Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: Засоби та механізми захисту програмного забезпечення

Виконавець: студент IV курсу, групи КБ-42

_____ **Карім НАССАР**
(підпис) (ім'я, прізвище)

	Ім'я, прізвище	**Підпис**
Керівник	Олена БОГУСЛАВСЬКА	

Нормоконтроль	Яніна ШЕСТАК	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-42** (група) **Нассар Каріму Махмуд Абду** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Засоби та механізми захисту програмного забезпечення

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Види програмного забезпечення, засоби впровадження захисту програмного забезпечення, модель ізольованого мережевого покриття, модель порушника, мережевий захист програмного забезпечення.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Проаналізувати сучасні методи захисту програмного забезпечення, навести приклади механізмів протидії зловмисникам, оцінити існуючі моделі захисту програмного забезпечення, дослідити поточні напрямки кібератак, порівняти загальні засоби захисту від таких атак, оцінити їх переваги та недоліки та створити мережеву модель ізольованої мережі для захисту програмного забезпечення від кібератак.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблена модель мережевого захисту програмного забезпечення від кібератак.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 24 жовтня 2022 року

Завдання видав _____ (підпис) Олена БОГУСЛАВСЬКА (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Карім НАССАР (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	24.10.2022 – 15.11.2022	*виконано*
2	Аналіз літератури	15.11.2022 – 30.12.2022	*виконано*
3	Обгрунтування вибору рішення	30.12.2022 – 03.01.2023	*виконано*
4	Огляд сучасних методів захисту програмного забезпечення	03.01.2023 – 03.02.2023	*виконано*
5	Дослідження загроз програмного забезпечення	03.02.2023 – 10.03.2023	*виконано*
6	Планування архітектури побудованої моделі захисту програмного забезпечення	10.03.2023 – 15.03.2023	*виконано*
7	Програмна реалізація моделі	15.03.2023 – 20.05.2023	*виконано*
8	Вироблення рекомендацій щодо впровадження ізольованої мережі на підприємстві	20.05.2023 – 31.05.2023	*виконано*
9	Оформлення пояснювальної записки	31.05.2023 – 02.06.2023	*виконано*
10	Підготовка до захисту кваліфікаційної роботи	02.06.2023 – 12.06.2023	*виконано*

Завдання видав _____ (підпис) Олена БОГУСЛАВСЬКА (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Карім НАССАР (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, має 64 сторінок основного тексту. Список використаних джерел містить 77 найменувань і займає 6 сторінок.

Методи дослідження кваліфікаційної роботи:

- метод аналізу;
- метод синтезу;
- метод порівняння;

Об'єктом дослідження є процес захисту програмного забезпечення.

Метою дослідження є розробка засобів та механізмів захисту програмного забезпечення. Поставлені наступні задачі:

- Виконати аналіз механізмів захисту доменного серверу від мережевих атак;
- Розглянути засоби сучасних механізмів захисту програмного забезпечення;
- Розробити засоби та механізми захисту програмного забезпечення;

Предметом дослідження є методи і засоби захисту програмного забезпечення від кібератак. Був проведений аналіз сучасних методів захисту від кібератак, поширених практик для реалізації захисту.

Побудована модель захищеного мережевого покриття, яке ізолює програмне забезпечення, розташоване у приватній мережі. Запропоновано використання розробленої моделі задля забезпечення захисту на установах, які мають потребу у захисті від мережевих атак.

Результати здійснених у кваліфікаційній роботі досліджень можуть бути використані спеціалістами з кібербезпеки на підприємстві з метою підвищення мережевого захисту програмного забезпечення.

Ключові слова: Захист програмного забезпечення, кібератака, DNS, VPN, технології захисту інформації.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

VPN	–	Virtual Private Network
API	–	Application Programming Interface
DNS	–	Domain Name Server
UDP	–	User Datagram Protocol
ПЗ	–	Програмне забезпечення
TLS	–	Transport Layer Security
OVPN	–	Open VPN
SDLC	–	Software Development Life Cycle
DNSSEC	–	Domain Name System Security Extentions
YAML	–	Yet Another Markup Language
SSL	–	Secure Sockets Layer
BIND	–	Berkeley Internet Name Domain

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі, що швидко розвивається, захист програмного забезпечення став критичною проблемою. Зі збільшенням частоти та складності кібератак, захист програмного забезпечення від шкідливих дій став актуальною проблемою для окремих осіб, організацій і суспільства в цілому. Дана робота має на меті розглянути тему інструментів і механізмів захисту програмного забезпечення, заглиблюючись у методи та стратегії, що застосовуються для захисту програмних систем від потенційних загроз.

Актуальність даної роботи визначається тим фактом, що суспільство постійно знаходиться під загрозою з боку кіберзлочинців, які використовують уразливості програмного забезпечення для шкідливих дій. Від витоку корпоративних даних і викрадення особистих даних до атак програм-вимагачів і викрадення інтелектуальної власності – наслідки порушення безпеки програмного забезпечення можуть бути руйнівними. Тому вкрай важливо розв'язувати цю проблему завчасно, розуміючи останні досягнення в галузі захисту програмного забезпечення та визначаючи надійні методи захисту програмного забезпечення від потенційних атак.

Аналіз останніх досліджень та літератури. Автори, які зробили внесок до галузі захисту програмного забезпечення: Брюс Шнайєр [1], Кевін Мітник [2], Брайан Кребс [3].

Ціль даної роботи полягає у наданні розробникам програмного забезпечення, фахівцям з кібербезпеки та зацікавленим сторонам знання та розуміння, необхідні для визначення, оцінки та впровадження надійних інструментів і механізмів захисту. Досліджуючи різні підходи до захисту програмного забезпечення, це дослідження має на меті надати можливість окремим особам і організаціям підвищити безпеку своїх програм, зменшуючи ризики, пов’язані з кіберзагрозами.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ МЕТОДІВ ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд на сучасну потребу захисту програмного забезпечення

Оскільки програмні додатки стали настільки важливою частиною промисловості та повсякденного життя, забезпечення їх захисту стало надзвичайно важливим. Розвиток технологій приносить із собою кіберзагрози, що змушує розробників програмного забезпечення та організації застосовувати передові тактики захисту від цих ризиків. У цьому розділі ми досліджуємо потреби захисту програмного забезпечення та механізми, які можуть ефективно захистити програмні додатки від несанкціонованого доступу, порушень даних або атак третіх осіб.

Цифровізація та розвиток Інтернету створили середовище, в якому кіберзлочинці постійно атакують уразливі місця програмного забезпечення. Традиційні заходи безпеки можуть більше не забезпечувати належного захисту від їхніх прогресивних атак. В результаті попит на сучасні та надійні рішення для захисту програмного забезпечення ніколи не був таким великим, як зараз.

Застосовуючи сучасні методи безпеки для захисту програмних додатків, розробники та організації можуть активно протидіяти загрозам безпеці протягом усього циклу розробки та гарантувати безпеку додатків на кожному етапі розробки. Ці розширені інструменти та механізми включають різноманітні методи безпеки, включаючи методи безпечного кодування, інструменти динамічного та статичного аналізу коду, процедури тестування програмного забезпечення, системи виявлення вторгнень і безпечні структури розгортання програмного забезпечення.

Розглянемо мережеву кібератаку, яка відбулась у 2011 році, направлену на компанію Sony та її підрозділ PlayStation Network [4].

У 2011 році компанія Sony зазнала серйозного порушення безпеки цифрового медіа-сервісу PlayStation Network (PSN). Ця атака призвела до витоку особистої

інформації, що належить мільйонам користувачів PSN, а також до значних збоїв у роботі мережевих послуг.

17 квітня 2011 року Sony виявила [4], що PSN було зламано неавторизованими особами, що змусило їх закрити його та провести подальше розслідування перед тим, як остаточно відновити роботу 14 травня 2011 року. Зловмисники скористалися вразливістю в мережевій інфраструктурі Sony, щоб отримати несанкціонований доступ як до PSN, так і до Qriocity – сервіс потокової трансляції музики та відео від Sony. Потрапивши в систему, кіберзлочинці отримали доступ до особистої інформації користувачів, серед якої було приблизно 77 мільйонів облікових записів, які включали імена, паролі, адреси електронної пошти та дані кредитних карток.

Через цю атаку служби PSN стали недоступними на тривалий період, позбавивши користувачів можливості використовувати сервіс, отримувати доступ до цифрового вмісту або використовувати будь-які його функції, а також викликали занепокоєння серед постраждалих користувачів щодо крадіжки особистих даних.

Sony швидко відреагувала, тимчасово призупинивши доступ до PSN і найнявши експертів з кібербезпеки, щоб оцінити ступінь порушення та посилити системи безпеки. Вони також залучили зовнішні фірми з кіберзахисту у форматі співпраці «Outsource» для процесу розслідування та відновлення.

Компанія піддалася критиці за те, що вона не змогла своєчасно повідомити постраждалих користувачів про порушення безпеки, що вплинуло на них, і можливу компрометацію їхніх особистих даних. Після отримання інформації про атаку, постраждалим особам було рекомендовано негайно змінити паролі та уважно стежити за фінансовими рахунками на предмет підозрілої діяльності.

Фірма зіткнулася з різними регулятивними запитами після того, як атака на PlayStation Network [4] викликала широке висвітлення та інтерес засобів масової інформації в усьому світі. Розслідування, проведені численними державними органами, сформували методи безпеки, тактику реагування на інциденти та методи захисту даних користувачів, які були рекомендовані компанії. У свою чергу це виявило необхідність посилення заходів захисту даних, що призвело до дискусій навколо правил і стандартів кібербезпеки в усьому світі.

Sony сильно постраждала від цього порушення у фінансовій сфері. Оцінені витрати досягли сотень мільйонів доларів. Розслідування, покращення системи захисту, юридичні врегулювання та компенсаційні виплати постраждалим користувачам – усе це призвело до тимчасового зниження довіри та лояльності клієнтів до Sony.

Атака PlayStation Network у 2011 році [4] підкреслила як її вразливість, так і потенційні наслідки порушень безпеки в онлайн-мережах. Таким чином, ця подія підкреслила необхідність надійних заходів безпеки, оперативних дій реагування на інциденти та ефективного спілкування з постраждалими користувачами, щоб мінімізувати вплив таких атак.

Одним із головніших методів сучасного захисту програмного забезпечення є безпечне кодування [5]. Дотримання встановлених інструкцій та найкращих практик написання коду, розробники можуть мінімізувати вразливості у коді. Таким чином, інженери зменшують простір можливостей для атак, використовуючи перевірку вхідних даних, шифрування вихідних даних, безпечне керування сеансами, процедури обробки помилок, які покривають несподівані винятки, міжсайтовий скриптінг (XSS) [6], переповнення буфера, а також реалізовуючи захист від SQL ін'єкцій [7].

Інструменти динамічного та статичного аналізу коду [8] допомагають виявляти слабкі місця у вихідному коді програмного забезпечення, виконуючи автоматичне сканування для виявлення потенційних недоліків безпеки. Це дозволяє розробникам вирішити певну кількість потенційних проблем ще до розгортання ПЗ на серверах.

Крім того, використання надійних методів тестування, таких як тестування на проникнення (Pentesting) [9], дозволяє ідентифікувати слабкі сторони за допомогою симуляції атак, допомагаючи забезпечити стійкість проти реальних загроз для забезпечення необхідної стійкості програмного забезпечення.

Інфраструктури безпечного розгортання програмного забезпечення пропонують всеохоплюючий підхід до безпечного розповсюдження програмного забезпечення кінцевим користувачам, використовуючи безпечні протоколи зв'язку,

механізми шифрування та процеси автентифікації для захисту конфіденційної інформації під час передачі та зберігання.

Багатофакторна автентифікація та біометрія додають додатковий рівень захисту від доступу до критичних ресурсів неавторизованих користувачів.

Сучасні підходи до захисту програмного забезпечення складаються з різноманітних складних інструментів і механізмів, які виходять за рамки традиційних практик безпеки. Використовуючи методи наведені вище, розробники програмного забезпечення та організації можуть підвищити безпеку додатків від кібератак. Вивчаючи та впроваджуючи сучасні засоби захисту, компанії можуть впевнено орієнтуватися в переліку загроз, що постійно змінюється, щоб захистити цілісність, конфіденційність і доступність свого головного активу – інформації. Як казав один з найвідоміших політиків світу Вінстон Черчилль: «Хто володіє інформацією – той володіє світом».

1.2 Практика безпечного кодування

Практики безпечного кодування відіграють вирішальну роль у запобіганні порушенням безпеки та зміцненні кібербезпеки. Дотримуючись встановлених правил і вказівок, розробники можуть значно зменшити вразливості, які загрожують цілісності програмних додатків.

Перевірка вхідних даних – одна з основних умов безпечного кодування. Вона передбачає перевірку всіх введених користувачами даних щодо очікуваного формату, перевірку на наявність зловмисного коду, таких як впровадження SQL атак [7] або атак із впровадженням команд, які дозволяють зловмисникам отримати контроль над програмами із зовнішніх джерел. Неможливість належної перевірки введених даних може спричинити уразливості впровадження коду, що дозволяє зловмисникам здійснювати атаки за межами сервера.

Кодування виводу – популярна практика безпечного кодування. Вона запобігає запобігти атакам міжсайтового скриптінгу (XSS) [6]. Забезпечуючи належне кодування даних користувача під час відображення на веб-сторінках, вихідне

кодування запобігає впровадженню зловмисниками зловмисних сценаріїв, які виконуватимуться автоматично, коли їх представлять зловмисники.

Ефективна обробка помилок є дуже важливою як з точки зору функціональності, так і безпеки. Впровадження належних механізмів обробки помилок, які відображають інформативні повідомлення про помилки без розкриття подробиць системи, допомагає захистити програми від потенційних зловмисників, які отримують уявлення про вразливі місця, і водночас запобігає витоку інформації.

Безпечне зберігання паролів є ключовим для захисту облікових записів користувачів і конфіденційних даних. Дотримуючись найкращих практик кодування, алгоритми безпечного хешування bcrypt [10] або Argon2 [11] пропонують додатковий захист, включаючи методи покращення обчислювальної вартості. Також ці алгоритми дозволяють використовувати ускладнення шифрування у вигляді механізму Salt [12], який ускладнює зловмисникам процес злому паролів.

Використання перевірених бібліотек є одним із ефективних методів підвищення безпеки програмного забезпечення. Такі бібліотеки містять готові компоненти та модулі, які пройшли вичерпні тести та огляди безпеки, таким чином зменшуючи ймовірність появи вразливостей через неправильне використання коду.

Регулярне проведення перевірок коду: як вручну, так і за допомогою автоматизованих інструментів – допомагає завчасно виявляти вразливі місця безпеки та виправляти їх перед використанням у реальних середовищах. Регулярна перевірка дозволяє розробникам раніше виявляти та усунути вразливості, зводячи до мінімуму будь-які шанси використання експлойту у робочих середовищах.

Життєвий цикл розробки безпечного програмного забезпечення (Software Development Life Cycle) [13] гарантує, що питання безпеки враховуються на всіх етапах процесу розробки програмного забезпечення, від моделювання загроз і аналізу вимог, до проектування, впровадження, тестування та поточного обслуговування.

Дотримуючись практик безпечного кодування та дотримуючись правил, розробники можуть значно зменшити кількість порушень безпеки, зміцнюючи загальну кібербезпеку. Дотримання таких практик дозволяє розробникам зменшити

вразливості, забезпечити надійні заходи безпеки додатків та захистити конфіденційні дані користувачів від зловмисного використання.

1.3 Інструменти динамічного та статичного аналізу коду

Динамічний і статичний аналіз коду є двома ефективними методами, які використовуються для виявлення вразливостей безпеки або потенційних недоліків у програмному коді.

Динамічний аналіз коду (DCA) [14], також відомий як динамічне тестування безпеки додатків (DAST), — це підхід, який використовується для тестування програм під час їх роботи або виконання в режимі реального часу для виявлення вразливостей, слабких місць безпеки та можливих загроз всередині них. Це передбачає уважне спостереження за тим, як компоненти поводяться один з одним перед виконанням цього аналізу, щоб якомога швидше виявити вразливі місця. Динамічний аналіз коду має на меті виявити потенційні недоліки безпеки, невидимі за допомогою статичного аналізу коду. Розглянемо більш детально інструменти динамічного аналізу коду.

Сканери веб-додатків використовують програмні сканери для перевірки веб-додатків на наявність вразливостей, зокрема впровадження SQL атак [7], атак міжсайтового скріптінгу (XSS) [6] і небезпечні прямі посилання на об’єкти. До популярних сканерів веб-додатків належать OWASP ZAP [15], Burp Suite [16] і Acunetix [17].

Інструменти фаззингу [18] генерують і вводять зловмисні або неочікувані дані в інтерфейси програми, щоб виявити такі вразливості, як переповнення буфера або помилки перевірки введення тощо. Популярними прикладами є American Fuzzy Lop (AFL) [19], Peach Fuzzer [20] і Sulley [21] як інструменти для розпушування.

Інструменти RASP (Runtime Application Security Protection) [22] відстежують і оцінюють поведінку програми під час виконання, щоб виявляти та запобігати атакам на безпеку, виявляти аномалії, генерувати політики безпеки та надавати захист у

режимі реального часу. Прикладами є Contrast Security [23], Sqreen [24] і AppTrana [25].

Інструменти pentest [9] імітують атаки на програму для виявлення вразливостей і можливих шляхів проникнення для зловмисників, часто за допомогою таких інструментів, як Metasploit [26], Nessus [27] і OpenVAS [28].

Статичний аналіз коду (або статична перевірка коду) [29] — це підхід, який використовується для перевірки коду програмного забезпечення без його запуску, зазвичай складається з перегляду вихідних або скомпільованих бінарних файлів для виявлення потенційних вразливостей, недоліків у побудові коду та невідповідності найкращим практикам або стандартам. Інструменти статичного аналізу коду переглядають вихідні файли в пошуках дірок у безпеці, помилок і проблем з обслуговуванням, які інакше могли б залишитися непоміченими. Розглянемо більш детально інструменти статичного аналізу коду.

Інструменти аналізу вихідного коду (Source code analysis tools) виконують статичний аналіз коду щодо файлів вихідного або скомпільованого двійкового коду, щоб виявити вразливості, недоліки та дотримання методів безпечного кодування у вихідних або бінарних файлах, включаючи такі проблеми, як переповнення буфера, незахищені криптографічні реалізації тощо. Популярні інструменти SAST [30], такі як SonarQube [31], Fortify Static Code Analyzer [32] і Checkmarx [33], можуть надати такі рішення для тестування.

Інструменти перевірки коду надають розробникам автоматичні перевірки та рекомендації, розроблені для підвищення якості та безпеки коду, наприклад ESLint [34], PMD [35] і FindBugs [36]. Вони визначають вразливості та відповідність стандартам кодування.

Сучасні компілятори та інструменти збирання часто включають функції або плагіни для виконання статичного аналізу коду під час процесів компіляції чи збирання, забезпечуючи додатковий захист від помилок кодування, вразливостей безпеки та проблем із продуктивністю. Прикладами таких компіляторів і інструментів є GCC [37], Clang [38] і Maven [39] (з відповідними плагінами).

Інструменти сканування залежностей оцінюють залежності програмного забезпечення, такі як сторонні бібліотеки, фреймворки та компоненти, на наявність відомих вразливостей і застарілих версій, допомагаючи виявляти та зменшувати ризики безпеки, створені незахищеними залежностями. Такі інструменти, як OWASP Dependency-Check [40], Snyk [41] і WhiteSource [42], можуть допомогти вірішенням цієї проблеми.

Ці інструменти надають дуже важливу допомогу у виявленні потенційних вразливостей системи безпеки, помилок у розробці коду та порушень правил безпечного кодування. Слід пам'ятати, що жоден засіб не може забезпечити повний захист. Щоб досягти всебічної оцінки, є необхідним поєднати інструменти динамічного та статичного аналізу коду разом із перевіркою коду вручну як частину загальної стратегії оцінки безпеки.

1.4 Інфраструктура безпечного розгортання програмного забезпечення

Налагодження безпечної інфраструктури розгортання програмного забезпечення стосуються практик і методів, які використовуються для забезпечення відповідного рівня безпеки під час розгортання та під час користування програмним забезпеченням.

Основною метою безпечного розгортання програмного забезпечення є захист цінних активів даних. Організації часто зберігають широкий спектр конфіденційної інформації, починаючи від записів про клієнтів і власних алгоритмів, до фінансових записів і комерційних таємниць. Будь-яке порушення під час розгортання може надати доступ злочинцям до конфіденційних матеріалів, що призведе до крадіжки особистих даних, фінансового шахрайства та репутаційної шкоди для організацій. Застосовуючи заходи безпеки, шифрування, контроль доступу та безпечні канали зв’язку, компанія може захистити свої активи від несанкціонованого доступу та зменшити пов’язані з цим ризики.

Впровадження методів безпечного керування конфігурацією гарантує, що процес розгортання програмного забезпечення відповідає суворим інструкціям щодо

безпеки, таким як безпечне зберігання конфігураційних файлів і зашифрованих конфіденційних даних та керування доступом, що запобігає несанкціонованим змінам або модифікаціям або будь-якому потенційному втручанню в інсталяцію програмного забезпечення.

Процес керування версіями програмного забезпечення включає перевірку цілісності та автентичності версій перед розгортанням за допомогою таких інструментів, як цифрові підписи, криптографічні хеші або безпечні канали розповсюдження, щоб гарантувати, що вони не будуть змінені в транзиту або скомпрометовано під час використання.

Коли йдеться про розгортання програмного забезпечення через мережі чи Інтернет, використання захищених каналів (наприклад, протоколів шифрування HTTPS[43], SSH[44]) допомагає підтримувати конфіденційність і цілісність даних під час процесу. Захищений зв'язок допомагає уникнути атак підслуховування, а також запобігає несанкціонованому доступу третіх сторін до конфіденційної інформації.

Безперервний моніторинг і ведення журналів. Інтеграція надійних механізмів моніторингу та ведення журналів під час розгортання програмного забезпечення забезпечує видимість процесу його розгортання в режимі реального часу, наприклад виявлення нестандартних дій, неавторизованих змін і потенційних інцидентів безпеки. Комплексні журнали забезпечують контрольний слід для дослідження, щоб допомогти точно визначити будь-які проблеми джерела або порушення, які виникають під час розгортання.

Підтримка безпеки середовища розгортання є надзвичайно важливою, починаючи від посилення безпеки серверів і систем, вимкнення непотрібних служб і портів, застосування оновлень безпеки до використання систем виявлення та запобігання вторгненням. Усе це сприяє створенню системи, де потенційні вразливості під час розгортання зменшуються.

Наявність безпечної стратегії випуску минулої версії та відновлення гарантує, що система може повернутися до попереднього робочого стану, якщо це необхідно. Регулярне резервне копіювання та тестування процедур відновлення допомагають зменшити ризики, пов'язані з проблемами розгортання або порушеннями.

Безпечне розгортання програмного забезпечення є невід'ємною складовою життєвого циклу розробки програмного забезпечення. Методи наведені вище забезпечують захист даних, зменшують фінансові та репутаційні ризики, сприяють дотриманню вимог до кібербезпеки на підприємстві та зміцнюють довіру клієнтів. Організації, які віддають пріоритет безпечному розгортанню, демонструють свою прихильність до захисту даних, мінімізуючи інциденти безпеки, створюючи міцну основу для успішної роботи своїх програмних додатків. Інвестуючи в надійні заходи безпеки під час розгортання, вони можуть захистити дані, зберігаючи безперебійність бізнес-операцій, залишаючись безпечними в сучасному цифровому середовищі, яке швидко змінюється.

1.5 Висновки до першого розділу

У першому розділі кваліфікаційної роботи розглядалися різні інструменти та механізми, що використовуються для захисту програмного забезпечення. Було наголошено на тому, наскільки важливим є впровадження ефективних заходів безпеки для підтримки цілісності, конфіденційності та доступності програмних систем.

Одним із інцидентів, який підкреслив важливість захисту програмного забезпечення, стала атака Sony на PlayStation Network у 2011 році. Ця подія продемонструвала організаціям наслідки, до яких можуть призвести неналежні механізми безпеки та невдалі засоби захисту.

Методи безпечного кодування стали незамінним елементом створення безпечних програмних систем. Дотримуючись інструкцій із безпечного кодування, розробники можуть активно пом'якшувати потенційні вразливості під час життєвого циклу розробки програмного забезпечення та знижувати ризики, такі як атаки SQL ін'єкцій, атаки міжсайтового сценарію XSS та інші. Крім того, методи безпечного кодування зміцнюють системи програмного забезпечення проти будь-якої форми зловмисного використання, роблячи системи стійкими до зловживання чи атак.

Найкращі практики розгортання програмного забезпечення відіграють важливу роль у підтримці безпечної екосистеми програмного забезпечення. Такі заходи, як безпечне керування конфігурацією, регулярне розгортання оновлень/патчів і надійний контроль доступу, сприяють підтримці цілісності конфіденційності та доступності розгорнутого програмного забезпечення від розробки до розгортання й далі. Ці методи допомагають гарантувати, що програмне забезпечення залишається захищеним у на кожному етапі протягом усього терміну використання.

Методи динамічного та статичного аналізу коду пропонують важливі функції для розпізнавання вразливостей у програмних системах. Динамічні методи, такі як тестування на проникнення та моніторинг під час виконання, дозволяють організаціям швидко реагувати на нові загрози. Статичні методи аналізу, такі як перегляд коду або автоматичне сканування, забезпечують завчасне попередження про потенційну слабкість у кодовій базі, допомагаючи завчасно зменшити ризики безпеки. Застосовуючи такі механізми, організації можуть підвищити безпеку програмного забезпечення, підвищуючи загальну стійкість до кіберзагроз.

Захист програмних систем від зловмисних дій вимагає інтегрованого та багатогранного підходу. Опираючись на резонансні інциденти, запроваджуючи методи безпечного кодування, найкращі практики розгортання програмного забезпечення та механізми аналізу коду як основу для значного зменшення вразливостей програмного забезпечення, організації можуть суттєво зменшити ризики, пов’язані з уразливістю програмного забезпечення, одночасно створюючи безпечне цифрове середовище для бізнес-операцій.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ МЕХАНІЗМІВ ЗАХИСТУ DNS СЕРВЕРУ

2.1 Особливості захисту серверу доменних імен

Domain Name Server (DNS) [45] означає систему доменних імен, невід'ємний елемент інфраструктури Інтернету, який перетворює зрозумілі людині доменні імена на їхні цифрові IP-адреси [46]. Простіше кажучи, DNS діє як телефонна книга для веб-сайтів і служб в Інтернеті, роблячи доступними домени, які легко запам'ятати, замість складних числових IP-адрес.

Коли користувачі вводять доменні імена, такі як "https://google.com", у веб-переглядачі, їхній веб-браузер повинен знати, яка IP-адреса відповідає цьому доменному імені, щоб встановити з'єднання з потрібним веб-сайтом. Цю роль виконує DNS – зіставляючи ці доменні імена на пов'язані IP-адреси. Це дозволяє браузерам швидше та надійніше знаходити відповідні веб-сервери та підключатися до них.

Система доменних імен працює в ієрархічному та розподіленому порядку. Ця мережа містить кілька DNS-серверів, відповідальних за надання інформації DNS. Розглянемо приклади серверів, які до них належать.

Рекурсивні DNS-розподілювачі [47]. Рекурсивні resolver-сервери зазвичай працюють постачальниками послуг Інтернету або сторонніми постачальниками послуг DNS і діють як посередники, коли користувачі надсилають DNS-запити. Після отримання запиту, ці сервери шукають IP-адресу, запитуючи інші DNS-сервери від імені користувачів, щоб повернути відповідь для отримання клієнтом IP-адреси.

Кореневі сервери імен [48] – служать верхнім рівнем ієрархії DNS, зберігаючи повний список DNS-серверів для доменів верхнього рівня (TLD [49]), таких як ".com", ".org" і ".net", а також з кодом країни верхнього рівня, наприклад «ua» або «fr». Кореневі сервери імен забезпечують перенаправлення до відповідних серверів імен верхнього рівня залежно від того, який домен запитували клієнти.

Сервери імен TLD [49]. Кожен домен верхнього рівня («TLD») підтримує власний набір серверів імен для зберігання даних про реєстрацію домену в своєму TLD, наприклад для «.com». Ці сервери імен обробляють запити, пов'язані з усіма доменними іменами, що закінчуються цією літерою чи цифрою, а також надають перенаправлення до авторитетних серверів імен, які їх обслуговують.

Авторитетні сервери імен (ANS [50]) – на цих серверах зберігаються фактичні записи DNS для окремих доменних імен. Коли DNS-перетворювачі отримують запити щодо певних доменів, вони зв'язуються з цими авторитетними серверами імен, щоб отримати пов'язані з ними IP-адреси або записи DNS (наприклад, адреси поштових серверів або інформацію DNSSEC [51]).

DNS працює за допомогою повідомлень запитів і відповідей, які надсилаються через певні протоколи, такі як UDP-порт 53 [52] для використання протоколу DNS, а також DoH [53] або DoT [53] для безпечного зв'язку. (рис. 2.1)

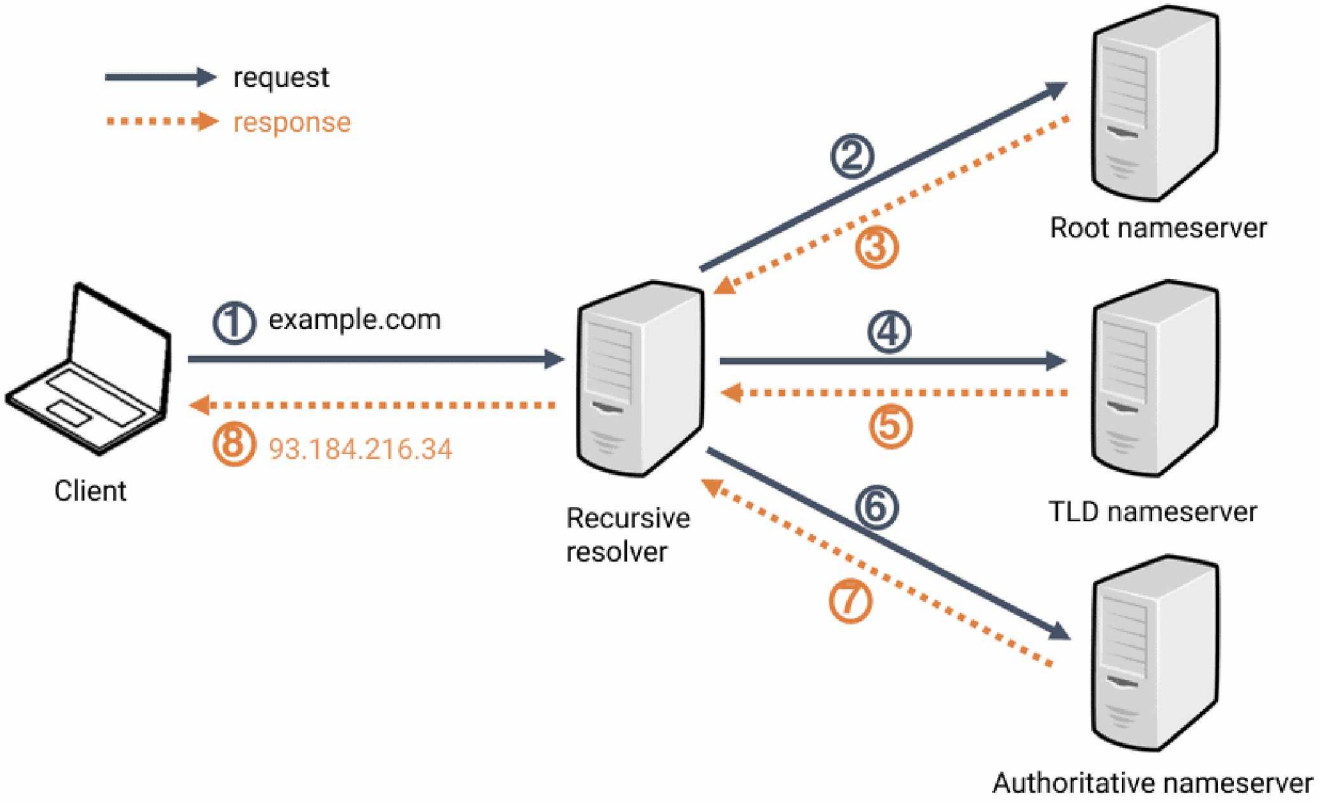

Рис 2.1 Архітектура DNS технології

Наявність захищеного DNS-сервера має вирішальне значення для захисту конфіденційності користувачів, запобігання атакам на інфраструктуру DNS,

підвищення безпеки мережі та дотримання стандартів відповідності. Завдяки пріоритетним заходам безпеки в рамках своїх інфраструктур DNS, організації можуть створити онлайн-середовище, яке захищає своїх клієнтів, а також їх конфіденційну інформацію від загроз.

DNS-сервери стали об'єктом різноманітних кібератак, зокрема DNS-спуфінгу [54], пошкодження кешу та викрадення cookie [55]. Такі атаки можуть призвести до пошкодження HTTP пакету, перенаправлення користувачів на шкідливі сайти та перехоплення конфіденційної інформації. Захищений DNS-сервер використовує такі заходи, як DNSSEC [51] (розширення безпеки DNS), які перевіряють автентичність і цілісність відповідей, тим самим зменшуючи ризики, пов'язані з такими видами атак.

Є можливість допомогти організаціям виявляти та блокувати підозрілу мережеву активність, використовуючи захищені DNS-сервери, оснащені функціями аналізу загроз, які виявляють відомі шкідливі домени або блокують зв'язок із серверами, які використовують зловмисне програмне забезпечення. Це покращує безпеку мережі, допомагаючи запобігти атаці, а також витоку даних.

Атаки, спрямовані на інфраструктуру DNS, можуть суттєво вплинути на онлайн-сервіси та доступність веб-сайтів і додатків, створюючи перешкоди в стабільній роботі. Захищені DNS-сервери використовують такі методи, як обмеження швидкості та фільтрація трафіку, щоб пом'якшити такі атаки та підтримувати доступність послуг для своїх доменних імен.

2.2 Особливості захисту серверу за допомогою приватної мережі

Оскільки конфіденційність даних і онлайн-безпека є надзвичайно важливими для сучасних користувачів Інтернету, сервери віртуальної приватної мережі (VPN [56]) стали потужними інструментами. Сервери VPN створюють безпечні підключення до Інтернету, шифруючи дані користувачів перед тим, як направляти їх через віддалені сервери з метою маршрутизації.

Одна з основних цілей VPN-серверів – захист онлайн-зв'язку через ненадійні мережі, такі як публічні точки доступу Wi-Fi. Створюючи зашифрований тунель між

пристроєм клієнта і сервером призначення, VPN-сервери створюють додатковий бар'єр для сторонніх очей, захищаючи конфіденційну інформацію.

Конфіденційність є невід'ємним правом кожного користувача у сучасну цифрову епоху, і сервери VPN відіграють важливу роль у його підтримці. Перенаправляючи інтернет-трафік через віддалені сервери та приховуючи IP-адреси за допомогою IP-адрес серверів, VPN додають ще один рівень анонімності, що ускладнює веб-сайтам, рекламодавцям або іншим організаціям відстежувати онлайн-діяльність або збирати дані про звички, надаючи користувачам можливість безпечно користуватись Інтернетом.

VPN-сервери дозволяють людям обходити обмеження та цензуру, встановлені країнами чи організаціями, підключаючись до VPN-сервера в іншому регіоні, який виглядає так, ніби ви отримуєте доступ до нього з цього регіону, надаючи доступ до заблокованого регіону вмісту, наприклад, соціальні медіа-платформи або веб-сайти.

Підвищення безпеки в загальнодоступних мережах. Загальнодоступні мережі Wi-Fi створюють численні загрози безпеці через свою відкритість для потенційних зловмисників, що робить VPN-сервери ефективним механізмом захисту від потенційних ризиків у цих мережах. Шифруючи та захищаючи з'єднання, мережа VPN допомагає захистити користувача від зловмисників, які намагаються проникнути, перехоплюючи комунікації або отримуючи доступ до пристроїв, підключених через загальнодоступні точки доступу Wi-Fi. Це є особливо важливо під час отримання доступу до конфіденційної інформації або проведення банківських транзакцій у цих точках доступу.

Сервери VPN забезпечують безпечний віддалений доступ до корпоративних мереж або внутрішніх ресурсів із віддалених місць, надаючи співробітникам доступ із зашифрованим зв'язком і захистом даних. (рис 2.2)

Вибираючи надійний сервер VPN, важливо брати до уваги такі фактори, як розташування серверів, протоколи шифрування, швидкість з'єднання та політика ведення журналів. Вибираючи постачальників послуг із сильними можливостями захисту конфіденційності та великими серверними мережами, можна мінімізувати ризики успішної кібератаки.

Сервери VPN стали безцінними інструментами для захисту онлайн-з'єднань, захисту конфіденційності та розблокування обмеженого вмісту. Завдяки шифруванню даних і маршрутизації їх через віддалені сервери вони пропонують користувачам безпечний досвід перегляду – незалежно від того, чи це означає захист зв'язку в публічних мережах, обхід певних обмежень або віддалений доступ до корпоративних ресурсів, надаючи окремим особам і організаціям впевненість, необхідну для впевненого орієнтування в цифровому світі.

У міру подальшого розвитку використання Інтернету ці критично важливі компоненти продовжують надавати користувачам покращений захист, одночасно захищаючи права користувачів в Інтернеті та захищаючи їх власну конфіденційність в Інтернеті.

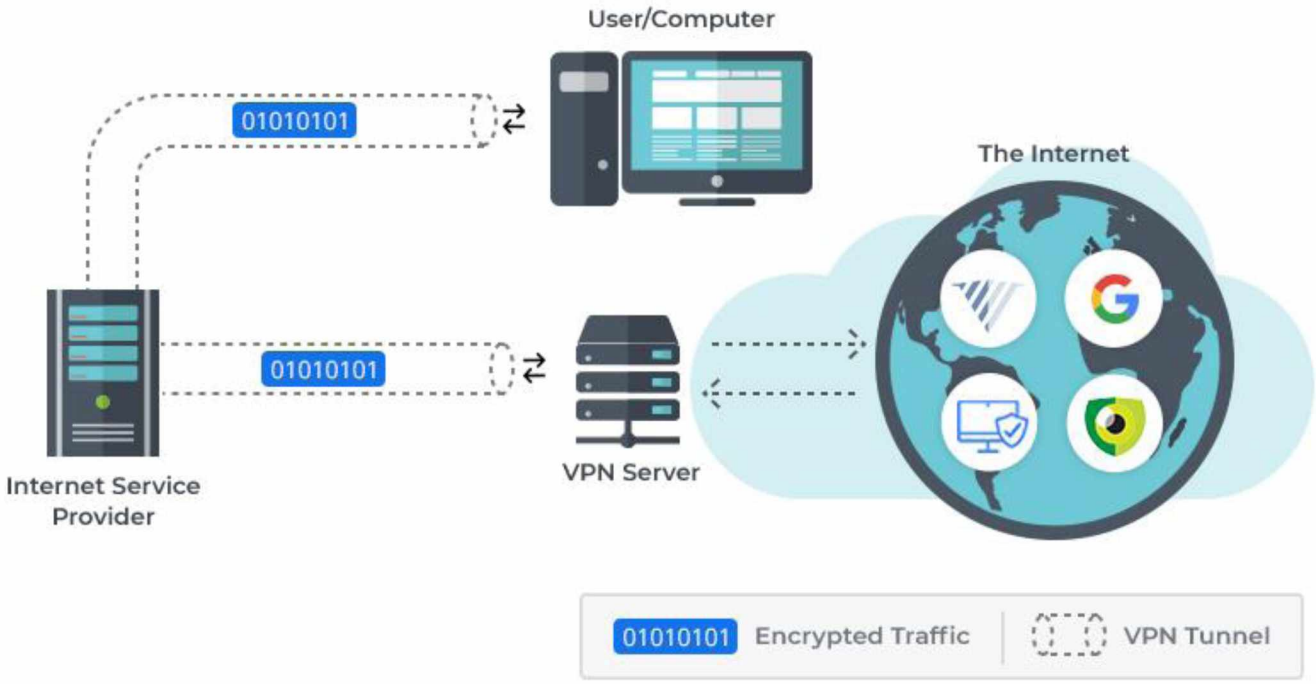

Рис 2.2 Архітектура VPN технології

OpenVPN [57] — це технологія віртуальної приватної мережі (VPN) із відкритим кодом, яка використовується для безпечного з'єднання через Інтернет між пристроями. Широко визнаний завдяки своїй надійності, гнучкості та крос-платформній сумісності. OpenVPN використовує захищені протоколи, такі як бібліотека OpenSSL [58], для встановлення зашифрованих тунелів для передачі даних

у своєму мережевому середовищі, таким чином створюючи безпечне мережеве середовище.

OpenVPN використовує надійні алгоритми шифрування, такі як AES [59] (Advanced Encryption Standard) і Blowfish [60], щоб захистити дані під час передачі від можливого прослуховування чи несанкціонованого доступу, захищаючи конфіденційну інформацію від можливого вторгнення третіх сторін.

OpenVPN пропонує велику універсальність завдяки сумісності з різними операційними системами: Windows, macOS, Linux або мобільними операційними системами, такими як iOS або Android. Оскільки він сумісний із різними платформами, він швидко став вибором для компаній, а також для окремих осіб, які шукають безпечне підключення на кількох пристроях.

Модульна архітектура OpenVPN дозволяє плавно масштабуватись для компаній будь-якого розміру, що робить його придатним як для малого бізнесу, так і для корпоративних організацій. Він може працювати з вражаюче високою кількістю одночасних з'єднань, дозволяючи організаціям розширювати свою інфраструктуру VPN, коли це необхідно.

OpenVPN пропонує різноманітні безпечні протоколи, такі як TCP [61] (протокол керування передачею) та UDP [52] (протокол дейтаграм користувача), що дає користувачам можливість вибрати один з них відповідно до їхніх індивідуальних потреб і вимог. Оскільки їм доступні обидва варіанти, завжди існує варіант, щоб збалансувати продуктивність і безпеку під час прийняття рішень щодо послуг OpenVPN.

OpenVPN підтримує автентифікацію TLS [62], щоб додати ще один рівень захисту від атак типу "людина посередині" шляхом перевірки клієнтів і серверів за допомогою цифрових сертифікатів для автентифікації як клієнтів і серверів відповідно. Це допомагає запобігти несанкціонованому доступу та запобігти атакам, спрямованим на використання «людини посередині».

OpenVPN надає користувачам численні параметри конфігурації, які дозволяють їм налаштувати свої VPN-з'єднання точно відповідно до індивідуальних потреб. Він

підтримує різні режими підключення, що забезпечує більшу гнучкість при створенні мереж VPN.

OpenVPN має велику базу користувачів і розробників, яка сприяє його постійному розвитку та підтримці.

OpenVPN забезпечує ефективний засіб безпечного підключення до загальнодоступних мереж Wi-Fi, які можуть становити потенційну загрозу безпеці, наприклад прослуховування або перехоплення даних. Перенаправляючи трафік через тунельне з'єднання OpenVPN, користувачі можуть захистити свої комунікації та дані від таких загроз, як прослуховування та перехоплення даних.

OpenVPN може допомогти окремим особам захистити їх конфіденційність в Інтернеті, зберігаючи при цьому анонімність шляхом шифрування інтернет-трафіку та маскування IP-адрес — особливо важливого інструменту під час перегляду з ненадійних мереж або країн з обмежувальною політикою в Інтернеті.

OpenVPN — це адаптивна та надійна технологія VPN, яка пропонує надійну безпеку, гнучкість і крос-платформну сумісність. Завдяки надійній технології шифрування та широким можливостям конфігурації, які підтримують різноманітні безпечні протоколи, такі як PKI [63] (протокол інфраструктури закритого ключа), OpenVPN довела, що є привабливим варіантом для окремих осіб та організацій, які шукають безпечний зв'язок через Інтернет – будь то віддалений доступ, сайт-до-підключення до сайту або забезпечення конфіденційності в загальнодоступних мережах Wi-Fi. OpenVPN забезпечує надійний зв'язок через Інтернет із надійними результатами, які захищають безпеку даних людей.

WireGuard [64] — це також популярний вдосконалений протокол VPN з відкритим кодом, призначений для захисту мережевого зв'язку. Розглянемо його головні переваги.

Мінімалістична кодова база WireGuard спрощує перевірку недоліків безпеки. Крім того, його архітектура значно знижує ризики безпеки порівняно з традиційними протоколами VPN і, таким чином, покращує їхній загальний рівень безпеки.

WireGuard використовує обмін ключами Діффі-Хеллмана [65] під час процесу handshake для встановлення захищених ключів сеансу, забезпечуючи

конфіденційність від зловмисників, навіть якщо їхній приватний ключ стає скомпрометованим, оскільки ключі сеансу змінюються з кожним сеансом сеансу.

WireGuard вбудовано в ядро Linux, щоб повною мірою скористатися всіма його функціями безпеки – захистом пам'яті та контролем доступу серед них – для захисту як цілісності мережевого трафіку, так і конфіденційності.

WireGuard був розроблений для швидкої та ефективної роботи, забезпечуючи безпечний зв'язок без обмеження пропускної здатності мережі. Його легкі криптографічні операції значно сприяють цій швидкості та ефективності.

WireGuard створено з урахуванням стійкості, його гнучкість дозволяє легко обробляти зміни, не порушуючи встановлений VPN-тунель, наприклад – під час адаптації динамічних IP-адрес або роумінгу між мережами, не порушуючи безпеку VPN-з'єднання.

Хоча WireGuard може похвалитися потужними функціями безпеки, жодне програмне забезпечення не застраховано від потенційних вразливостей. Так, наприклад, більшість компаній все ще віддають перевагу традиційному протоколу OpenVPN, оскільки він перевірений часом та більш здатний до масштабованості ніж WireGuard.

2.3 Механізм захисту DNS сервера за допомогою VPN

У сучасному світі, захист серверів системи доменних імен (DNS) став важливим для підтримки цілісності та доступності онлайн-сервісів. Одним з ефективних методів підвищення безпеки DNS-сервера є його інтеграція з сервером віртуальної приватної мережі (VPN).

DNS-сервери є спокусливими мішенями для кіберзлочинців, враховуючи їх невід'ємну роль у перекладі доменних імен на IP-адреси. Тому використання вразливостей у цих серверів може призвести до викрадення домену, атак з ураженням кешу або порушень доступу до даних – ризиків, які організації можуть мінімізувати, використовуючи технології VPN для захисту DNS-сервера для встановлення більш стійкої присутності в Інтернеті.

Інтеграція VPN-серверів із DNS-серверами покращує шифрування та конфіденційність, створюючи безпечний тунель між DNS-запитами та відповідями, захищаючи їх від можливого перехоплення чи маніпулювання зловмисниками. Використовуючи переваги технології VPN, адміністратори DNS-серверів можуть забезпечити конфіденційність і цілісність свого трафіку та значно ускладнити його перехоплення або зміну зловмисниками.

Сервери VPN пропонують організаціям ще один рівень захисту від загроз мережевого рівня, спрямованих проти інфраструктури DNS. Перенаправляючи DNS-запити через сервер VPN, організації можуть ефективно захистити свої сервери від прямого впливу загальнодоступних інтернет-загроз, таких як об'ємні DDoS-атаки [66]. Діючи як поглинач або фільтр для поглинання потенційно зловмисного трафіку перед тим, як перенаправляти його на DNS-сервер, вони допомагають захистити від багатьох мережевих атак.

Сервер VPN пропонує адміністраторам захищений віддалений доступ до їх інфраструктури DNS, дозволяючи їм безпечно керувати та контролювати сервери з будь-якого місця. Підключившись до VPN-сервера своєї організації, вони можуть створювати зашифровані тунелі для доступу до DNS-серверів, що дозволяє їм виконувати важливі завдання, такі як оновлення конфігурації, розробку програмного забезпечення та моніторинг, більш безпечно, ніж за допомогою інших методів, таких як пряме відкриття інтерфейсів керування через Інтернет.

Організації, які планують інтегрувати VPN-DNS, повинні ретельно враховувати такі фактори, як моделі розгортання серверів, протоколи, такі як OpenVPN або IPsec [67], вимоги до керування сертифікатами та потреби в масштабуванні під час вибору свого рішення. Вибираючи рішення, вони повинні забезпечувати надійні механізми шифрування з надійними механізмами автентифікації, а також підтримувати безпечні протоколи тунелювання. Крім того, для підтримки безпечної інфраструктури необхідно проводити регулярний аудит.

Захист DNS-серверів за допомогою технології VPN забезпечує ефективний захист від потенційних атак на онлайн-інфраструктуру, підвищуючи загальний захист інфраструктури. З'єднавши VPN-сервер і DNS-сервер разом, організації можуть

захистити DNS-трафік, одночасно захищаючи від мережевих атак та забезпечуючи безпечний віддалений доступ. Технології VPN і DNS створюють ефективну комбінацію для зміцнення критичної інфраструктури DNS, захищаючи її стійкість від нових кіберзагроз. Інтеграція VPN-серверів у DNS-сервери залишається важливою для захисту онлайн-інфраструктури від потенційних зловмисників, які хочуть використати вразливості DNS.

2.4 Технології, що використані під час роботи

Перед початком створення прототипу, потрібно зазначити технології та методи, які будуть використані у розробці.

Docker [68] — це платформа програмного забезпечення з відкритим вихідним кодом, призначена для автоматизованого розгортання, керування та виконання програм у ізольованих контейнерах. Docker використовує технологію віртуалізації на рівні операційної системи у своїх контейнерах, щоб ізолювати програми одна від одної, а також захищати їхні залежності від системних проблем. Мета використання цієї технології – забезпечити віртуалізацію середовищ та імітувати сервери, які комунікують між собою у мережі однієї хостової системи.

Ключові поняття про Docker, які використані у реалізації моделі:

- Контейнери – забезпечують ізольоване середовище виконання, у якому додатки та залежності працюють без втручання з боку зовнішніх середовищ, використовуючи образи, визначені за допомогою файлів конфігурації, для формування самодостатніх одиниць, які працюють узгоджено в різних середовищах.
- Образи – це шаблони лише для читання, які визначають середовище та конфігурацію, необхідні для запуску певних програм або служб, включаючи компоненти ОС, бібліотеки та код програми. Файли Dockerfile містять інструкції, які використовують при будуванні образу.
- Dockerfile – це текстовий файл із інструкціями для створення образів Docker, наприклад, який базовий образ слід використовувати та які

встановлені залежності слід завантажити. Крім того, тут описано будь-які команди, необхідні під час процесу створення образу, і команди, необхідні під час запуску образів докерів з нуля. Файли Docker надають декларативний і відтворюваний спосіб налаштування середовища, необхідного для програми.

Docker Compose [69] — це інструмент, який дозволяє визначати багато-контейнерні програми Docker і керувати ними. Він використовує файл YAML [70] (docker-compose.yml) для налаштування служб, мереж і томів, необхідних для стека програми. За допомогою Docker Compose можна визначати зв'язки між різними контейнерами, налаштовувати змінні середовища та керувати загальною конфігурацією стека додатків.

Tunnelblick [71] — це безкоштовний графічний інтерфейс з відкритим кодом для OpenVPN, який забезпечує безпечні зашифровані з'єднання через Інтернет. OpenVPN став одним із найбільш широко використовуваних протоколів віртуальної приватної мережі (VPN) із понад 700 мільйонами інсталяцій по всьому світу, пропонуючи користувачам більше захисту від вразливостей у Інтернеті.

BIND [72] (Berkeley Internet Name Domain) — одне з найпоширеніших у світі програмних рішень DNS із відкритим вихідним кодом, яке надає основні інфраструктурні послуги. BIND може перетворювати доменні імена в IP-адреси для безперебійної роботи в глобальному Інтернет-середовищі.

Nslookup [73] — це утиліта командного рядка, призначена для запитів до системи доменних імен (DNS), збору інформації про домени, IP-адреси та записи DNS у більшості операційних систем, включаючи Windows, macOS і Linux.

2.5 Висновки до другого розділу

У другому розділі кваліфікаційної роботи досліджено використання серверів VPN для захисту DNS-серверів. Сервери DNS виконують важливу функцію, перетворюючи зрозумілі людині доменні імена в IP-адреси для Інтернет-зв'язку. Їх

функціонал робить їх основними цілями для кіберзлочинців, які прагнуть порушити роботу сервісів, здійснити атаки або отримати конфіденційні дані.

Організації можуть значно підвищити безпеку та конфіденційність своєї мережевої інфраструктури, використовуючи разом VPN і DNS-сервери для формування зашифрованого тунелю між пристроями користувачів і мережами, забезпечуючи конфіденційність, цілісність даних і знижуючи ризики, пов'язані з отруєнням кешу.

Однак слід пам'ятати, що успішний захист DNS-серверів за допомогою VPN-серверів залежить від ретельної реалізації та практики керування. Під час налаштування VPN і DNS-серверів організації повинні використовувати найкращі галузеві практики, наприклад використовувати надійні протоколи шифрування з надійними механізмами автентифікації та регулярними оновленнями для усунення відомих уразливостей. Крім того, необхідно запровадити механізми моніторингу/реєстрації, щоб виявити підозрілу діяльність, яка виникає в трафіку DNS, забезпечуючи своєчасні протидії.

Захист DNS-серверів за допомогою VPN-серверів забезпечує ефективну стратегію підвищення безпеки та конфіденційності мережевої інфраструктури. Застосовуючи цей комбінований підхід, організації можуть зменшити ризики, пов'язані з уразливістю DNS, одночасно зміцнюючи захист від атак.

РОЗДІЛ 3
СТВОРЕННЯ МОДЕЛІ ЗАХИСТУ СЕРВЕРУ ДОМЕННИХ ІМЕН

3.1 Планування архітектури застосунку.

Реалізація моделі використовує docker образи та контейнери, для віртуалізації середовищ виконання коду. Це дає перевагу на етапі конфігурації системи на новій машині, бо для усіх операційних систем не буде відрізнятись процес ініціалізації середовища.

Метою архітектурного рішення є забезпечити гнучкість розгортання програмного забезпечення на серверах. Використання системи віртуалізації docker і є головним принципом забезпечення цієї гнучкості, через те, що система імітує поведінку контейнеру, ніби вона знаходиться на відокремленому сервері, маючи власні ресурси.

Операційною системою для розробки моделі був обраний популярний дістрибутив Linux – Ubuntu.

Архітектура додатку передбачає два запущених контейнери. Перший – контейнер з запущеним VPN сервером OpenVPN. Він відповідає за тунелювання запитів клієнта та надання доступу у приватну мережу. Другий – контейнер з запущеним DNS сервером. Він відповідає за надання IP адресам приватної мережі власних доменних імен.

Головний принцип реалізації полягає у фільтрації запитів, які отримує DNS сервер. Потрібно налаштувати конфігурацію таким чином, щоб сервер був здатний опрацьовувати запити лише від авторизованого сервісу, яким виступає віртуальна приватна мережа.

Практичною перевіркою працездатності рішення є виконання команди nslookup “domain” localhost, де:

- domain – домен, IP адресу якого ми хочемо перевірити
- localhost – адреса DNS серверу. У даному випадку запит відбувається на локальне середовище.

У відповідь на команду, наведену вище, очікується оригінальна IP адреса серверу, домен якої було вказано. У разі, якщо клієнт не під'єднаний до мережі VPN, домен не повинен бути знайденим.

Зв'язок між контейнерами реалізований за допомогою вбудованої функції застосунку Docker – docker network adapter. Ця функція дозволяє поширити IP адреси та порти контейнерів між собою.

Під'єднання до мережі VPN усередині контейнеру реалізоване за допомогою застосунку з відкритим програмним кодом Tunnelblick. Воно дозволяє зручно завантажувати OpenVPN профілі та реалізовувати з'єднання з VPN сервером.

Реалізація передбачає функціонування системи у 10 етапів (рис 3.1):

1. Відправка nslookup запиту з операційної системи до Docker контейнеру.
2. Тунелювання запиту через VPN через застосунок Tunnelblick.
3. Отримання сигналу VPN контейнером.
4. Передача сигналу до DNS серверу через Docker network adapter.
5. Отримання сигналу DNS сервером.
6. Пошук доменної адреси у довіднику.
7. У разі знайденої адреси – передача сигналу до VPN серверу через Docker network adapter.
8. Отримання сигналу адаптером, передача сигналу до VPN серверу.
9. Отримання сигналу VPN сервером, передача сигналу до клієнта.
10. Відповідь клієнту на отриманий запит.

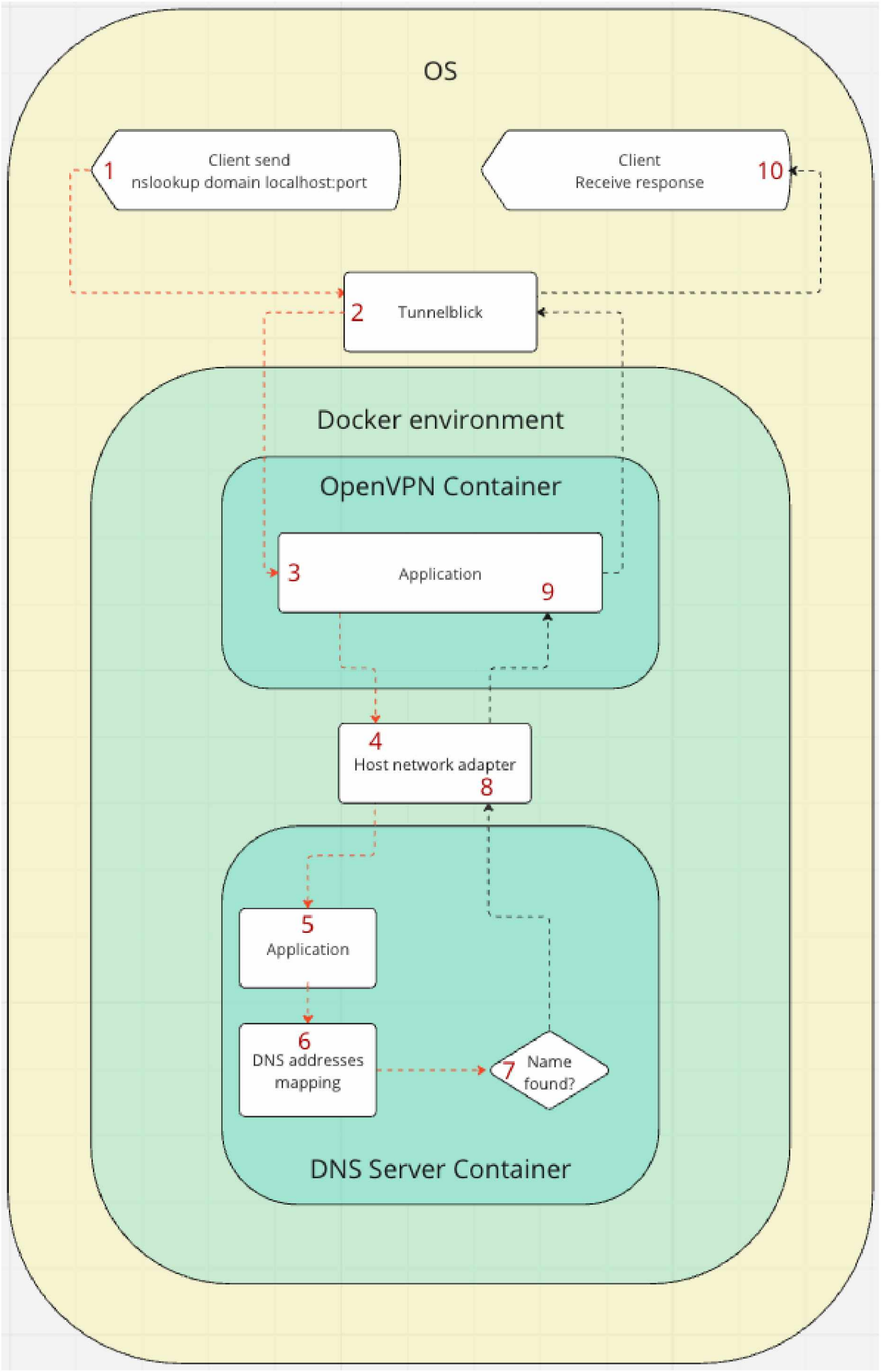

Рис. 3.1 Діаграма архітектури застосунку.

3.2 Створення контейнеру VPN сервера.

Запуск OpenVPN у контейнері Docker пропонує кілька переваг, зокрема легше розгортання, ізоляцію, портативність і можливості масштабування. Завдяки контейнеризації серверів OpenVPN та їхніх залежностей в окремих середовищах легше керувати чи розподіляти – цьому підходу сприяє його портативність.

Мережа OpenVPN передбачає налаштування та керування мережевими з'єднаннями між сервером і клієнтами, причому OpenVPN пропонує гнучке, але безпечне встановлення віртуальної приватної мережі в загальнодоступних мережах, щоб забезпечити безпечний зв'язок і доступ до ресурсів.

Ключовими аспектами налаштування мережі OpenVPN є:

- Тунелювання. OpenVPN створює зашифрований тунель між своїм сервером і клієнтами, загортаючи трафік VPN в інший протокол (зазвичай Transport Layer Security), захищаючи всі передачі між ними від потенційних перехоплювачів. Це гарантує конфіденційність даних під час з'єднання.
- Трансляція мережевих адрес. Клієнтам дозволено отримувати доступ до ресурсів під час встановлення контакту з інтернет-ресурсами через з'єднання з інтернет-сервером OpenVPN. Крім того, OpenVPN може налаштувати NAT [74] на собі, щоб маскувати або пересилати трафік між своєю підмережею та будь-якими зовнішніми мережами.
- Маршрутизація. OpenVPN пропонує численні конфігурації для контролю мережевого трафіку між своїм сервером і клієнтами, такі як налаштування статичних маршрутів або надсилання маршрутів безпосередньо клієнтам. Також підтримується використання динамічних протоколів, таких як OSPF [75] (open shortest path first), для обміну інформацією про динамічну маршрутизацію.

Створення серверу OpenVPN також стосується процесу створення всіх необхідних файлів і конфігурацій, необхідних для налаштування та роботи сервера

або клієнта OpenVPN, включаючи сертифікати SSL/TLS, криптографічні ключі та файли конфігурації, специфічні для цього екземпляра OpenVPN.

Для створення контейнеру, потрібно звернутись до вже існуючих рішень контейнеризації OpenVPN сервісу. Вимоги до реалізації: рішення повинно бути гнучким.

Створимо директорію, де будуть розміщені файли конфігурації контейнеру та виконаємо команду клонування репозиторію за допомогою утиліти контролю версій git (рис 3.2).

```
~ $ mkdir vpn-dns-server
~ $ cd vpn-dns-server/
~/vpn-dns-server $ git clone https://github.com/kylemanna/docker-openvpn.git
Cloning into 'docker-openvpn'...
remote: Enumerating objects: 1732, done.
remote: Total 1732 (delta 0), reused 0 (delta 0), pack-reused 1732
Receiving objects: 100% (1732/1732), 385.19 KiB | 234.00 KiB/s, done.
Resolving deltas: 100% (1006/1006), done.
```

Рис 3.2 Створення директорії та клонування репозиторію.

Виконаємо команду tree, для того щоб перевірити зміст репозиторію та переконатись в успішному клонуванні (рис 3.3).

```
~/vpn-dns-server [127]$ tree
.
└── docker-openvpn
    ├── Dockerfile
    ├── Dockerfile.aarch64
    ├── bin
    │   ├── ovpn_copy_server_files
    │   ├── ovpn_genconfig
    │   ├── ovpn_getclient
    │   ├── ovpn_getclient_all
    │   ├── ovpn_initpki
    │   ├── ovpn_listclients
    │   ├── ovpn_otp_user
    │   ├── ovpn_revokeclient
    │   ├── ovpn_run
    │   └── ovpn_status
    ├── init
    │   ├── docker-openvpn@.service
    │   └── upstart.init
    └── otp
        └── openvpn

5 directories, 15 files
~/vpn-dns-server $
```

Рис 3.3 Зміст директорії vpn-dns-server.

Директорія склонувалась правильно. Наступним кроком потрібно перевірити зміст файлу Dockerfile (рис 3.4). Від нього залежить налаштування контейнеру, в якому буде розгорнутий сервер VPN.

```
FROM alpine:latest

RUN echo "http://dl-cdn.alpinelinux.org/alpine/edge/testing/" >> /etc/apk/repositories && \
    apk add --update openvpn iptables bash easy-rsa openvpn-auth-pam google-authenticator pamtester libqrencode && \
    ln -s /usr/share/easy-rsa/easyrsa /usr/local/bin && \
    rm -rf /tmp/* /var/tmp/* /var/cache/apk/* /var/cache/distfiles/*

ENV OPENVPN=/etc/openvpn
ENV EASYRSA=/usr/share/easy-rsa \
    EASYRSA_CRL_DAYS=3650 \
    EASYRSA_PKI=$OPENVPN/pki

VOLUME ["/etc/openvpn"]

EXPOSE 1194/udp

CMD ["ovpn_run"]

ADD ./bin /usr/local/bin
RUN chmod a+x /usr/local/bin/*

ADD ./otp/openvpn /etc/pam.d/
```

Рис 3.4 Зміст файлу Dockerfile.

Бачимо, що Dockerfile містить декілька команд конфігурації контейнеру. Розберемо більш детально кожен рядок:

- **FROM alpine:latest** – відповідає за базовий образ операційної системи, яка буде створена у контейнері. У даному випадку – це операційна система linux alpine, яка є найбільш легковісною серед популярних дистрибутвів Linux.
- **RUN echo …** - рядок, який відповідає за завантаження базових утиліт, необхідних для реалізації OpenVPN серверу, серед яких є консольний інтерфейс bash, утиліта моніторінгу мережевих інтерфейсів iptables, безпосередньо утиліта openvpn тощо.
- **ENV …** – рядок, який відповідає за створення змінних середовища. Встановлюється коренева директорія для OpenVPN, та змінні для коректної роботи бібліотеки EasyRSA [76], яка використовується для створення RSA ключів.

- **VOLUME** – параметр, який визначає директорію з хостової системи, яка буде використовуватись для роботи усередині контейнеру.
- **EXPOSE 1194/udp** – відповідає за відкриття порту 1194 у контейнері для приймання UDP пакетів. Як зазначено у теоретичній частині роботи, OpenVPN працює саме за stateless [77] архітектурою мережевих підключень.
- **CMD ovpn_run** – відповідає за запуск процесу OpenVPN з консолі контейнеру.
- **ADD ./bin /usr/local/bin** – відповідає за клонування директорії /bin до контейнера у директорію /usr/local/bin
- **RUN chmod a+x /usr/local/bin/*** – надає права на запис до усіх файлів директорії /usr/local/bin

Після переконання, що репозиторій є цілісним та має усі необхідні файли для запуску, потрібно збудувати образ Docker за допомогою команди docker build vpn-server (рис 3.5).

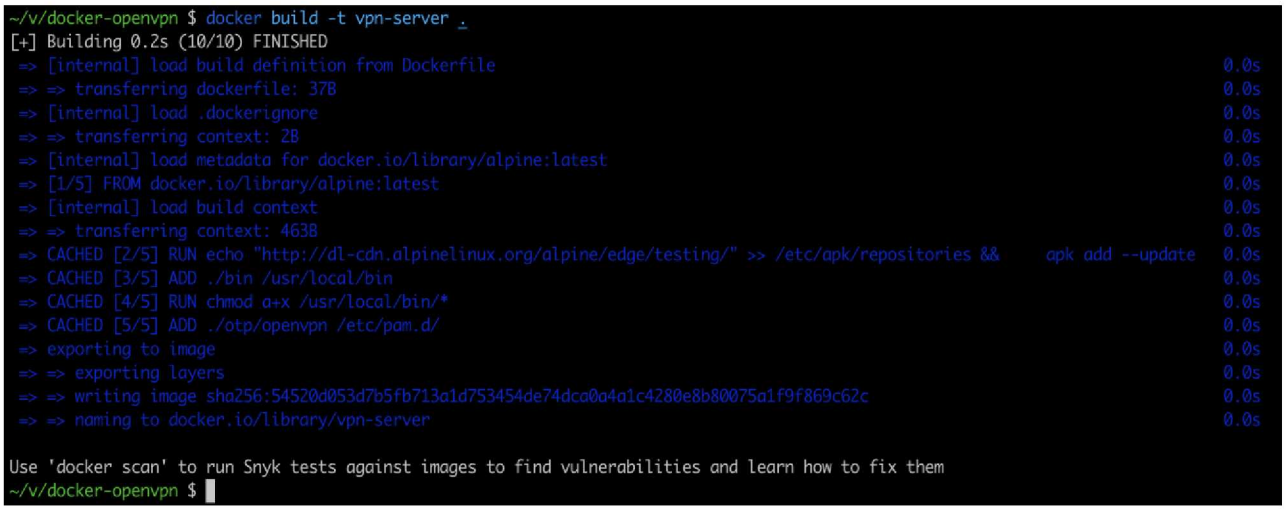

Рис 3.5 Процес будування образу Docker за допомогою команди docker build.

Після успішної побудови образу, треба додати ще декілька файлів для коректної роботи сервісу, зокрема, директорію vpn-data та файл vars для збереження змінних під час роботи серверу (рис 3.6).

```
~/v/docker-openvpn $ cd ..
~/vpn-dns-server $ mkdir vpn-data
~/vpn-dns-server $ touch vpn-data/vars
~/vpn-dns-server $ cd vpn-data/
~/v/vpn-data $ tree
.
└── vars

1 directory, 1 file
~/v/vpn-data $
```

Рис 3.6 Зміст директорії vpn-data.

Після створення необхідних файлів можна створювати конфігурацію для OpenVPN серверу. Реалізуємо це за допомогою команди звернення до Docker контейнеру, який ми попередньо збудували (рис 3.7).

```
~/v/vpn-data $ cd ..
~/vpn-dns-server $ docker run -v $PWD/vpn-data:/etc/openvpn --rm vpn-server ovpn_genconfig -u udp://localhost:3000
Processing PUSH Config: 'block-outside-dns'
Processing Route Config: '192.168.254.0/24'
Processing PUSH Config: 'dhcp-option DNS 8.8.8.8'
Processing PUSH Config: 'dhcp-option DNS 8.8.4.4'
Processing PUSH Config: 'comp-lzo no'
Successfully generated config
Cleaning up before Exit ...
~/vpn-dns-server $
```

Рис 3.7 Генерація конфігураційного файлу OpenVPN.

Деталі виконання команди:

- *docker run -v* – запуск Docker контейнеру у режимі відладки, тобто, з усім можливим логуванням процесу.
- *$PWD/vpn-data:/etc/openvpn* – визначаємо яка директорія буде відповідати за середу виконання команди.
- *--rm* – видалити контейнер після виконання.
- *vpn-server* – назва образу, що запускається.
- *ovpn_getconfig* – команда створення конфігураційного файлу OpenVPN.
- *udp://localhost:3000* – опція яка вказує на якому хості та який протокол буде приймати сервіс.

У виведеній інформації у термінал бачимо, що файл був успішно згенерований.

Наступним етапом є генерація PKI [63] системи. (рис 3.8) PKI (інфраструктура відкритих ключів) відіграє незамінну роль у захисті підключень віртуальної приватної мережі (VPN). Система PKI включає технології, протоколи та практики, які сприяють безпечному управлінню, розповсюдженню та використанню криптографії з відкритим ключем. PKI (інфраструктура відкритих ключів) може застосовуватися в середовищах VPN для автентифікації, шифрування та керування ключами. PKI забезпечує безпечний зв'язок у VPN за допомогою шифрування. Після початкової автентифікації VPN-сервер і клієнти встановлюють безпечний сеанс за допомогою симетричного шифрування. Їхній сеансовий ключ шифрування або дешифрування може потім безпечно обмінюватися між учасниками, використовуючи як відкритий, так і закритий ключі, гарантуючи, що всі передані дані залишаються конфіденційними та захищеними від сторонніх очей.

```
~/vpn-dns-server $ docker run -v $PWD/vpn-data:/etc/openvpn --rm -it vpn-server ovpn_initpki

Notice
------
'init-pki' complete; you may now create a CA or requests.

Your newly created PKI dir is:
* /etc/openvpn/pki

* Using Easy-RSA configuration:

* IMPORTANT: Easy-RSA 'vars' template file has been created in your new PKI.
             Edit this 'vars' file to customise the settings for your PKI.
             To use a global vars file, use global option --vars=<YOUR_VARS>

* Using x509-types directory: /usr/share/easy-rsa/x509-types


* Using SSL: openssl OpenSSL 3.1.0 14 Mar 2023 (Library: OpenSSL 3.1.0 14 Mar 2023)

* Using Easy-RSA configuration: /etc/openvpn/pki/vars


Enter New CA Key Passphrase:
```

Рис 3.8 Генерація PKI для OpenVPN серверу.

Деталі виконання команди:

- *docker run -v* - запуск Docker контейнеру у режимі відладки, тобто, з усім можливим логуванням процесу.

- *$PWD/vpn-data:/etc/openvpn* – визначаємо яка директорія буде відповідати за середу виконання команди.
- *--rm* – видалити контейнер після виконання.
- *-it* – запуск контейнеру у інтерактивному режимі.
- *vpn-server* – назва образу, що запускається.
- *ovpn_initpki*– команда створення конфігураційного файлу OpenVPN.

Після виконання команди, сервер запитує у нас ключ для генерації сертифікату. Оскільки це тестове середовище, ключом було обрано комбінацію 1111 (рис 3.9).

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:KarimFIT

Notice
------
CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/etc/openvpn/pki/ca.crt
```

Рис 3.9 Генерація сертифікату OpenVPN.

Також, було запитано ключове слово для генерації сертифікату. Було обрана комбінація KarimFIT.

Бачимо, що скрипт згенерував нам ключі та сертифікат (рис 3.9).

На цьому етапі наш сервер готовий до запуску. Тепер потрібно провести консольну команду для ініціалізації серверу усередині контейнера (рис 3.10).

```
~/vpn-dns-server $ docker run -v $PWD/vpn-data:/etc/openvpn -d -p 3000:1194/udp --cap-add=NET_ADMIN vpn-server
26e81bfd76ddcb0387ebbc7fdab8e18ed611811019f85fc249e4e037a05b5002
~/vpn-dns-server $
```

Рис 3.10 Запуск серверу усередині контейнеру.

Деталі виконання команди:

- *docker run -v* - запуск Docker контейнеру у режимі відладки, тобто, з усім можливим логуванням процесу.
- *$PWD/vpn-data:/etc/openvpn* – визначаємо яка директорія буде відповідати за середу виконання команди.
- *-d* –запуск контейнеру у фоновому режимі.
- *-p 3000:1194/udp* – специфікація протоколів та портів, які будуть використані контейнером.
- *--cap-add=NET_ADMIN* – надання контейнеру дозволу керування мережевим інтерфейсом хостової машини.
- *vpn-server* – назва образу, що запускається.

Після виконання бачимо у консолі вивід хешу контейнеру, який є його унікальним ідентифікатором. Щоб переконатись у тому, що контейнер запущений, потрібно виконати команду docker stats, яка відобразить статистику запущених контейнерів, а саме їх використані ресурси (рис 3.11).

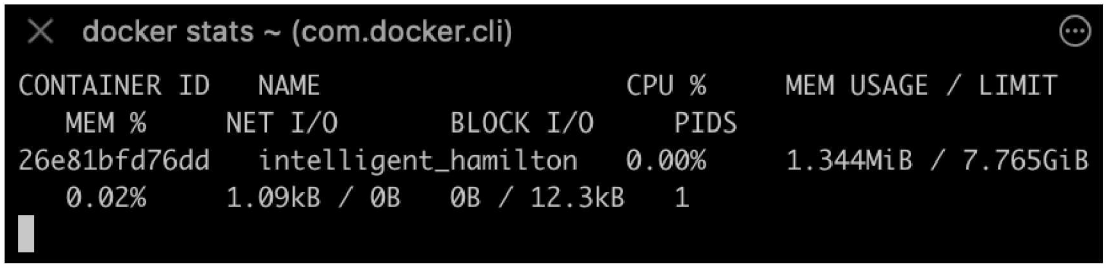

Рис 3.11 Вивід команди docker stats.

На малюнку бачимо, що контейнер використовує мінімум ресурсів CPU та RAM. Назва контейнеру була автоматично сгенерована самим Docker – intelligent_hamilton. Також бачимо у колонці PIDS, що контейнер має один запущений процес. Це і є OpenVPN сервер.

Для того, щоб під'єднатись до серверу, потрібно згенерувати файли конфігурації для клієнтів. Для цього виконаємо команду звернувшись до контейнеру (рис 3.12).

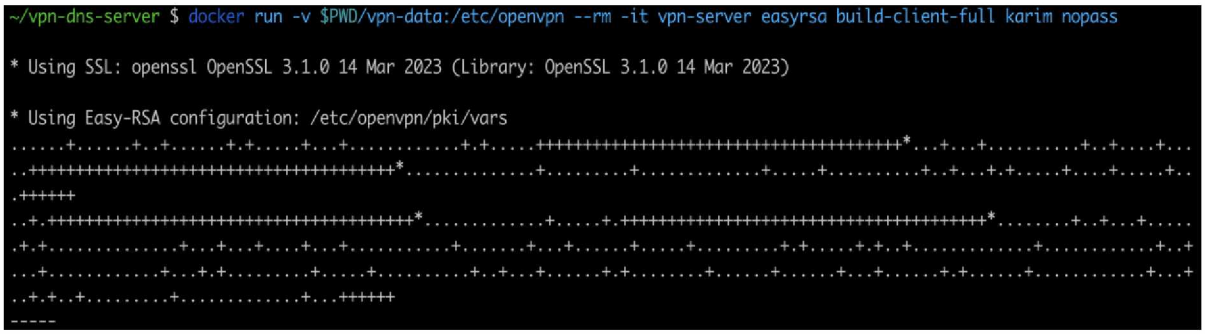

Рис 3.12 Генерація файлу конфігурації для клієнта

Деталі виконання команди:

- *docker run -v* - запуск Docker контейнеру у режимі відладки, тобто, з усім можливим логуванням процесу.
- *$PWD/vpn-data:/etc/openvpn* – визначаємо яка директорія буде відповідати за середу виконання команди.
- *--rm* – видалити контейнер після виконання.
- *-it* – запуск контейнеру у інтерактивному режимі.
- *vpn-server* – назва образу, що запускається.
- *easyrsa build-client-full* – команда створення конфігураційного файлу клієнта.
- *karim nopass* – визначення паролю та логіну для клієнта.

Далі потрібно згенерувати ovpn файл з отриманої конфігурації до користувача karim. Робимо це звернувшись до контейнеру (рис 3.13).

Рис 3.13 Виконання команди створення ovpn файлу

Деталі виконання команди:

- *docker run -v* - запуск Docker контейнеру у режимі відладки, тобто, з усім можливим логуванням процесу.
- *$PWD/vpn-data:/etc/openvpn* – визначаємо яка директорія буде відповідати за середу виконання команди.
- *--rm* – видалити контейнер після виконання.

- *vpn-server* – назва образу, що запускається.
- *ovpn_getclient karim > karim.ovpn* – команда створення ovpn файлу та запис його на хостову машину.

Після виконання команди, отримуємо файл конфігурації для користувача karim (рис 3.14).

Рис 3.14 Сгенерований файл karim.ovpn

Тепер наш сервер готовий до підключення. Реалізуємо підключення за допомогою утіліти Tunnelblick. Для цього, просто перетягнемо цей файл до іконки запущеної програми у навігаційній панелі, та введемо пароль хостової машини (рис 3.15).

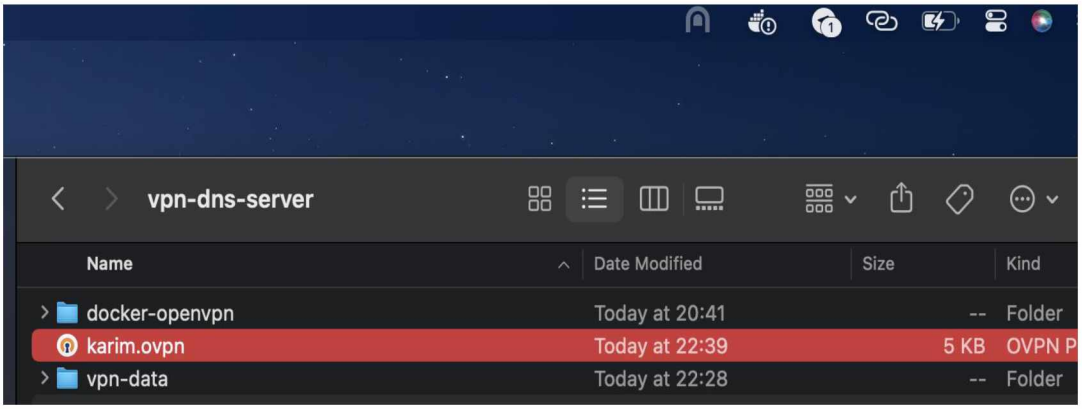

Рис 3.15 Конфігурація Tunnelblick за допомогою сгенерованого ovpn файлу

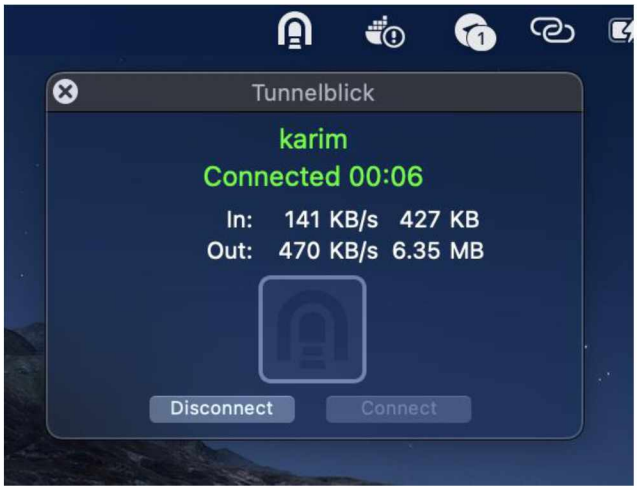

Рис 3.16 Підключений клієнт до серверу OpenVPN

Бачимо, що клієнт був успішно підключений до серверу (рис 3.16). Для того, щоб перевірити підключення на боці серверу, потрібно перейти до контейнеру та перевірити файл логування, який за замовчуванням знаходиться у директорії /tmp/openvpn-status.log.

```
~/vpn-dns-server [126]$ docker exec -it 26e81bfd76dd "bash"
26e81bfd76dd:/# cat /tmp/openvpn-status.log
OpenVPN CLIENT LIST
Updated,2023-05-21 21:48:57
Common Name,Real Address,Bytes Received,Bytes Sent,Connected Since
karim,254.254.0.1:62930,79656426,15669091,2023-05-21 21:46:13
ROUTING TABLE
Virtual Address,Common Name,Real Address,Last Ref
192.168.255.6,karim,254.254.0.1:62930,2023-05-21 21:48:56
GLOBAL STATS
Max bcast/mcast queue length,0
END
26e81bfd76dd:/#
```

Рис 3.17 Файл логування статусу OpenVPN серверу

Бачимо, що до серверу успішно під'єднався користувач з іменем karim, та успішно надіслав 15669091 байтів даних (рис 3.17). Це може свідчити, що підключення успішно встановлене.

3.4 Створення контейнеру DNS сервера.

Наступним кроком у реалізації моделі є створення DNS сервера. Спосіб реалізації сервера було обрано також за допомогою використання Docker.

Docker compose дозволяє створювати сервіси у межах одного репозиторію, що дозволяє одночасну зручну конфігурацію кожного окремого сервісу.

Для того, щоб реалізувати DNS сервер, потрібно створити docker-compose.yml файл. Для цього, створимо нову директорію та ініціалізуємо файл (рис 3.18).

```
~/vpn-dns-server $ mkdir dns
~/vpn-dns-server $ cd dns
~/v/dns $ touch docker-compose.yml
~/v/dns $
```

Рис 3.18 Створення директорії та створення docker-compose.yml файлу.

dns > docker-compose.yml

```yaml
version: "3"

services:
  bind9:
    container_name: dns
    image: ubuntu/bind9:latest
    environment:
      - BIND9_USER=root
      - TZ=Europe/Kyiv
    ports:
      - "53:53/tcp"
      - "53:53/udp"
    volumes:
      - ./config:/etc/bind
      - ./cache:/var/cache/bind
      - ./records:/var/lib/bind
    restart: unless-stopped

```

Рис 3.19 Зміст файлу docker-compose.yml

Файл docker-compose.yml (рис 3.19) є YAML конфігурацією для контейнерів, які ініціалізовані за допомогою Docker. Файл конфігурації контейнеру DNS серверу налічує наступні параметри:

- *version* – версія docker-compose.yml файлу.
- *services* – перелік контейнерів, які будуть запущені.
- *bind9* – назва контейнеру, яка використовується іншими контейнерами.
- *container_name* – назва контейнеру, яка використовується розробником.
- *image* – образ, який використовується контейнером. У даному прикладі це образ ubuntu версії bind9
- *environment* – змінні віртуального середовища. Виставляємо часовий пояс та користувача для образу bind9.
- *ports* – перелік портів, які встановлені відкритими для з'єднання. Виставляємо порт 53 на прийняття tcp та udp пакетів.

- *volumes* – директорії, які будуть використані контейнером, взяті з хостової машини.
- *restart* – умова для автоматичної перезагрузки контейнеру.

Для того, щоб сконфігурувати наш DNS сервер на отримання запитів по певним доменим адресам, потрібно створити директорію config, та ініціалізувати два файла. Один – для зонування DNS серверу, інший – для конфігурації дозволених з'єднань (рис 3.20).

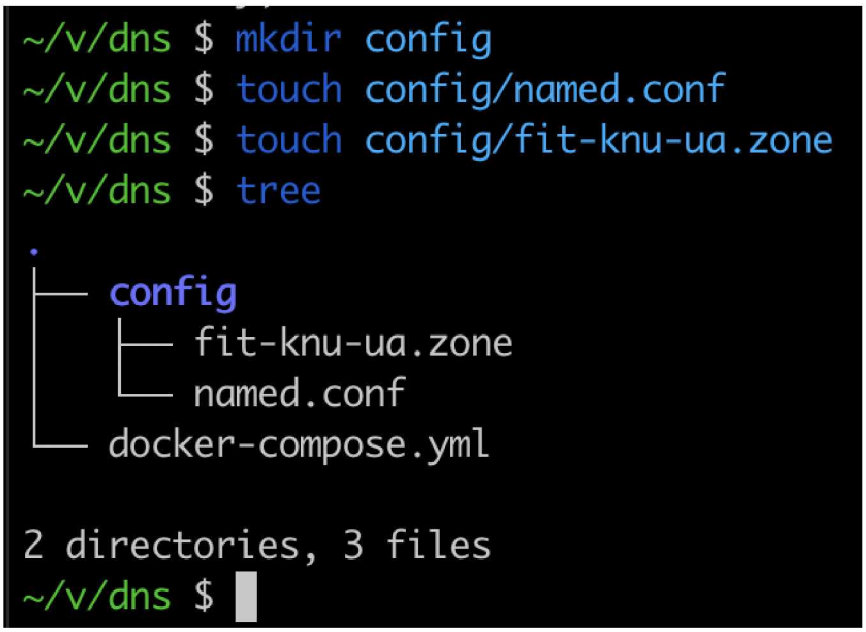

Рис 3.20 Створення конфігураційних файлів DNS серверу

Для того, щоб наш сервер міг приймати підключення, потрібно виставити перелік дозволених IP адрес, з яких можуть надходити запити. Для цього, сконфігуруємо файл named.conf (рис 3.21).

```
dns > config > named.conf
1   acl internal {
2     localhost;
3     any;
4   };
5
6   options {
7     forwarders {
8       1.1.1.1;
9       1.0.0.1;
10    };
11    allow-query {
12      internal;
13    };
14  };
15
16  zone "fit.knu.ua" IN {
17    type master;
18    file "/etc/bind/fit-knu-ua.zone";
19  };
```

Рис 3.21 Конфігурація файлу named.conf

У конфігурації потрібно виставити наступні параметри:

- *acl internal* – створення змінної internal, з переліком можливих IP адрес. Так як це тестове середовище, встановимо параметр any, щоб приймати запити з будь-якої клієнтської IP адреси.
- *options* – блок конфігурації підключення.
- *forwarders* – IP адреси DNS серверів, які будуть використані у разі якщо адреса не буде знайдена.
- *allow-query* – встановлення параметрів підключення спеціальних IP адрес
- *zone* – вказівник на файл конфігурації домену fit.knu.ua

Наступним кроком потрібно сконфігурувати fit-knu-ua.zone файл для того, щоб створити доменні імена та надати їм IP адреси (рис 3.22).

dns > config > fit-knu-ua.zone

```
$TTL 2d

$ORIGIN fit.knu.ua.

@             IN     SOA    cyber.fit.knu.ua. cyber.knu.ua (
                                2023052200     ; serial
                                1h            ; refresh
                                10m            ; retry
                                4w             ; expire
                                1h )           ; minimum TTL

              IN     NS     cyber.fit.knu.ua.

cyber            IN     A      10.20.3.3

cybersecurity    IN     A      10.20.3.5
*.cybersecurity  IN     A      10.20.3.6

```

Рис 3.22 Файл конфігурації fit-knu.ua.zone

У конфігурації потрібно виставити наступні параметри:

- *TTL* – time-to-live – час життя одного пакету, який не був доставлений до отримувача
- *ORIGIN* – вказує на початок конфігурації певного домену

Наступним кроком потрібно вказати IP адреси для кожного варіанту домену, який:

- Починається з cyber – 10.20.3.3
- Починається з cybersecurity– 10.20.3.5
- Закінчується cyber-info – 10.20.3.6

Після конфігурації файлу, потрібно перейти до директорії /dns та виконати команду створення контейнерів docker-compose (рис 3.23).

```
~/v/dns $ docker-compose up
[+] Running 4/4
 ⠿ bind9 Pulled                                                    11.2s
   ⠿ 75ff66e3d15c Pull complete                                     6.6s
   ⠿ 1613f7670919 Pull complete                                     7.6s
   ⠿ a969c60eee56 Pull complete                                     7.7s
[+] Running 2/2
 ⠿ Network dns_default  Created                                     0.1s
 ⠿ Container dns        Created                                     0.6s
```

Рис 3.23 Створення контейнеру DNS сервера

```
dns  | 22-May-2023 02:00:26.483 all zones loaded
dns  | 22-May-2023 02:00:26.484 running
dns  | 22-May-2023 02:00:26.484 address not available resolving './NS/IN': 2001:dc3::35#53
dns  | 22-May-2023 02:00:26.484 address not available resolving './NS/IN': 2001:500:200::b#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:2f::f#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:2d::d#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:2::c#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:1::53#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:503:c27::2:30#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:7fd::1#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:a8::e#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:503:ba3e::2:30#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:12::d0d#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:7fe::53#53
dns  | 22-May-2023 02:00:26.485 address not available resolving './NS/IN': 2001:500:9f::42#53
dns  | 22-May-2023 02:00:26.979 managed-keys-zone: Key 20326 for zone . is now trusted (acceptance timer complete)
dns  | 22-May-2023 02:00:26.987 resolver priming query complete: success
```

Рис 3.24 Логи DNS серверу усередині Docker контейнера

Логи контейнера відображають, що сервер був запущений та готовий до приймання запитів (рис 3.24).

Протестуємо відправку запитів за допомогою утіліти nslookup.

```
~ $ nslookup cyber.fit.knu.ua localhost
Server:         localhost
Address:        ::1#53

Name:   cyber.fit.knu.ua
Address: 10.20.3.3

~ $ nslookup cybersecurity.fit.knu.ua localhost
Server:         localhost
Address:        ::1#53

Name:   cybersecurity.fit.knu.ua
Address: 10.20.3.5
```

Рис 3.25 Запит на отримання IP адреси домену cyber.fit.knu.ua

З отриманих результатів (рис 3.25) можна зробити висновок, що DNS сервер працює та відповідає правильними IP адресами на передані домени.

3.5 Захист доменів, розташованих у периметрі VPN серверу.

Наступним етапом роботи є конфігурація DNS серверу таким чином, щоб він отримував запити лише від авторизованого джерела. У даному випадку авторизованим джерелом виступає VPN сервер.

При підключенні до VPN серверу, клієнт змінює свою публічну IP адресу на IP адресу серверу. З цього можна зробити висновок, що DNS сервер повинен приймати запити лише з IP адреси VPN сервера.

Для початку потрібно дізнатись, яку IP адресу має контейнер з запущеним VPN сервером. Для цього, виконаємо команду звернувшись до модулю inspect утіліли docker, та передавши унікальний ідентифікатор контейнера (рис 3.26).

```
~ $ docker inspect \
          -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 26e81bfd76dd
254.254.0.2
~ $
```

Рис 3.26 IP адреса контейнеру VPN сервера

Деталі виконання команди:

- *docker inspect* – команда, яка дозволяє переглянути мета дані контейнера.
- *-f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'* – шлях до IP адреси у конфігурації контейнера.
- *26e81bfd76dd* – унікальний ідентифікатор контейнера.

Бачимо, що VPN сервер має свою IP адресу – 254.254.0.2.

Для того, щоб сконфігурувати DNS сервер на приймання запитів лише з певної IP адреси, потрібно переглянути та змінити файл named.conf (рис 3.27).

```
dns > config > named.conf
1   acl internal {
2     254.254.0.2;
3   };
4
5   options {
6     forwarders {
7       1.1.1.1;
8       1.0.0.1;
9     };
10    allow-query {
11      internal;
12    };
13  };
14
15  zone "fit.knu.ua" IN {
16    type master;
17    file "/etc/bind/fit-knu-ua.zone";
18  };
```

Рис 3.27 Змінений файл named.conf. Додана IP адреса

Після додавання IP адреси у параметр дозволених IP адрес з яких DNS сервер приймає запити, потрібно перезавантажити сервер. Це можна зробити за допомогою комбінації Ctrl + C у терміналі docker контейнеру (рис 3.28).

```
dns  | 22-May-2023 02:10:15.245 managed-keys-zone: Key 20326 for zone . is now trusted (acceptan
ce timer complete)
^CGracefully stopping... (press Ctrl+C again to force)
[+] Running 1/1
 ⠿ Container dns  Stopped                                                          0.3s
canceled

~/v/dns [SIGINT]$ docker-compose up
[+] Running 1/0
 ⠿ Container dns  Created                                                          0.0s
Attaching to dns
dns  | Starting named...
```

Рис 3.28 Перезапуск docker контейнеру з DNS сервером.

Після перезапуску, протестуємо запит до DNS серверу без підключеного VPN. Для цього, виконаємо команду nclookup cybersecurity localhost (рис 3.29).

```
~ $ nslookup cybersecurity.fit.knu.ua localhost
Server:         localhost
Address:        ::1#53

** server can't find cybersecurity.fit.knu.ua: REFUSED

~ [1]$
```

Рис 3.29 Відповідь DNS сервера при відключенному VPN

DNS сервер розірвав підключення, не надавши відповідь клієнту, як і очікувалось. У логах DNS серверу можна побачити наступне (рис 3.30)

```
dns  | 22-May-2023 02:27:17.659 managed-keys-zone: Key 20326 for zone . is now trusted (acceptance timer complete)
dns  | 22-May-2023 02:27:17.662 resolver priming query complete: success
dns  | 22-May-2023 02:27:20.138 client @0xffff54002878 10.254.4.1#60210 (cybersecurity.fit.knu.ua): query 'cybersecurity.fit.knu.ua/A/IN' denied
```

Рис 3.30 Логи DNS сервера

DNS сервер каже про невідому IP адресу та розірване підключення через невідомий серверу клієнт.

Спробуємо під'єднатись до VPN серверу за допомогою утиліти Tunnelblick та попередньо згенерований файл конфігурації для користувача karim. Слід пам'ятати, що у момент підключення, локальна IP адреса зміниться на IP адресу Docker контейнеру (рис 3.31).

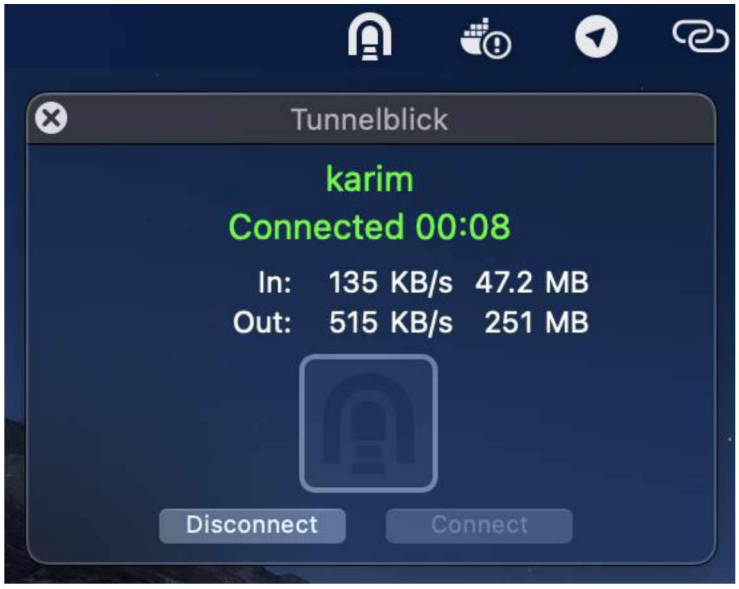

Рис 3.31 Підключення до утиліти Tunnelblick

Після підключення, виконаємо команду nslookup (рис 3.32).

```
~ [1]$ nslookup cybersecurity.fit.knu.ua localhost
Server:         localhost
Address:        ::1#53

Name:   cybersecurity.fit.knu.ua
Address: 10.20.3.5

~ $
```

Рис 3.32 Виконання команди nslookup

Сервер відповідає IP-адресою домену, яка була сконфігурована попередньо під час будування DNS серверу. З цього можна зробити висновок, що DNS сервер відображає IP адресу домену тільки у разі успішного підключення клієнта до VPN серверу, що накладує додатковий рівень безпеки до ресурсів, розміщених у мережі Інтернет.

3.6 Рекомендації до реалізації захиту DNS сервера

Вибираючи надійне рішення VPN для захисту DNS-сервера, потрібно врахувати такі фактори, як міцність шифрування, механізми автентифікації та сумісність із існуючою інфраструктурою. OpenVPN або Wireguard забезпечують надійні функції безпеки з широким аудитом кібербезпеки, що проводиться сторонніми компаніями.

Потрібно налаштовувати свій VPN-сервер таким чином, щоб встановлювати безпечні з'єднання між клієнтами та DNS-сервером, використовуючи надійні алгоритми шифрування, такі як AES (Advanced Encryption Standard) для захисту даних під час передачі, одночасно правильно налаштовуючи протоколи та параметри, пов'язані з обміном ключами та механізми, щоб гарантувати безпечний зв'язок між клієнтськими пристроями та сервером DNS.

Створення ефективної мережевої архітектури для ефективного поєднання VPN-сервера з DNS-сервером потребує уваги розміщення VPN-сервера відносно інших компонентів у мережевому середовищі. Потрібно переконатись, що до DNS-сервера

можна отримати доступ лише з середовища VPN-сервера. Можливо, на цьому етапі потрібно застосувати належні параметри брандмауера, щоб обмежити несхвалені підключення безпосередньо за межами середовища VPN-сервера.

Рекомендується застосувати заходи контролю доступу, щоб обмежити доступ до VPN і DNS-серверів лише для авторизованого персоналу, використовуючи надійні унікальні паролі або двофакторну автентифікацію для доступу до VPN-сервера та контроль доступу на основі ролей (RBAC), щоб визначити та застосувати привілеї доступу для адміністраторів, які керують DNS сервер. Також, регулярно перевіряти та оновлювати засоби контролю доступу, щоб зберегти цілісність системи.

Моніторинг і журналювання – надійні механізми стеження за діяльністю серверів VPN і DNS. Моніторинг з'єднань VPN, є способом виявлення будь-яких аномалій, які можуть стати інцидентами безпеки. За потреби, є можливість реєструвати відповіді DNS у режимі реального часу для цілей аудиту або усунення несправностей і регулярно аналізувати журнали на пошук підозрілої активності чи ознак неавторизації. Відстежувати підозрілі підключення за допомогою аналітичного програмного забезпечення.

Важливо, щоб серверне програмне забезпечення VPN і DNS залишалося в актуальному стані з виправленнями безпеки та оновленнями від постачальників. Проведення регулярних перевірок на наявність можливих недоліків у безпеці або вразливості є необхідним заходом безпеки будь якого підприємства.

Рекомендується проводити регулярні аудити безпеки та тести на проникнення, щоб виявити будь-які слабкі місця чи потенційні вразливості в архітектурі VPN-DNS. Консультуватися з професійними експертами з безпеки або проводити внутрішні оцінки, щоб оцінити загальний стан безпеки, ефективність реалізовану архітектуру, а також усувати вразливості, виявлені під час оцінювання, щойно вони виникають, і проводити періодичне тестування, щоб забезпечити постійну стійкість системи.

Розгортання архітектури, де DNS-сервер приховано за VPN-серверами, пропонує організаціям додатковий рівень захисту для посилення конфіденційності, цілісності та доступності інфраструктури DNS. Дотримуючись наведених тут рекомендацій, компанії можуть підвищити безпеку своїх служб. Рекомендується

обирати надійні рішення VPN із захищеними з'єднаннями VPN, які відповідають критеріям безпеки. Також, розробити відповідну мережеву архітектуру, яка включає надійні засоби контролю доступу з регулярним моніторингом активності сервера, підтримуючи програмне забезпечення в актуальному стані, проходячи регулярні аудити безпеки, гарантуючи, що сервера залишаються захищеними від потенційних атак.

3.7 Висновки до третього розділу

У рамках третього розділу кваліфікаційної роботи, була розроблена система для захисту серверів DNS і VPN за допомогою технології контейнеризації Docker. Разом ці технології створюють ефективний захист від потенційних вразливостей і загроз мережевої інфраструктури. Використовуючи переваги Docker, організації можуть підвищити масштабованість, портативність і безпеку щодо розгортання DNS і VPN.

Контейнерізація за допомогою Docker пропонує багато переваг, коли йдеться про захист серверів DNS і VPN, пропонуючи контейнери, які ізолюють програми та їхні залежності одне від одного, допомагаючи пом'якшити конфлікти або вразливості, які можуть загрожувати хост-системам. Розміщуючи DNS-сервери та VPN-сервери в окремі контейнери, організації можуть гарантувати, що будь-який компроміс або порушення в одному компоненті не вплине негативно на інші служби або їх цілісність або доступність.

Docker полегшує розгортання та керування DNS-серверами та VPN-серверами завдяки своїй моделі розгортання на основі образів, яка швидко створює нові екземпляри в різних середовищах.

Контейнерізація також сприяє безпеці інфраструктур DNS і VPN, пропонуючи ще один рівень ізоляції від несанкціонованого доступу або маніпулювання критично важливими службами. Кожен контейнер Docker може мати власний мережевий стек, щоб мінімізувати поверхню атаки та одночасно зменшити потенційний вплив злому. Крім того, Docker пропонує такі вбудовані функції, як обмеження ресурсів і точне

керування доступом, які додають додаткову стратегію поглибленого захисту, яка зміцнює загальну безпеку системи.

Таким чином, контейнеризація Docker пропонує комплексний і ефективний засіб захисту мережевої інфраструктури за допомогою безпеки DNS-серверів із серверами VPN. Ізолюючи служби DNS і VPN в окремих контейнерах, організації можуть досягти ізоляції, масштабованості та портативності, одночасно обмежуючи ризики та захищаючи критично важливі служби від компрометації. Технологія контейнеризації Docker додатково зміцнює безпеку, забезпечуючи ізоляцію та вбудовані функції безпеки, що дозволяє організаціям ефективно реагувати та бути стійкими до постійного розвитку загроз. Прийнявши такий підхід, організації можуть посилити захист середовища, одночасно захищаючи програмні системи.

ВИСНОВКИ

У рамках кваліфікаційної роботи було досліджено інструменти та механізми захисту програмного забезпечення, особливо зосереджуючись на використанні серверів VPN із DNS для захисту цілісності, конфіденційності та доступності онлайн-сервісів.

Інтегруючи функції сервера DNS і VPN, організації можуть посилити свою онлайн-інфраструктуру проти різних загроз. Поєднання їх функціональних можливостей покращує конфіденційність шляхом інкапсуляції DNS-запитів і відповідей у зашифрований тунель для максимального захисту від перехоплення чи втручання. Додатково маршрутизація DNS-трафіку через сервери VPN захищає їх від мережевих атак, таких як DDoS-атаки, щоб гарантувати безперебійну доступність послуг.

Сервери VPN також забезпечують безпечне дистанційне керування та моніторинг інфраструктури DNS з будь-якого місця. Адміністратори можуть безпечно підключатися до свого DNS-сервера через VPN для виконання важливих завдань, таких як оновлення конфігурації, виправлення програмного забезпечення та моніторинг, таким чином не використовуючи менш безпечні методи, які відкривають інтерфейси керування DNS безпосередньо в Інтернеті.

У рамках дослідження було реалізовано модель мережевого покриття DNS серверу за допомогою VPN серверу використовуючи технології хостової віртуалізації. Досвід впровадження застосунку локально може бути використаний спеціалістами з кібербезпеки та інженерами програмного забезпечення, заради підвищення кібербезпеки та захисту від мережевих атак на підприємстві.

Щоб успішно реалізувати архітектуру, у якій DNS-сервер фільтрує запити тільки від VPN-сервера, необхідно врахувати певні міркування. Вони включають вибір надійного рішення VPN із надійним шифруванням, налаштування безпечних VPN-з’єднань і проектування відповідної архітектури мережі, а також контроль

доступу, моніторинг активності сервера та надання регулярних оновлень програмного забезпечення та послуг керування виправленнями.

Оскільки технологія стрімко розвивається, а кіберзагрози стають все складнішими, організації повинні визначити пріоритети захисту своїх програмних систем. Впровадження DNS-серверів і VPN-серверів забезпечує економічно ефективний підхід для захисту онлайн-інфраструктури, значно посилюючи стан кібербезпеки програмного забезпечення, одночасно захищаючи від вразливостей і гарантуючи безперебійну роботу сервісів.

Досліджені інструменти та механізми забезпечують інклюзивну стратегію кібербезпеки програмного забезпечення. Розуміючи та застосовуючи ці концепції в діяльності організацій, можна зменшити ризики успішної кібератаки, збільшити захист конфіденційності, забезпечити гарантію цілісності та захист цифрових активів у все більш цифровому та вразливому світі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Hacker's mind / Б. Шнайєр – Режим доступу: https://www.schneier.com/books/a-hackers-mind/
2. The Art of Invisibility / К. Мітник – Режим доступу: https://www.mitnicksecurity.com/the-art-of-invisibility-mitnick-security
3. Spam Nation / Б. Кребс – Режим доступу: https://krebsonsecurity.com/about/
4. 2011 PSN Network Outage [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/2011_PlayStation_Network_outage
5. Secure coding [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Secure_coding
6. Practical analysis on the algorithm of the Cross-Site Scripting Attacks / Blerton Abazi, Edmond Hajrizi – Sofia, 2022
7. SQL Injection Detection and Prevention using Aho-Corasick Pattern Matching Algorithm / Shreya Kini, Anushka Parvate Patil, Adithya Balasubramanyam – Belgaum, 2022
8. Static program analysis [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Static_program_analysis
9. Pentesting on web applications / Rina Elizabeth López de Jiménez –San Jose, 2016
10. Secure Hashing using BCrypt for Cryptographic Applications / C. Skanda, B. Srivatsa, B.S. Premananda – Vijapur, 2022
11. Argon2 [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Argon2
12. Salt (cryptography) [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Salt_(cryptography)
13. Analysis for security implementation in SDLC / Gaurav Raj, Dheerendra Singh, Abhay Bansal – Noida, 2014

14. Dynamic code analysis [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Dynamic_program_analysis
15. OWASP ZAP: 8 Key Features and How to Get Started [Електронний ресурс] – Режим доступу: https://brightsec.com/blog/owasp-zap/
16. What is Burp Suite? [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/what-is-burp-suite/
17. Introduction to Acunetix [Електронний ресурс] – Режим доступу: https://www.acunetix.com/support/docs/introduction/
18. Grammar-based Fuzzing Tool Using Markov Chain Model to Generate New Fuzzing Inputs / Hamad Al Salem, Jia Song – Las Vegas 2021
19. American fuzzy lop (fuzzer) [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/American_fuzzy_lop_(fuzzer)
20. Peach Fuzzing [Електронний ресурс] – Режим доступу: https://wiki.mozilla.org/Security/Fuzzing/Peach
21. Sulley: Fuzzing framework [Електронний ресурс] – Режим доступу: http://www.fuzzing.org/wp-content/SulleyManual.pdf
22. How RASP security works? [Електронний ресурс] – Режим доступу: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-runtime-application-self-protection-rasp/
23. Contrast security for developers [Електронний ресурс] – Режим доступу: https://www.contrastsecurity.com/developer
24. Sqreen: Application security management platform [Електронний ресурс] – Режим доступу: https://www.sqreen.com/
25. AppTrana Software Tool [Електронний ресурс] – Режим доступу: https://www.softwareadvice.com/risk-management/apptrana-profile/
26. Metasploit – Penetration tool software [Електронний ресурс] – Режим доступу: https://www.metasploit.com/
27. What is Nessus? [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchnetworking/definition/Nessus

28. OpenVAS – Open Vulnerability Assessment Scanner [Електронний ресурс] – Режим доступу: https://openvas.org/
29. Applying source code analysis techniques: A case study for a large mission-critical software system / Haralambi Haralambiev, Stanimir Boychev, Delyan Lilov, Kraicho Kraichev – Lisbon, 2011
30. What is SAST? [Електронний ресурс] – Режим доступу: https://www.synopsys.com/glossary/what-is-sast.html
31. SonarQube – code quality tool [Електронний ресурс] – Режим доступу: https://www.sonarsource.com/products/sonarqube/
32. Fortify static code analyzer [Електронний ресурс] – Режим доступу: https://www.microfocus.com/media/data-sheet/fortify_static_code_analyzer_static_application_security_testing_ds.pdf
33. Checkmarx – application security testing [Електронний ресурс] – Режим доступу: https://checkmarx.com/
34. The Adoption of JavaScript Linters in Practice: A Case Study on ESLint / Kristín Fjóla Tómasdóttir, Maurício Aniche, Arie Van Deursen - 2018
35. PMD – Network security tool [Електронний ресурс] – Режим доступу: https://www.oreilly.com/library/view/network-security-tools/0596007949/ch06s03.html
36. FindBugs [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/FindBugs
37. GNU Compiler Collection [Електронний ресурс] – Режим доступу: https://www.incredibuild.com/integrations/gcc
38. Clang: a C language family frontend for LLVM [Електронний ресурс] – Режим доступу: https://clang.llvm.org/
39. Apache Maven tool [Електронний ресурс] – Режим доступу: https://maven.apache.org/
40. OWASP Dependency Check [Електронний ресурс] – Режим доступу: https://owasp.org/www-project-dependency-check/

41. Snyk dependency checking tool [Електронний ресурс] – Режим доступу: https://docs.snyk.io/scan-application-code/snyk-open-source/starting-to-fix-vulnerabilities/differences-in-vulnerability-counts-across-environments
42. WhiteSource vs OWASP dependency check [Електронний ресурс] – Режим доступу: https://www.saashub.com/compare-whitesource-vs-owasp-dependency-check
43. What is HTTPS? [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/ssl/what-is-https/
44. What is SSH? [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchsecurity/definition/Secure-Shell
45. DNS and BIND, 5th Edition / Cricket Liu, Paul Albitz – 2006
46. IP-address [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/IP_address
47. A Client Based DNSSEC Validation Mechanism with Recursive DNS Server Separation / Yong Jin, Masahiko Tomoishi, Nariyoshi Yamai – Jeju 2018
48. Availability and effectiveness of root DNS servers: A long term study / Bu-Sung Lee, Yu Shyang Tan, Yuji Sekiya, Atsushi Narishige, Susumu Date – Osaka 2010
49. Top-level domain [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Top-level_domain
50. Authoritative Servers [Електронний ресурс] – Режим доступу: https://www.ibm.com/docs/en/zos/2.3.0?topic=servers-authoritative
51. DNS Security Extensions [Електронний ресурс] – Режим доступу: https://cloud.google.com/dns/docs/dnssec
52. Optimal Design of UDP Protocol in Embedded Real-Time OS / Haohang Li, Yufeng Zhao, Runzhi Wu – Yanji 2021
53. Understanding DoT and DoH (DNS over TLS vs. DNS over HTTPS) [Електронний ресурс] – Режим доступу: https://www.cloudns.net/blog/understanding-dot-and-doh-dns-over-tls-vs-dns-over-https/

54. What is DNS Spoofing? [Електронний ресурс] – Режим доступу: https://www.proofpoint.com/uk/threat-reference/dns-spoofing
55. HTTP Cookie [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/HTTP_cookie
56. Design and Implementation of UDP Tunneling-based on OpenSSH VPN / Irfaan Coonjah, Pierre Clarel Catherine, K. M. S. Soyjaudah – Greater Noida 2018
57. Performance Evaluation and Analysis of OpenVPN on Android / Junhua Qu, Tao Li, Fangfang Dang – Chongqing 2012
58. OpenSSL [Електронний ресурс] – Режим доступу: https://www.openssl.org/
59. From AES-128 to AES-192 and AES-256, How to Adapt Differential Fault Analysis Attacks on Key Expansion / Noemie Floissac, Yann L'Hyver – Nara 2011
60. Performance Comparison Between AES256-Blowfish and Blowfish-AES256 Combinations / Muhammad Abdul Muin, Arief Setyanto, Kartika Imam Santoso – Semarang 2018
61. What is TCP? [Електронний ресурс] – Режим доступу: https://www.fortinet.com/resources/cyberglossary/tcp-ip
62. What is TLS? [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
63. A light-PKI Fast Authentication Model / Derong Li, Shaobin Wang, Zi Li – Yichang 2011
64. Hardening a Work from Home Network with Wireguard and Suricata / Dimas Febriyan Priambodo, Amiruddin, Nanang Trianto – Padang 2021
65. Implementing Diffie-Hellman key exchange method on logical key hierarchy for secure broadcast transmission / Hüseyin Bodur, Resul Kara – Girne 2017
66. What is DDoS attack? [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/
67. What is IPSec ? [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/network-layer/what-is-ipsec/
68. Docker: Accelerated, Containerized Application Development [Електронний ресурс] – Режим доступу: https://www.docker.com/

69. Docker Compose overview [Електронний ресурс] – Режим доступу: https://docs.docker.com/compose/
70. What is YAML? [Електронний ресурс] – Режим доступу: https://www.redhat.com/en/topics/automation/what-is-yaml
71. Tunnelblick - free software for OpenVPN [Електронний ресурс] – Режим доступу: https://tunnelblick.net/
72. BIND Domain name service [Електронний ресурс] – Режим доступу: https://ubuntu.com/server/docs/service-domain-name-service-dns
73. Nslookup [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchnetworking/definition/nslookup
74. How NAT works [Електронний ресурс] – Режим доступу: https://www.comptia.org/content/guides/what-is-network-address-translation
75. Open Shortest Path First [Електронний ресурс] – Режим доступу: https://www.metaswitch.com/knowledge-center/reference/what-is-open-shortest-path-first-ospf
76. EasyRSA – Home [Електронний ресурс] – Режим доступу: https://easy-rsa.readthedocs.io/en/latest/
77. Stateless protocol [Електронний ресурс] – Режим доступу: https://byjus.com/gate/difference-between-stateless-and-stateful-protocol/