс URL: https://medium.com/geekculture/what-is-nginx-2edfdad3722b
25. ДДос URL: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/
26. ДДос URL: https://www.kaspersky.com/resource-center/threats/ddos-attacks
27. ДДос URL: https://www.imperva.com/learn/ddos/denial-of-service/
28. ДДос URL: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-ddos/
29. ДДос URL: https://en.wikipedia.org/wiki/Denial-of-service_attack
30. ДДос URL: https://www.comptia.org/content/guides/what-is-a-ddos-attack-how-it-works
31. Фаєрвол додатків URL: https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/
32. Фаєрвол додатків URL: https://www.cloudflare.com/waf/
33. Фаєрвол додатків URL: https://en.wikipedia.org/wiki/Web_application_firewall
34. Фаєрвол додатків URL: https://www.f5.com/services/resources/glossary/web-application-firewall
35. Фаєрвол додатків URL: https://www.imperva.com/learn/application-security/what-is-web-application-firewall-waf/
36. Фаєрвол додатків URL: https://www.techtarget.com/searchsecurity/definition/Web-application-firewall-WAF
37. Фаєрвол URL: https://www.forcepoint.com/cyber-edu/firewall
38. Фаєрвол URL: https://www.checkpoint.com/cyber-hub/network-security/what-is-firewall/

39. Фаєрвол URL: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html
40. Фаєрвол URL: https://www.kaspersky.com/resource-center/definitions/firewall
41. Фаєрвол URL: https://www.techtarget.com/searchsecurity/definition/firewall
42. Фаєрвол URL: https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-firewall
43. Фаєрвол URL: https://en.wikipedia.org/wiki/Firewall_(computing)
44. Fail2ban URL: https://www.fail2ban.org/wiki/index.php/Main_Page
45. Fail2ban URL: https://en.wikipedia.org/wiki/Fail2ban
46. Fail2ban URL: https://www.digitalocean.com/community/tutorials/how-fail2ban-works-to-protect-services-on-a-linux-server
47. Fail2ban URL: https://www.hostinger.com/tutorials/fail2ban-configuration
48. CSF&LFD URL: https://raymii.org/s/articles/Configserver_Firewall_and_Security_CSF_LFD.html
49. CSF&LFD URL: https://configserver.com/configserver-security-and-firewall/
50. Eric C, Rich G. Distributed Denial of Service 2021. 452 с.
51. Sathivel A. DDOS ATTACKS: PREPARATION-DETECTION-MITIGATION 2022. 221 с.
52. Владстон Ф. Ф. Теоретичний мінімум по Computer Science 2018. 472 с.
53. А.Бхаргава. Грокаємо алгоритми 2019. 354 с.
54. WRK URL: https://github.com/wg/wrk
55. Програма 12 факторів URL: https://12factor.net/uk/
56. Leonardo G. First-class objects in Python: Higher-order functions, wrappers, and factories 2018. 321 с.
57. Oswald C. Python Tools for Data Scientists: Pocket Primer 2019. 842 с.
58. Gaurav S., Vitalii S. Technical Building Blocks: A Technology Reference for Real-world Product Development 2018. 142 с.

59. Sean P., Karl M. Docker: Up & Running: Shipping Reliable Containers in Production, 3rd Edition (Early Release) 2016. 329 с.
60. Robert S., Austin B. The Python Apprentice (2022) 2018. 255 с.
61. Alejandro G. Data Structures and Algorithms for Job Interviews: Prep for the interview and get the job you want 2017. 300 с.
62. Prem K., Jon L. Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development, 3rd Edition (Final Release) 2015. 734 с.
63. Harry Y. Python Mini Reference 2022: A Quick Guide to the Modern Python Programming Language for Busy Coders 2015. 791 с.
64. Dimitrios X., Christos M., Ourania K. Handbook of Computer Programming with Python 2013. 908 с.
65. Andreas W., Michael W. Amazon Web Services in Action, Third Edition: An in-depth guide to AWS (MEAP v7) 2012. 285 с.
66. Johannes E., Peter K. Python 3: The Comprehensive Guide to Hands-On Python Programming 2014. 558 с.
67. Mamta M., Gopi B., Bhimavarapu U., Lalit M. Text Analysis with Python: A Research Oriented Guide 2012. 175 с.
68. Дэн Б., Дэвид Э., Флетчер Х. Знакомство с Python 2013. 558 с.
69. Raghunathan R., Resmi S. Data Science for Engineers 2014. 921 с.
70. Derek F. Application Security Program Handbook: A guide for software engineers and team leaders (Final Release) 2017. 410 с.
71. Alex M., Anna M., Steve H. Python in a Nutshell: A Desktop Quick Reference, 4th Edition (6th Early Release) 2013. 478 с.
72. Abhinandan B. Edge Computing with Python: End-to-end Edge Applications, Python Tools and Techniques, Edge Architectures, and AI Benefits 2017. 347 с.
73. Habib I. Problems on Algorithms: A Comprehensive Exercise Book for Students in Software Engineering 2015. 534 с.
74. Anna S. Learning Git: A Visual Approach for Using Git Successfully (4th Early Release) 2016. 412 с.

75. Juntao Q. 10 Refactorings to Boost your Clean Code Efficiency: Small steps to make your code more readable 2018. 201 с.
76. Craig B. Docker for Web Developers: A concise, practical, and easy-to-follow guide 2015. 848 с.
77. Simon B. Software Architecture for Developers: Technical leadership and the balance with agility 2019. 549 с.
78. Simon B. Nordic APIs. Developer Experience: Top strategies to improve the developer experience of your API 2014. 683 с.
79. Rahul A. Advanced Python Tips: A Simple Book on Advanced Python concepts 2013. 393 с.
80. Hershel C. How To Speed Up Slow Python Code With Concurrent Programming And The Cutting-edge Asyncio Library 2018. 321 с.
81. Антонов Ю.С., Римар П.В., Антонова О.Г. Проблема DoS/DDoS атак навчальних ресурсів студентами. Сучасний захист інформації. 2019. No 4(40). С. 52-62
82. А.О. Олійник, Ю.С. Антонов. Програмний аналіз журналів веб серверів Apache з метою запобігання мережевим атакам. Матеріали другої всеукраїнської наукової конференції Комп'ютерні технології обробки даних https://jktod.donnu.edu.ua/article/view/11715

# ДОДАТКИ

Додаток А

клас Singleton

```
class Singleton(object):
    __obj = False  # Private class variable.

    def __new__(cls, *args, **kwargs):
    if cls.__obj:
    return cls.__obj
    cls.__obj = super(Singleton, cls).__new__(cls)
    return cls.__obj
```

клас IptablesBlockMethod

```
from abstract.abstractBlock import BanMethod
from helpers import get_logger

import iptc

logger = get_logger(__name__)

class IptablesBlockMethod(BanMethod):
    """
    class that implemets iptables ip blocking logic
    """
    custom_chain_name = 'nginx-ddos-protection-chain'
    ip_datastore = set()

    def __init__(self):
```

```
        """
        on init we createing new empty chain inside "INPUT" table to hold all blocked users separately
        then we redirect all connections to custom chain
        init is an idempotent method
        """
        table = iptc.Table(iptc.Table.FILTER)
        logger.debug('got FILTER table')

        # if chain exists bound it and load ips from it
        for chain in table.chains:
        if chain.name == self.custom_chain_name:
                logger.info('custom chain already exists')
                self.custom_chain = chain
                self.export_chain_rules_to_datastore()
                return

        logger.info('create and insert custom chain')

        # create and configure chain
        self.custom_chain = table.create_chain(self.custom_chain_name)
        custom_chain_return_rule = iptc.Rule()
        custom_chain_return_rule.create_target("RETURN")
        self.custom_chain.append_rule(custom_chain_return_rule)

        logger.debug('RETURN rule inserted to custom chain')

        # add link to chain "input"
        input_chain = iptc.Chain(table, "INPUT")
        custom_chain_redirect_rule = iptc.Rule()
```

```
custom_chain_redirect_rule.create_target(self.custom_chain_name)
input_chain.insert_rule(custom_chain_redirect_rule)

logger.debug('created and inserted custom chain')


def export_chain_rules_to_datastore(self):
for rule in self.custom_chain.rules:
self.ip_datastore.add(rule.src)
logger.debug('exported rules to in-memory datastore')

def add_ip_to_datastore(self, ip):
self.ip_datastore.add(ip)
logger.debug(f"{ip} added to datastore")

def remove_ip_from_datastore(self, ip):
self.ip_datastore.discard(ip)
logger.debug(f"{ip} femoved from datastore")

def is_ip_in_datastore(self, ip):
return ip in self.ip_datastore


def ban(self, ip):
"""
add drop rule with *ip*
"""
if not self.is_ip_in_datastore(ip):
rule = iptc.Rule()
```

```
rule.src = ip
rule.create_target("DROP")
rule.final_check()
self.custom_chain.append_rule(rule)

self.add_ip_to_datastore(ip)
logger.info(f"{ip} banned by iptables")
else:
logger.warning(f"{ip} already in datastore but tried to ban")


def unban(self, ip):
"""
delete drop rule with *ip*
"""
if self.is_ip_in_datastore(ip):
rule = iptc.Rule()
rule.src = ip
rule.create_target("DROP")
rule.final_check()
self.custom_chain.delete_rule(rule)

self.remove_ip_from_datastore(ip)
logger.info(f"{ip} unbaned by iptables")
else:
logger.warning(f"{ip} not in datastore but tried to unban")

def flush(self):
"""
clean blocked users chain
```

```
        """
        self.custom_chain.flush()
        logger.debug(f"chain '{self.custom_chain_name}' flushed")

        def cleanup(self):
        """
        remove all chains and rules created by class
        """
        input_chain = iptc.Chain(iptc.Table(iptc.Table.FILTER), "INPUT")
        custom_chain_rule = iptc.Rule()
        custom_chain_rule.create_target(self.custom_chain_name)
        input_chain.delete_rule(custom_chain_rule)

        self.flush()
        self.custom_chain.delete()

        logger.info("all resources were cleaned up")
```

допоміжна функція  для логування

```
import logging
import re
from dotenv import dotenv_values
from os import environ
from abstract.singleton import Singleton
from sys import stdout, stderr



def get_logger(name):
```

```
    config = Config()

    # Create a custom logger
    logger = logging.getLogger(name)

    logger.setLevel(str2LogLevel(config['LOG_LEVEL']))

    # Create handlers
    stdout_handler = logging.StreamHandler(stdout)
    file_handler = logging.FileHandler(config['LOG_FILE'])

    stdout_handler.setLevel(str2LogLevel(config['LOG_LEVEL']))
    file_handler.setLevel(str2LogLevel(config['LOG_LEVEL']))

    # Create formatters and add it to handlers
    formatter = logging.Formatter('{asctime} |{levelname:<8}| {name}:
{message}', datefmt='%Y-%m-%d %H:%M:%S', style="{")

    stdout_handler.setFormatter(formatter)
    file_handler.setFormatter(formatter)

    # Add handlers to the logger
    logger.addHandler(stdout_handler)
    logger.addHandler(file_handler)

    return logger

def str2LogLevel(value):
    try:
        if isinstance(value, int) or value.isdigit():
```

```
            ll = int(value)
            else:
            ll = getattr(logging, value.upper())
        except AttributeError:
            raise ValueError(f"Invalid log level {value}")
        return ll


клас Config
class Config(Singleton):

        _defaults_ = {
        "LOG_LEVEL":"INFO",
        "LOG_FILE":"nginx_ddos_protection.log",

        "SLEEP_ON_EMPTY_LOG_SECONDS":"0.5",
        "RATELIMIT":"300",
        "RATELIMIT_PERIOD_SECONDS":"60",

        "SOURCE_LOG_FILE":"/var/log/nginx/access.log",
        }

        def __init__(self):
        self.dotenv = dotenv_values()


        def get(self, key):
        """
        get value in priority:
        environment
```

```
        .env file
        defaults
        """
        value = environ.get(key)
        if value:
        return value

        value = self.dotenv.get(key, None)
        if value:
        return value

        value = self._defaults_.get(key, None)
        if value:
        return value

        raise KeyError(f"Key '{key}' does not exist in config")


        def __getitem__(self, key):
        return self.get(key)


клас SplitParser

logger = get_logger(__name__)

class SplitParser():
        """
        this parser gets string like <ip>|<timestamp><new line>
        and returns list like [ip, timestamp]
```

```
        """
        separator = "|"

        def __init__(self, datasource):
        """
        datasource must be iterable
        """

        self._datasource = datasource

        def __iter__(self):
        return self

        def __next__(self):
        line = self._datasource.next()
        if not line:
        logger.debug(f"datasource is empty")
        return None

        parsed_line = line.strip().split(self.separator)
        return parsed_line
клас ChunkReader

from time import sleep
import os

logger = get_logger(__name__)
class ChunkReader():
        """
        chunk reader
```

```
        it reads file then splits with separator and returns tuple
        """

        # read from end of file
        lines_per_chunk = 500

        buffered_lines = []

        def __init__(self, target_file, last_bytes_to_read_count=100000):
        if os.path.isfile(target_file) and os.access(target_file, os.R_OK):
        self.file = target_file
        self.buffered_lines = self._tail(last_bytes_to_read_count)
        logger.info(f"succesfully opened file and tail {last_bytes_to_read_count}
lines")
        else:
        logger.error(f"Unable to open file '{target_file}'")
        raise IOError("file not exist or not readable")


        def _tail(self, lines=150, _buffer=8192):
        """Tail a file and get X lines from the end"""
        with open(self.file, 'rt') as f:
        lines_found = []
        f.seek(0, 2)
        self.end_position = f.tell()
        position = self.end_position - _buffer
        while (len(lines_found) < lines) and position >= 0:
                f.seek(position)
                got_lines = [ item + '\n' for item in f.read(_buffer).split('\n')[1:-1]]
```

```
            lines_found = got_lines + lines_found

            position -= _buffer

        logger.info(f"{lines} tailed successfully")
        return lines_found[-lines:]

        def load_chunk(self):
        with open(self.file, 'rt') as f:
        f.seek(self.end_position)
        got_lines = f.readlines(self.lines_per_chunk)
        self.buffered_lines.extend(got_lines)
        self.end_position = f.tell()
        logger.debug(f"file chunk loaded successfully, end position is
{self.end_position}")

        def __iter__(self):
        return self

        def __next__(self):
        if (len(self.buffered_lines) <= self.lines_per_chunk / 2):
        self.load_chunk()
        try:
        return self.buffered_lines.pop(0)
        except IndexError:
        logger.debug(f"end of file reached")
        return None

        def next(self):
```

```
        return self.__next__()


клас Model

from time import time, sleep
from iptables import IptablesBlockMethod
from reader import ChunkReader
from parser import SplitParser
from helpers import get_logger, Config

logger = get_logger(__name__)

class Model(object):

        frequency_datastore = {}

        def __init__(self, datasource, ban_engine):
        self.datasource = datasource
        self.ban_engine = ban_engine

        config = Config()

        self.period_seconds = float(config['RATELIMIT_PERIOD_SECONDS'])
        self.ratelimit = float(config['RATELIMIT'])
        self.sleep_seconds = float(config['SLEEP_ON_EMPTY_LOG_SECONDS'])

        logger.info('model initialized')
```

```
def calculate_frequency_and_ban(self):
time_previous = 0
time_delta = 0

for sample in parser:
if time_delta > self.period_seconds:
        logger.info(f"end of time sample reached. filtering results")
        self.filter_ips()
        self.frequency_datastore = {}
        time_delta = 0

if not sample:
        logger.info(f"end of log file reached. will wait {self.sleep_seconds}
seconds")
        sleep(self.sleep_seconds)
        time_delta += self.sleep_seconds * 1000
        continue

ip = sample[0]
timestamp = float(sample[1])

time_delta += timestamp - time_previous
time_previous = timestamp

try:
        self.frequency_datastore[ip] += 1
except KeyError:
        logger.debug(f"IP {ip} added to datastore first time per time sample")
        self.frequency_datastore[ip] = 1
```

```
        def filter_ips(self):
        for ip in self.frequency_datastore:
        if self.frequency_datastore[ip] > self.ratelimit:
                logger.info(f"IP {ip} due to ban")
                self.ban_engine.ban(ip)
                logger.info(f"IP {ip} banned")


if __name__ == '__main__':
        config = Config()

        ban_engine = IptablesBlockMethod()
        reader = ChunkReader(config['SOURCE_LOG_FILE'])
        parser = SplitParser(reader)

        model = Model(datasource=parser, ban_engine=ban_engine)

        model.calculate_frequency_and_ban()import abc


абстрактний клас BanMethod
class BanMethod(metaclass=abc.ABCMeta):

        @abc.abstractmethod
        def ban(self, address):
        pass

        @abc.abstractmethod
```

```
def unban(self, address):
passclass Singleton(object):
__obj = False  # Private class variable.

def __new__(cls, *args, **kwargs):
if cls.__obj:
return cls.__obj
cls.__obj = super(Singleton, cls).__new__(cls)
return cls.__obj
```

абстрактний клас Reader

```
class Reader(metaclass=abc.ABCMeta):

@abc.abstractmethod
def __next__(self):
pass
@abc.abstractmethod
def __iter__(self):
```

Саган Максим Ярославович
Прізвище, ім’я по батькові

Інформаційних і прикладних технологій
Факультет

122 Комп’ютерні науки
Шифр і назва спеціальності

Комп’ютерні технологій обробки даних
Освітня програма

## ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему «РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ЗАХИСТУ ВІД DDOS АТАК НА ОСНОВІ ЛОГІВ NGINX» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____

(дата)
(підпис здобувача освіти)