Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

**БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА**

на тему:

«Розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів»

Виконав: студент 4 курсу, гр.1КІТС-20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Слюсар Д.Ю.
(прізвище та ініціали)

Керівник: к.т.н., доц., каф. МБІС
Грицак А.В.
(прізвище та ініціали)
«07» червня 2024 р.

Рецензент: к.т.н., доц., каф. ОТ
Войцеховська О.В.
(прізвище та ініціали)
«07» червня 2024 р.

**Допущено до захисту**
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю. Є.
“07” червня 2024 р.

Вінниця ВНТУ – 2024

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

**ЗАТВЕРДЖУЮ**
**Голова секції УБ, кафедра МБІС**

**д.т.н., проф. Яремчук Ю.Є.**
“ 22 ” березня 2024 р.

**З А В Д А Н Н Я**
**на бакалаврську дипломну роботу студенту**
Слюсару Дмитру Юрійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи: «Розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів»
Керівник роботи к.т.н., доц. каф. МБІС Грицак Анатолій Васильович
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “11” березня 2024року № 80
2. Термін подання студентом роботи за тиждень до захисту.
3. Вихідні дані до роботи:
Метою роботи є здійснення розробки інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку.
4. Зміст текстової частини:
У першому розділі роботи досліджено теоретичні основи функціонування DNS та DNSSEC та актуальності проблеми виявлення та реагування на атаки DNS Cache Poisoning. У другому розділі роботи здійснено розробку архітектури інтелектуальної системи та бази даних для виявлення аномальних патернів в DNS трафіку. У третьому розділі роботи на основі розроблених алгоритмів здійснено розробку інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку. У висновках підбиваються усі результати роботи.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
У першому розділі бакалаврської дипломної роботи наявно 12 рисунків, у другому розділі – 16 рисунків, у третьому розділі – 16 рисунків.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Розділ І | Грицак А.В., к.т.н., доц. каф. МБІС | | |
| Розділ ІІ | Грицак А.В., к.т.н., доц. каф. МБІС | | |
| Розділ ІІІ | Грицак А.В., к.т.н., доц. каф. МБІС | | |
| | | | |
| | | | |
| | | | |
| | | | |

7.Дата видачі завдання 22 березня 2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва та зміст етапу | Термін виконання | | Примітка |
|---|---|---|---|---|
| | | початок | закінчення | |
| 1 | Визначення напрямку бакалаврської роботи, формулювання теми | 22.03.2024 | 26.03.2024 | Виконано |
| 2 | Аналіз предметної області обраної теми | 27.03.2024 | 15.04.2024 | Виконано |
| 3 | Розробка алгоритму роботи | 16.04.2024 | 03.05.2024 | Виконано |
| 4 | Написання бакалаврської роботи на основі розробленої теми | 04.05.2024 | 23.05.2024 | Виконано |
| 5 | Передзахист бакалаврської дипломної роботи | 24.05.2024 | 02.06.2024 | Виконано |
| 6 | Виправлення, уточнення, коригування бакалаврської дипломної робити | 03.06.2024 | 11.06.2024 | Виконано |
| 7 | Захист бакалаврської дипломної роботи | 13.06.2024 | 14.06.2024 | Виконано |

**Студент** _____ Слюсар Д. Ю.
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ Грицак А. В.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Бакалаврська дипломна робота складається із 107 сторінок формату А4, на яких є 44 рисунків, список використаних джерел містить 31 найменувань.

Метою роботи є розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів.

У першому розділі роботи досліджено теоретичні основи функціонування DNS та DNSSEC, актуальність проблеми виявлення та реагування на атаки DNS Cache Poisoning, проведено аналіз існуючих загроз DNS системи, можливих наслідків та огляд існуючих рішень у сфері виявлення аномальних патернів з використанням машинного навчання.

У другому розділі роботи здійснено розробку архітектури інтелектуальної системи та бази даних для виявлення аномальних патернів в DNS трафіку, що включає в себе застосування моделі машинного навчання для виявлення атак DNS, здійснено вибір та обґрунтування обраних алгоритмів та методів машинного навчання та мови програмування для реалізації системи.

У третьому розділі роботи здійснено розробку графічного інтерфейсу інтелектуальної системи, описано інструкцію користувача для ефективної роботи з програмною реалізацією, проведено тестування системи на реальних сценаріях атак та реакція на аномалії DNS трафіку.

**Ключові слова:** DNS, DNSSEC, атаки DNS Cache Poisoning, машинне навчання, захист мережі.

# ABSTRACT

The bachelor’s thesis consists of 107 pages of A4 format, on which there are 44 figures, the list of used sources contains 31 titles.

The purpose of the work is to develop an intelligent system for detecting and responding to "DNS Cache Poisoning" attacks using machine learning to detect abnormal patterns.

In the first part of the work, the theoretical basis of DNS and DNSSEC functioning, the relevance of the problem of detecting and responding to DNS Cache Poisoning attacks, the analysis of existing threats to the DNS system and possible consequences, and an overview of existing solutions in the field of detecting anomalous patterns using machine learning were investigated.

In the second part of the work, the development of the architecture of an intelligent system for detecting anomalous patterns in DNS traffic, which includes the application of a machine learning model to detect DNS attacks, the integration of the developed system with existing DNS security measures; the selection and justification of the selected algorithms and methods of machine learning and programming language for the implementation of the system was carried out.

In the third section of the work, the graphic interface of the intelligent system was developed, the user manual for effective work with the software implementation was described, the system was tested on real attack scenarios and the reaction to DNS traffic anomalies.



**Keywords:** DNS, DNSSEC, DNS Cache Poisoning attacks, machine learning, network protection.

# ЗМІСТ

# ВСТУП

**Актуальність.** В епоху цифрових технологій і розповсюдження Інтернету, безпека комп'ютерних мереж та інформаційних систем стає все більш критичною. Одним з важливих компонентів, що забезпечує функціонування мережі Інтернет, є система доменних імен (DNS - Domain Name System). DNS перетворює людино-зрозумілі доменні імена в IP-адреси, необхідні для маршрутизації трафіку. Однак, DNS вразлива до атак «отруєння кешу», при яких зловмисники намагаються підмінити легітимні записи DNS невірними даними. Такі атаки можуть призводити до перенаправлення користувачів на шкідливі сайти, крадіжки конфіденційних даних, недоступність важливих мережевих служб, фінансові втрати та інших серйозних наслідків.

Постійність та еволюція атак DNS Cache Poisoning підкреслюють нагальну потребу у вдосконалених стратегіях виявлення та усунення наслідків. Традиційні заходи безпеки, хоча і є необхідними, виявилися недостатніми проти деяких складних атак, що призвело до нагальної потреби в інноваційних рішеннях. Метою цього дослідження є не тільки усунення безпосередніх загроз, пов'язаних з отруєнням кешу DNS, але й внесок у більш широку сферу кібербезпеки шляхом підвищення стійкості інфраструктури DNS за допомогою інтелектуальних, адаптивних засобів захисту.

Саме тому розробка ефективних методів виявлення та протидії атакам "отруєння кешу DNS" є актуальним завданням забезпечення кібербезпеки.

**Метою роботи є** розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку. Для досягнення цієї мети проаналізовано сучасні методи захисту, розроблено програмний модуль для виявлення і блокування атакуючих IP-адрес, а також конфігуровано автоматичні сповіщення адміністратору.

**Задачами дослідження є:**

– аналіз проблеми атак "отруєння кешу DNS" та існуючих методів їх виявлення і протидії;

– вивчення підходів машинного навчання для виявлення аномалій в мережевому трафіку;

– розробка навчальної моделі для класифікації DNS-запитів та ідентифікації атак "отруєння кешу";

– створення системи моніторингу та аналізу DNS-трафіку з використанням навченої моделі;

– тестування та оцінка ефективності розробленої інтелектуальної системи.

**Об’єкт дослідження:** система для виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку.

**Предмет дослідження:** процес розробки системи, яка забезпечує захист DNS серверів від атак "отруєння кешу DNS".

**Новизна роботи** полягає в розробці інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів»

**Практична цінність:** розроблено програмний продукт, який являє собою систему виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку.

**Апробація результатів:** Основні наукові результати роботи доповідалися та обговорювалися на 1 конференції:

1. Міжнародна науково-практична інтернет-конференція на тему «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)».

# 1 ЗАГАЛЬНИЙ АНАЛІЗ ПРОБЛЕМИ DNS CACHE POISONING

В даному розділі здійснимо аналізу обраної галузі, а саме визначимо актуальність дослідження, оглянемо існуючі рішення у даній сфері. Також будуть розглянуті існуючі загрози DNS системи та проведемо аналіз методів та інструментів забезпечення комп'ютерних мереж і захисту від атак DNS Cache Poisoning.

## 1.1 Актуальність проблеми виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів

Атаки "отруєння кешу DNS" (DNS cache poisoning) є однією зі значних загроз безпеці інтернет-протоколів, особливо в контексті сучасного цифрового світу, де доступ до мережі є необхідністю. Ця форма атаки може призвести до серйозних наслідків, включаючи перенаправлення трафіку на веб-сайти зловмисників, крадіжку конфіденційних даних та порушення конфіденційності користувачів [1].

Система доменних імен (DNS) має фундаментальне значення для роботи Інтернету, оскільки вона перетворює доменні імена, що читаються людиною, на машинні IP-адреси. Отруєння кешу DNS, також відоме як підміна DNS, - це процес введення неправдивої інформації в кеш DNS-розпізнавача, щоб він повертав неправильні IP-адреси у відповідь на запити користувачів. Це може перенаправляти користувачів на шкідливі веб-сайти та сприяти фішинговим атакам, розповсюдженню шкідливого програмного забезпечення і витоку даних. Складність цих атак зумовлена вразливістю протоколу DNS та використанням моделі довіри, яка працює в DNS, що робить їх особливо складними для виявлення та усунення (рис. 1.1).

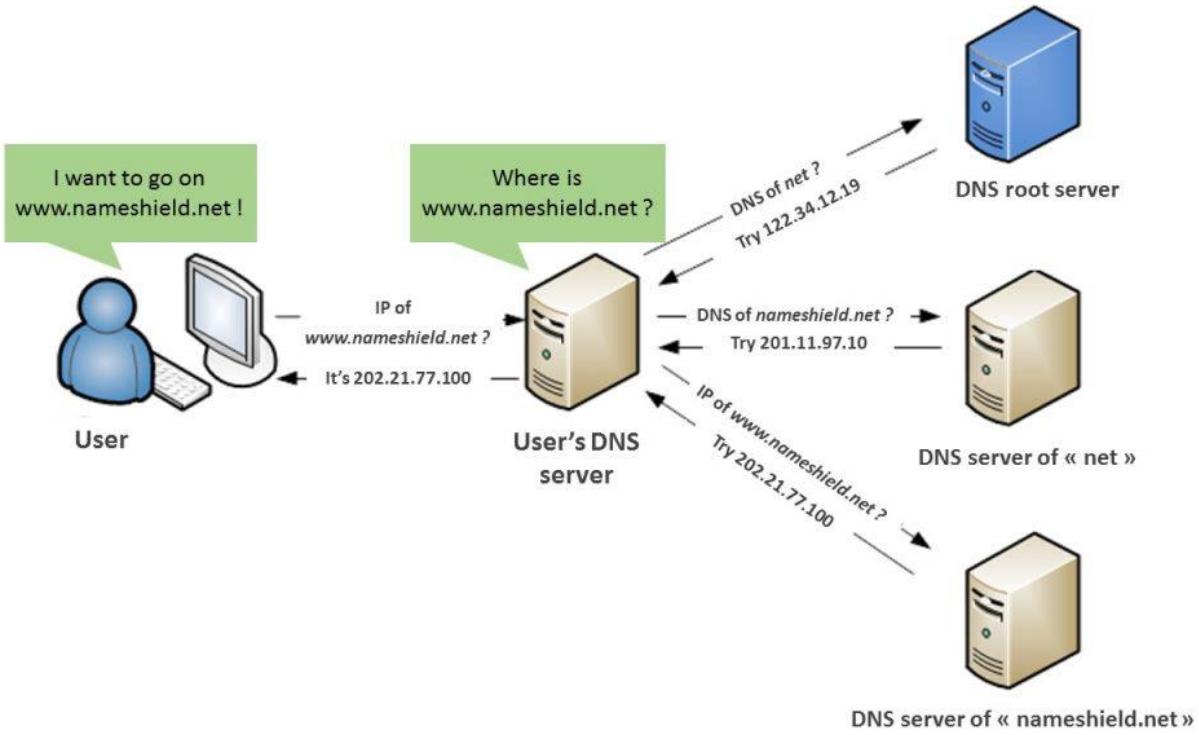



Рисунок 1.1 – Схема роботи DNS [2]

DNS дозволяє веб-користувачу повідомити доменне ім'я у своєму веб-браузері для доступу до веб-сайту. Потім браузер «розв'язує» це доменне ім'я, щоб отримати IP-адресу веб-сервера, який розміщує цей веб-сайт і відображає його. Даний процес називається «дозвіл DNS».

З основною функцією DNS (Domain Name System) пов'язані ключові аспекти функціонування інтернету, такі як перетворення доменних імен в IP-адреси. Однак, через цей процес також можуть виникати вразливості, які використовуються зловмисниками для отруєння кешу DNS. Це стає особливо актуальним у сучасному світі, де безпека мережевих систем і захист від кіберзагроз стають все більшою проблемою.

Нагальність боротьби з отруєнням кешу DNS зумовлена потенційною загрозою для цілісності та надійності Інтернету. Враховуючи поширення Інтернету та зростання кількості кіберзагроз, розробка ефективних методів виявлення та реагування на атаки "отруєння кешу DNS" стає необхідністю для забезпечення безпеки користувачів та надійності мережевих інфраструктур. Успішні атаки можуть призвести до поширення дезінформації, втрати довіри користувачів і

значної фінансової та репутаційної шкоди для постраждалих організацій. Крім того, розвиток технологій TCP (Transmission Control Protocol), включаючи використання сучасного шкідливого програмного забезпечення та нових мережевих протоколів, випереджає традиційні заходи безпеки. Така ескалація вимагає інноваційних рішень, здатних швидко виявляти та усувати такі загрози [3].

Зловмисники, які використовують атаки "отруєння кешу DNS", використовують різні методи, щоб викликати погану поведінку кеш-системи DNS, щоб тим самим вплинути на процес перетворення доменних імен у відповідні IP-адреси. Ці атаки може бути важко виявити та захиститися від них через їхню хитрість та різноманітність.

На сьогоднішній день у світі Інтернету актуальність проблеми виявлення та реагування на атаки "отруєння кешу DNS" підкреслюється швидким розвитком технологій та зростаючою складністю мережевих середовищ. Сучасні атаки можуть бути здійснені з використанням різноманітних інструментів та методів, що вимагає від фахівців з кібербезпеки постійного вдосконалення методів виявлення та захисту.

У зв'язку з цим, розробка та впровадження ефективних та надійних захисних механізмів стають ключовим завданням для організацій, які прагнуть забезпечити безпеку своїх мереж та інфраструктури. Правильна реакція на потенційні загрози "отруєння кешу DNS" вимагає співпраці між фахівцями з кібербезпеки, мережевими адміністраторами та іншими зацікавленими сторонами, а також вдосконалення систем виявлення та моніторингу.

Так, використання машинного навчання для виявлення аномальних патернів може бути дуже корисним в контексті боротьби з атаками "отруєння кешу DNS". Машинне навчання може допомогти виявляти незвичайні та аномальні активності в мережі, які можуть свідчити про спроби атаки або зловмисну поведінку.

Машинне навчання (ML) є перспективним напрямком для покращення безпеки DNS, оскільки воно може виявляти аномальні патерни, що вказують на DoS-атаки. На відміну від традиційних механізмів безпеки, які покладаються на заздалегідь визначені підписи та шаблони, алгоритми ML можуть навчатися на даних і виявляти потенційні загрози на основі відхилень від встановлених норм. Ця функція

особливо актуальна для отруєння кешу DNS, оскільки вектори атак постійно змінюються. За допомогою ML можна розробляти системи, які адаптуються до нових загроз і підвищують стійкість інфраструктури DNS до сучасних атак.

Розробка та впровадження систем виявлення на основі ML вимагає суворої методології. Це включає збір і попередню обробку даних DNS-трафіку, вибір і навчання відповідних моделей ML, а також постійне оцінювання та вдосконалення моделей для підтримки високої точності виявлення. Ефективність таких систем залежить від їхньої здатності збалансувати чутливість (здатність виявляти атаки) і специфічність (здатність уникати помилкових спрацьовувань), щоб гарантувати, що легітимний DNS-трафік не буде перехоплений неналежним чином.

Наприклад, моделі машинного навчання можуть бути навчені аналізувати типові шаблони запитів DNS та виявляти відхилення від цих шаблонів, що може свідчити про можливу спробу атаки. Також можна навчити моделі розпізнавати характеристики зловмисницької поведінки, які можуть вказувати на атаку "отруєння кешу DNS".

Однак, важливо пам'ятати, що успішне використання машинного навчання для виявлення аномальних патернів потребує якісних даних для навчання моделей, а також постійного моніторингу та адаптації моделей до нових загроз і змін у мережевому середовищі. Також потрібно урахувати можливість фальсифікації зловмисниками, які можуть намагатися обманути систему виявлення аномалій.

Емпіричні дані демонструють потенціал методів віртуалізації для підвищення безпеки DNS. Дослідження показали, що різні алгоритми ML, включаючи дерева рішень, машини опорних векторів і нейронні мережі, є високоефективними у виявленні аномалій DNS з високою точністю. Однак практична реалізація цих рішень пов'язана з певними труднощами, зокрема з необхідністю великих обсягів навчальних даних, ризиком переналаштування моделі та обчислювальними ресурсами, необхідними для аналізу в реальному часі.

Використання машинного навчання може бути корисним інструментом в боротьбі з атаками "отруєння кешу DNS", але це лише один із компонентів комплексної стратегії кібербезпеки.

Впровадження машинного навчання у алгоритм аналізу трафіку є корисним кроком для поліпшення ефективності та надійності проекту. Кілька способів, які можуть бути використані для інтеграції машинного навчання у алгоритм пошуку:

– автоматичне виявлення аномальних запитів DNS (використання моделей машинного навчання для автоматичного виявлення аномальних запитів DNS може допомогти швидко реагувати на потенційно шкідливу активність та захищати вашу мережу від атак "отруєння кешу DNS");

– покращення пошукових результатів (машинне навчання може бути використане для аналізу та врахування користувацьких патернів пошуку, що дозволить покращити релевантність та точність результатів пошуку);

– автоматизація підтримки інфраструктури (можна розглянути використання машинного навчання для автоматичного моніторингу та оптимізації інфраструктури проекту, що дозволить зменшити навантаження на систему та покращити її продуктивність).

Інтеграція машинного навчання у алгоритм пошуку може вимагати додаткового дослідження та розробки, але це може принести значні переваги в плані ефективності та користувацького досвіду.

Отже, неможливо переоцінити нагальність вирішення критичної проблеми безпеки, пов'язаної з отруєнням кешу DNS. Такі атаки є складними і можуть порушити фундаментальні аспекти роботи Інтернету, що вимагає передових методів виявлення. Машинне навчання дає змогу розширити можливості виявлення і забезпечує життєздатний шлях вперед, який обіцяє розвиватися разом із загрозами, які він має на меті пом'якшити. Успішне розгортання системи захисту DNS на основі ML вимагає узгоджених зусиль, що включають глибокі дослідження, емпіричну перевірку та постійне вдосконалення. Пошук такого рішення є не лише технічним викликом, але й фундаментальним імперативом для захисту цілісності та надійності Інтернету.

## 1.2 Аналіз існуючих загроз DNS-системи та можливих наслідків DNS Cache Poisoning

Система доменних імен (DNS) є фундаментальною частиною інфраструктури Інтернету, яка перетворює людські доменні імена в IP-адреси. Однак ця важлива функція затьмарюється зростаючою кількістю витончених загроз, які намагаються використати її вразливості. Серед них отруєння кешу DNS та інші вектори атак становлять значний ризик для цілісності, доступності та конфіденційності онлайн-сервісів.

Отруєння кешу DNS є ключовим вектором загроз, коли зловмисники маніпулюють DNS-резолверами (сервер, який перетворює доменні імена на IP-адреси) для кешування некоректних відображень адрес і перенаправлення користувачів на шкідливі веб-сайти. Часто користувачів перенаправляють на копії веб-сайтів відомих організацій, які використовують у зловмисних цілях, наприклад, для поширення шкідливого програмного забезпечення або входу в особисті акаунти для збору конфіденційної інформації та видачі себе за іншу особу [4]. Такі атаки ставлять під загрозу не лише безпеку користувачів, але й надійність онлайн-сервісів. Витонченість таких атак зростає, і вони продовжують уникати виявлення та продовжувати свою шкідливу діяльність, використовуючи такі методи, як швидке потокове передавання DNS та алгоритми генерації доменів (DGA). Зловмисники розробляють DGA, щоб дозволити шкідливому програмному забезпеченню швидко генерувати список доменів, які можуть бути використані для отримання інструкцій та інформації. Зловмисники часто використовують DGA, щоб швидко змінювати домени, які вони використовують для атак. Програмне забезпечення та постачальники намагаються блокувати та знищувати шкідливі домени якомога швидше [5].

На додаток до отруєння кешу, зловмисники використовують різноманітні механізми для використання вразливостей DNS. До них відносяться посилення DNS (використання відкритих вирішувачів (Open Solver) для перетворення DNS-серверів на зброю для розподілених атак на відмову в обслуговуванні (DDoS)),

перехоплення DNS (коли зловмисники перенаправляють DNS-запити на зловмисний сервер), тунелювання DNS (коли DNS-запити та відповіді використовуються для обходу мережі), а також DNS-запити та відповіді використовуються для обходу заходів мережевої безпеки, витоку даних та управління мережею) (рис. 1.2) [6].

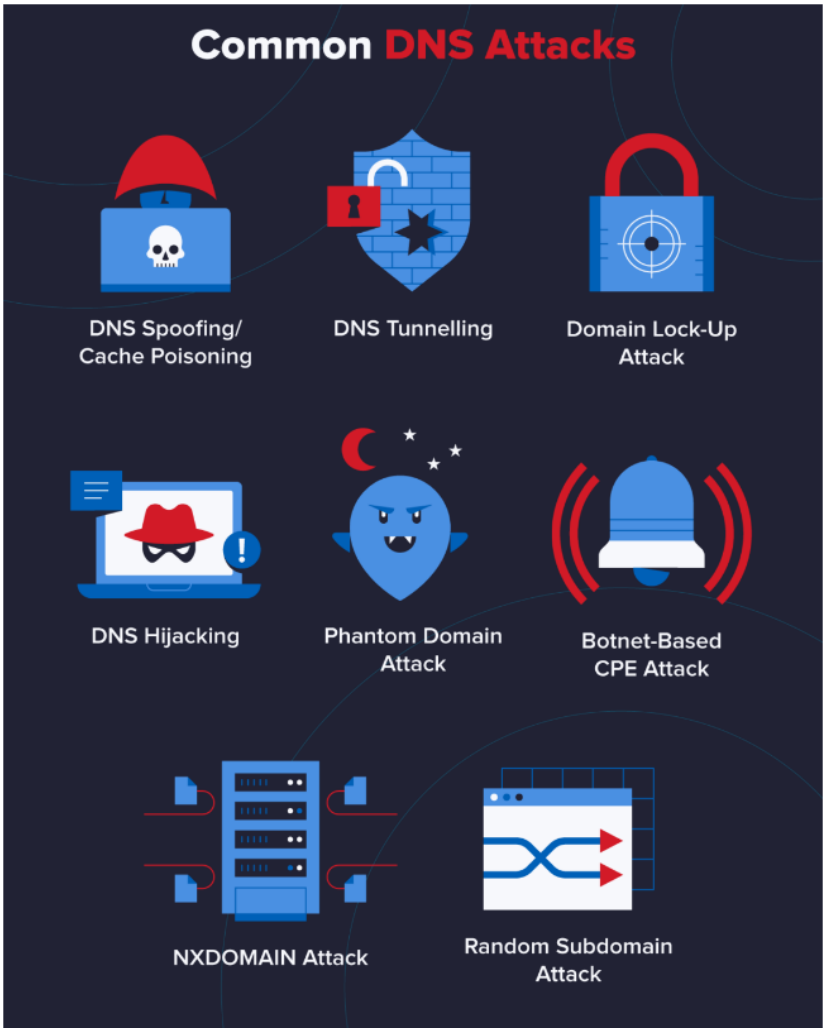



Рисунок 1.2 – Поширені DNS атаки [7]

Отруєння кешу - це тип атаки, під час якої підроблені DNS-дані вводяться в кеш DNS-розпізнавача, в результаті чого він повертає фальшиву IP-адресу для домену.

DNS-тунелювання в основному використовується для обходу контролю безпеки мережі для віддалених з’єднань. Цей тип атаки використовує інші протоколи для тунелювання DNS-запитів і відповідей. Використовуючи SSH, TCP або HTTP, зловмисники можуть впроваджувати шкідливе програмне забезпечення або

викрадену інформацію в DNS-запити, які більшість брандмауерів не завжди можуть виявити. Зловмисники використовують DNS-розпізнавачі для перенаправлення запитів на сервер зловмисника, на якому встановлено тунельне програмне забезпечення; після встановлення з'єднання між жертвою і зловмисником через DNS-розпізнавач тунель може бути використаний для витоку даних або для інших зловмисних цілей, які можуть бути реалізовані.

Викрадення DNS – це акт переспрямування вашого звичайного DNS-трафіку шляхом зміни реєстратора домену на інший цільовий DNS-сервер.

Атака NXDOMAIN - процес DDoS-атаки авторитетного DNS-сервера нелегітимними запитами домену для примусової відповіді. Це тип DNS-атаки, в якій зловмисник намагається спричинити відмову в обслуговуванні легітимного трафіку, запитуючи записи для доменних імен, які не існують, надсилаючи велику кількість запитів на DNS-сервер. Атаки NXDOMAIN також можуть бути спрямовані на рекурсивні резолвери, щоб переповнити кеш резолвера небажаними запитами.

Атака на фантомний домен змушує розпізнавач DNS чекати відповіді від не існуючих доменів, що, у свою чергу призводить до низької продуктивності. Ця атака має схожі результати з NXDOMAIN-атакою на DNS-резолвери. Зловмисник створює велику кількість "примарних" доменних серверів, які дуже повільно або взагалі не відповідають на запити. Після цього, вирішувач змушений отримувати потік запитів до цих доменів і чекати на відповідь, що призводить до низької продуктивності і відмови в обслуговуванні.

Випадкова атака субдоменів - це скомпрометовані хости та бот-мережі , які по суті, дозують законний домен, але зосереджують свій вогонь на фальшивих субдоменах, щоб змусити пошук записів DNS і перевизначити службу. У цьому випадку зловмисник надсилає DNS-запити на кілька неіснуючих випадкових піддоменів одного і того ж авторитетного сайту. Мета полягає в тому, щоб викликати відмову в обслуговуванні на сервері доменних імен авторитетного сайту і унеможливити знаходження сайту сервером імен. Як побічний ефект, може

постраждати інтернет-провайдер, який обслуговує зловмисника, а кеш рекурсивного вирішувача буде завантажений неправильно оформленими запитами.

Атака на блокування домену – це надсилання численних небажаних відповідей, щоб утримувати розв’язувані задіяними та заблокованими щодо ресурсів. Зловмисники організовують цю форму атаки, налаштовуючи спеціальний домен і вирішувач для створення TCP-з'єднань з іншими легітимними вирішувачами. Коли цільовий вирішувач надсилає запит, ці домени надсилають назад повільний потік випадкових пакетів, перевантажуючи ресурси вирішувача.

Атака CPE на основі ботнету - це сукупність комп’ютерів, модемів, маршрутизаторів тощо, які використовуються для зосередження обчислювальної потужності на одному конкретному веб-сайті чи ресурсі, щоб перевантажити його запитами трафіку.

Протягом багатьох років зловмисники успішно здійснювали різноманітні атаки на основі DNS проти корпоративних мереж та їхніх користувачів. Зловмисники часто використовують DNS для забезпечення командування і контролю над мережею ботнетів. Це може призвести до несанкціонованого доступу до мережі, підслуховування та вилучення даних.

Згідно з інформацією, що надала компанія EfficientIP (компанія, яка займається автоматизацією та безпекою та спеціалізується на безпеці DNS), близько 79% постачальників послуг у 2022 році стикалися з атаками, що використовують DNS. У звіті під назвою "IDC 2022 Global DNS Threat Report" зазначено, що, незважаючи на стійкий ріст усвідомлення проблем безпеки DNS з року в рік, кількість атак продовжує збільшуватися, що призводить до збільшення фінансових втрат компаній від цих інцидентів (рис. 1.3). [8]

Найпоширенішими видами атак на DNS є фішинг (37%), шкідливе програмне забезпечення, спрямоване на DNS (34%), DDoS-атаки (27%), атаки на блокування доменів, що виснажують ресурси серверів системи доменних імен (22%), а також атаки з підсиленням, які можуть спричинити відмову в роботі мережі компанії та значні економічні втрати (21%).

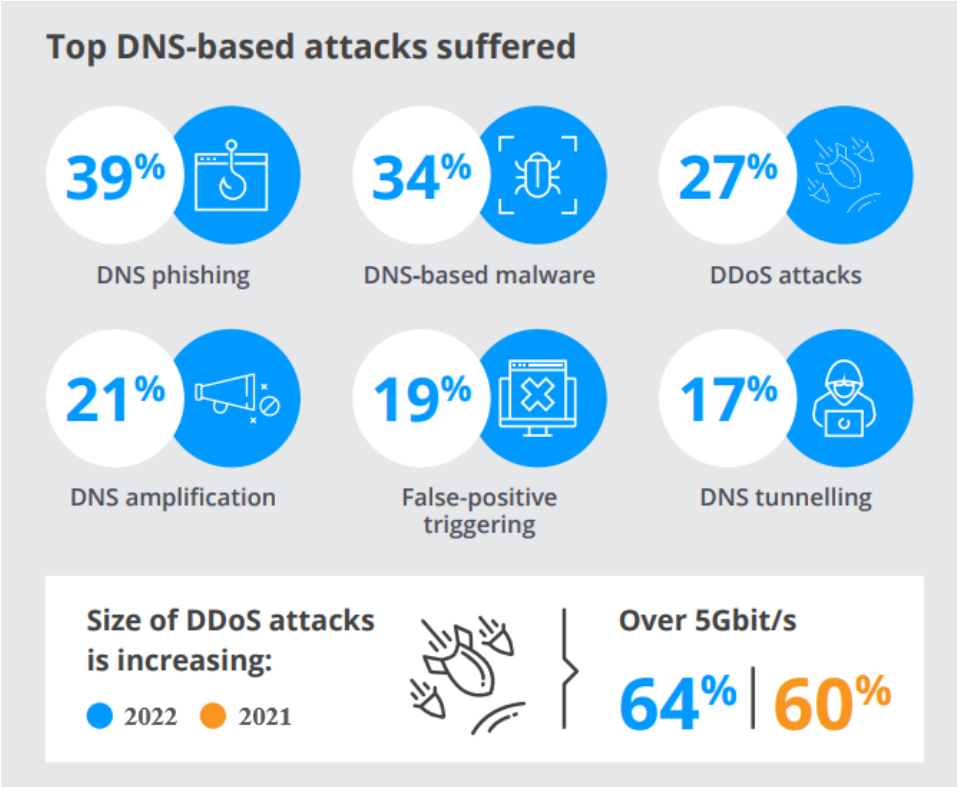



Рисунок 1.3 – Рейтинг популярності типів атак, що використовують DNS [8]

Розглянемо 2 реальних приклади атак з використанням DNS Cache Poisoning, а саме підробка відповіді за допомогою Birthday Attack та експлойт Камінського.

Підробка відповіді за допомогою Birthday Attack - DNS не перевіряє відповіді на рекурсивні запити, тому перша відповідь зберігається в кеші. Зловмисники використовують «парадокс дня народження», щоб спробувати передбачити та надіслати підроблену відповідь автору запиту. Ця атака на день народження використовує математику та теорію ймовірності, щоб зробити припущення. У цьому випадку зловмисник намагається вгадати ідентифікатор транзакції вашого запиту DNS, тому підроблена відповідь із підробленим записом DNS потрапляє до вас раніше, ніж справжня відповідь. Атака на день народження не є гарантованим успіхом, але зрештою зловмисник заховає підроблену відповідь у кеш. Після успішної атаки зловмисник бачитиме трафік із підробленого запису DNS, доки не закінчиться час життя (TTL) [19].

Експлойт Камінського є різновидом атаки на день народження, представленої на BlackHat 2008. Спочатку зловмисник надсилає цільовому резолверу DNS-запит

для неіснуючого домену, наприклад «fake.varonis.com». Потім резолвер пересилає запит до авторитетного сервера імен, щоб отримати IP-адресу для фальшивого субдомену. У цей момент зловмисник заповнює резолвер величезною кількістю підроблених відповідей, сподіваючись, що одна з цих підробок збігається з ідентифікатором транзакції оригінального запиту. Якщо вони успішні, зловмисник отруїв DNS-кеш цільового резолвера підробленою IP-адресою для – у цьому прикладі – varonis.com. Резолвер продовжуватиме повідомляти будь-кому, хто його запитає, що IP-адреса для varonis.com є підробленим запитом до TTL.

Щоб виявити атаку DNS Cache Poisoning - відстежуйте свої DNS-сервери на наявність індикаторів можливих атак. Люди не мають обчислювальної потужності, щоб встигати за кількістю DNS-запитів, які вам потрібно відстежувати. Застосуйте аналітику безпеки даних до моніторингу DNS, щоб відрізнити нормальну поведінку DNS від атак. Наприклад, раптове збільшення активності DNS з одного джерела щодо одного домену вказує на потенційну атаку Birthday, а збільшення активності DNS з одного джерела, яке запитує ваш DNS-сервер щодо кількох доменних імен без рекурсії, вказує на спробу знайти запис для використання для отруєння.

Окрім моніторингу DNS , слідкуйте за подіями Active Directory і поведінкою файлової системи на наявність ненормальної активності. А ще краще, використовуйте аналітику, щоб співвіднести дії між усіма трьома векторами, щоб додати цінний контекст до вашої стратегії кібербезпеки.

З огляду на критично важливу роль DNS в роботі Інтернету і потенційну можливість широкомасштабних збоїв і пошкоджень, нагальність усунення цих загроз є особливо високою. Традиційні механізми захисту, такі як статичні брандмауери і системи виявлення на основі сигнатур, виявилися недостатніми для боротьби з динамічним і адаптивним характером цих загроз. Ця неадекватність зумовила необхідність дослідження та впровадження передових рішень безпеки, таких як машинне навчання для виявлення аномалій, DNSSEC для перевірки відповідей DNS, а також проактивний моніторинг та управління конфігурацією для зменшення ризиків.

Отже, незважаючи на важливість системи DNS, вона також є вразливою до активної експлуатації різноманітними зловмисниками. З огляду на складність і серйозність цих загроз, необхідно докласти спільних зусиль для зміцнення безпеки DNS з використанням передових технологій і передового досвіду. Розуміючи, як працюють DNS-атаки, характеристики потенційних зловмисників і вразливості, які вони використовують, зацікавлені сторони можуть краще захистити себе від цих поширених загроз. Використання машинного навчання для виявлення аномалій у поєднанні з DNSSEC та проактивним управлінням безпекою є ефективним способом захисту інфраструктури DNS, а отже, і ширшої цифрової екосистеми.

### 1.3 Аналіз методів та інструментів захисту від атак DNS Cache Poisoning

Система доменних імен (DNS) є наріжним каменем функціональності Інтернету і вразлива до багатьох форм атак, але отруєння кешу DNS є однією з найбільш руйнівних атак. Цей метод атаки порушує цілісність відповідей DNS і спричиняє низку проблем з безпекою. Протягом багатьох років було розроблено багато методів та інструментів для захисту від таких вразливостей. Оглянемо основні методи:

– використання DNSSEC (DNS Security Extensions) - це набір розширень для DNS, які надають аутентифікацію та цілісність даних DNS. Це дозволяє перевіряти автентичність отриманих DNS-відповідей і попереджати отруєння кешу DNS;

– використання фаєрволів і фільтрація трафіку - встановлення правил фаєрвола для обмеження доступу до DNS-серверів та фільтрація неперевірених запитів може допомогти у запобіганні атак "отруєння кешу DNS";

– моніторинг DNS-трафіку - системи моніторингу DNS-трафіку можуть виявляти аномальність та вчасно сповіщати про можливі атаки, що допомагає оперативно реагувати на них;

– використання Intrusion Detection/Prevention Systems (IDS/IPS) IDS/IPS можуть виявляти та блокувати спроби атак "отруєння кешу DNS" на рівні мережі;

– обмеження використання кешу DNS - встановлення обмежень на кешування DNS-відповідей може зменшити ризик отруєння кешу DNS.

Зазвичай ці методи використовують в комбінації для максимального захисту від атак "отруєння кешу DNS". Вибір конкретних методів повинен враховувати конкретні потреби та характеристики мережі.

Більшість користувачів захищають свій веб-трафік від перехоплення і переміщення за допомогою сучасних технологій безпеки, таких як DNSCrypt, тобто https, але, на жаль, багато хто забуває про інший незахищений аспект - DNS-запити. За невеликими винятками, архітектура DNS не змінювалася з 1983 року. Кожного разу, коли браузер намагається відкрити веб-сайт, він надсилає запит на DNS-сервер з доменним ім'ям, а DNS-сервер відповідає необхідною IP-адресою (рис. 1.4). Ні запит, ні відповідь не шифруються і надсилаються у відкритому вигляді. Це означає, що провайдери, мережеві адміністратори або зловмисники, які використовують MITM-атаки (Man in the middle, тобто "людина посередині") , можуть не тільки зберігати історію всіх ваших веб-сайтів, але й підробляти відповіді на ці запити.

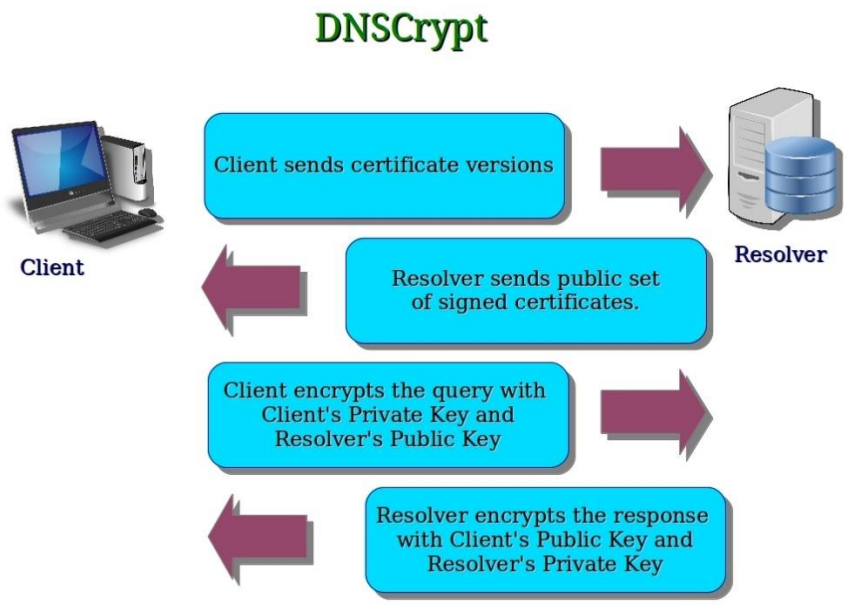



Рисунок 1.4 – Принцип роботи DNSCrypt [10]

Наступним прикладом є DNSSEC - розшифровується як Domain Name System Security Extensions і використовується для перевірки DNS-записів без необхідності знати детальну інформацію навколо кожного конкретного DNS-запиту. DNSSEC

використовує пари ключів цифрового підпису (PKI (Public Key Infrastructure)), щоб перевірити, чи надходить відповідь на запит DNS із належного джерела. Впровадження DNSSEC - це не лише найкраща галузева практика, яку вимагає NIST, але й ефективний спосіб боротьби з простотою роботи DNS і уникнення більшості DNS-атак (рис. 1.5) [11].

Розширення безпеки системи доменних імен (DNSSEC) є фундаментальним підходом до пом'якшення наслідків отруєння кешу DNS. Дозволяючи підписувати відповіді DNS цифровим підписом, DNSSEC гарантує автентичність відповіді і не дозволяє зловмисникам вводити шкідливі дані в кеш DNS. DNSSEC значно підвищує безпеку, але не може бути повністю ефективною через свою складність і необхідність широкого впровадження в ланцюжку DNS [12].

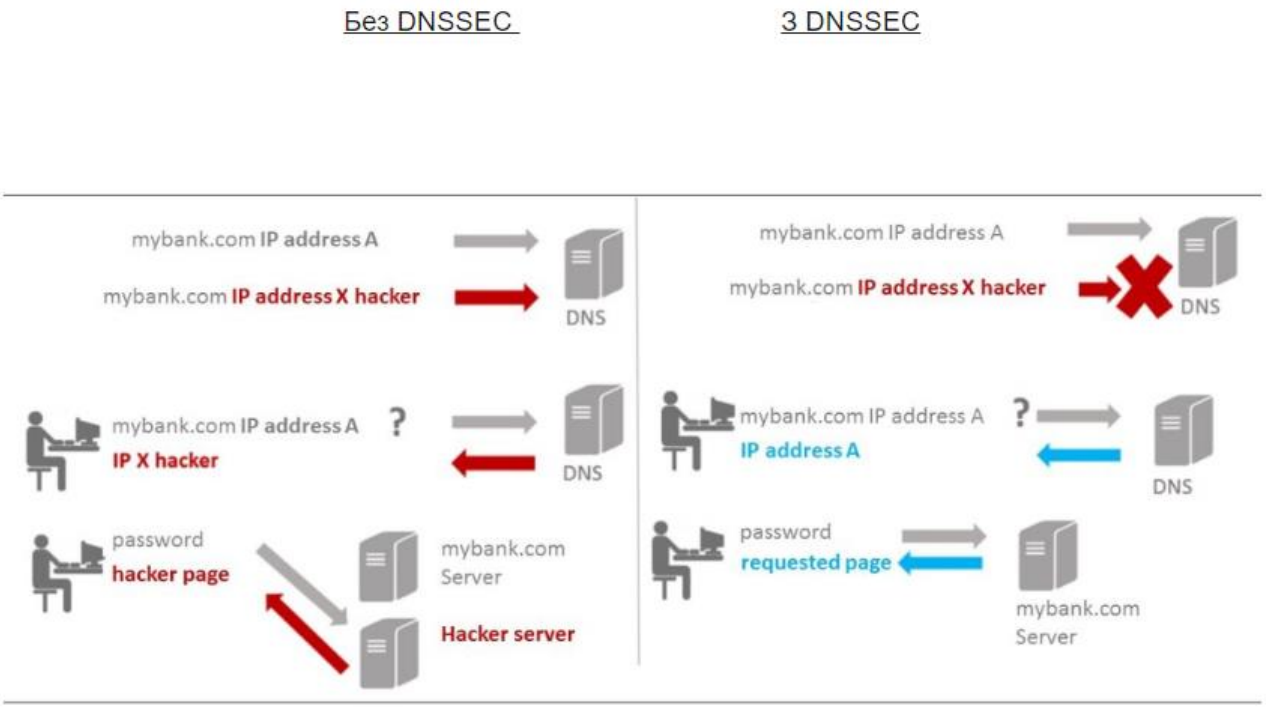



Рисунок 1.5 – Схема роботи без та з DNSSEC [2]

Процес DNSSEC працює наступним чином, щоб забезпечити цілісність вирішення DNS, DNSSEC розробляє ланцюг довіри, який повертається до кореня DNS (рис. 1.5)). Безпека даних забезпечується механізмом ключів (KSK (Key Signing Key) для ключа підпису ключа та ZSK (Zone Signing Key) для ключа підпису

зони), який підписує записи DNS у власній зоні. Відкриті ключі надсилаються до відповідного реєстру для архівації; реєстр, зв'язаний DNSSEC з кореневим сервером, розвивається ланцюг довіри. Кожна батьківська зона DNS забезпечує автентичність ключів своїх дочірніх зон, підписуючи їх. DNSSEC працює подібно до TLS/HTTPS, використовуючи пари відкритих/приватних ключів для цифрового підпису записів DNS [13].

Також існують криптографічні рішення, відмінні від DNSSEC, такі як TLS для DNS (DoT) і HTTPS для DNS (DoH), шифрують DNS-запити і відповіді, щоб запобігти підслуховуванню і фальсифікації даних DNS під час передачі. Хоча ці протоколи покращують конфіденційність і безпеку, вони також перекладають відповідальність на DNS-розв'язувачі і вимагають довіри до цих централізованих суб'єктів.

TLS найчастіше використовується як частина HTTPS («SSL») у вашому веб-переглядачі, оскільки надсилаються запити для захисту серверів HTTP.

DNS Over TLS (DoT) використовує TLS для шифрування UDP-трафіку звичайних запитів DNS (рис. 1.6). Шифрування цих простих текстових запитів допомагає захистити користувачів або програми, які надсилають запити, від кількох різних атак. До прикладу, Man-In-The-Middle: без шифрування система-посередник, що знаходиться між клієнтом і авторитетним DNS-сервером, потенційно може відповісти клієнту неправдивою або шкідливою інформацією; шпигунство та відстеження: за відсутності шифрування запитів системам-посередникам легко побачити, до яких сайтів звертається певний користувач або програма. Хоча конкретну сторінку на сайті не буде видно лише з DNS, просто знати, які домени запитуються, достатньо, щоб сформувати профіль системи чи особи [14].

DNS через HTTPS (або DoH) - це експериментальний протокол, який очолюють Mozilla та Google і має дуже схожі цілі з DoT: покращення конфіденційності людей, які переглядають Інтернет, шляхом шифрування DNS-запитів і відповідей.

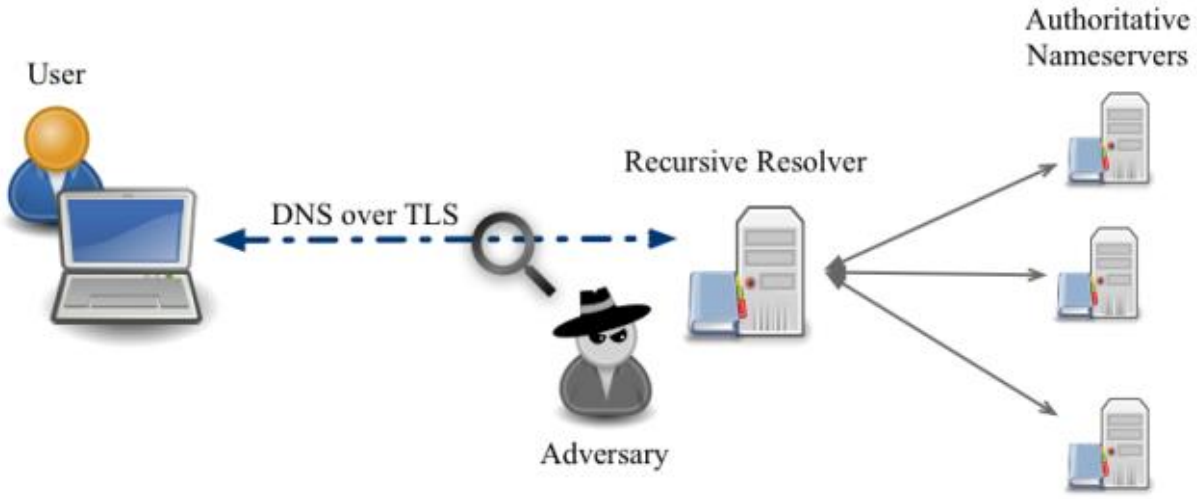



Рисунок 1.6 – Схема роботи DoT [15]

Стандартні DNS-запити передаються через UDP, а запити та відповіді можна переглядати за допомогою таких інструментів, як Wireshark . DoT шифрує ці запити, але їх усе ще можна ідентифікувати як досить чіткий трафік UDP у мережі.

DoH використовує інший підхід і передає зашифровані запити на розв'язання DNS через з'єднання HTTPS, які через з'єднання виглядають як будь-які інші веб-запити. Ця різниця має значні наслідки як для системних адміністраторів, так і для майбутнього розпізнавання імен (рис. 1.7).

Фільтрація DNS – це поширений спосіб фільтрації веб-трафіку для захисту користувачів від фішингових атак, сайтів розповсюдження зловмисного програмного забезпечення чи іншої потенційно шкідливої діяльності в Інтернеті в корпоративній мережі. DoH обходить ці фільтри, потенційно наражаючи користувачів і мережі на більш високий рівень ризику.

Протокол DNS через HTTPS - це просто процес надсилання запитів DNS і отримання відповідей DNS, але зашифрований через HTTPS. У результаті ті, хто спостерігає за гарячою картоплею DNS-запитів і відповідей, не можуть зрозуміти, що запитують [17].

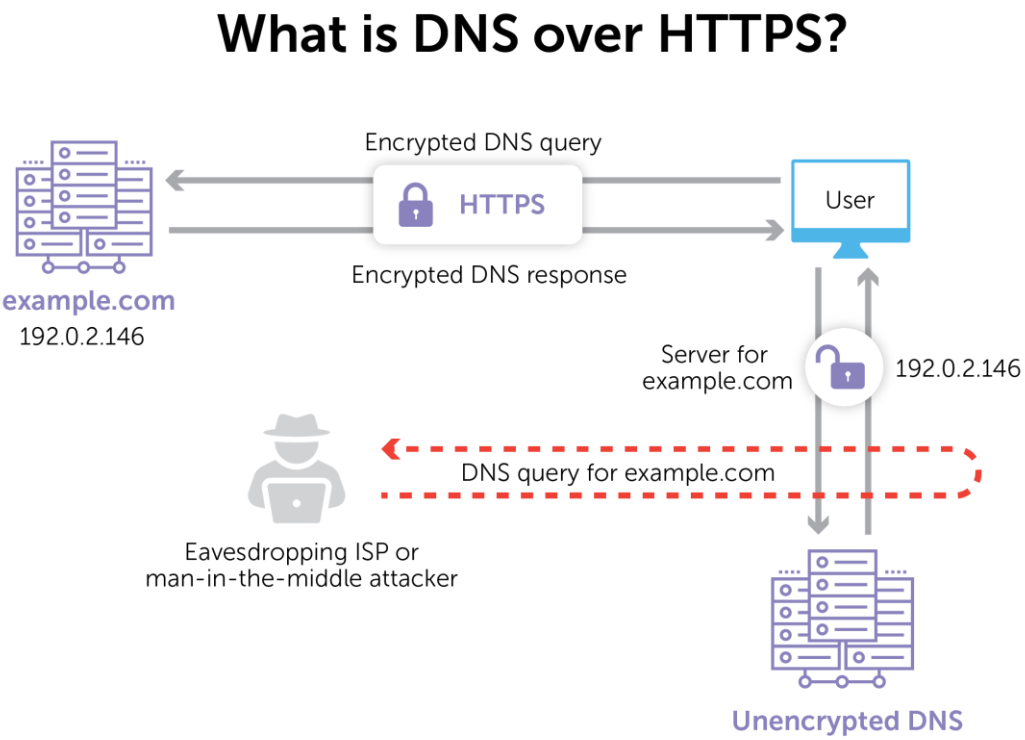



Рисунок 1.7 – Схема роботи DoH [16]

Різниця між DoT та DoH полягає, щодо DoT - основна увага тут полягає в тому, що вихідний протокол DNS не змінюється; він просто безпечно передається по захищеному каналу, а DoH інкапсулює DNS у формат HTTP перед надсиланням запитів.

Отже, системи виявлення аномалій, особливо ті, що використовують алгоритми машинного навчання, є передовим способом виявлення та пом'якшення наслідків отруєння кешу DNS. Аналізуючи шаблони трафіку DNS, ці системи можуть виявляти аномалії, які вказують на атаки отруєння кешу. Ефективність цих систем залежить від якості навчальних даних і здатності алгоритмів адаптуватися до нових векторів атак.

Гібридні підходи поєднують кілька методів, щоб забезпечити кілька рівнів захисту від отруєння кешу DNS. Наприклад, системи виявлення аномалій можуть бути розгорнуті паралельно з DNSSEC, щоб забезпечити як перевірку DNS-відповідей, так і виявлення патернів аномалій, які вказують на наявність атаки. Такий багаторівневий підхід забезпечує надійний захист, але вимагає ретельного

налаштування та управління для оптимізації ефективності та мінімізації помилкових спрацьовувань [18].

### 1.4 Огляд існуючих рішень у сфері виявлення аномальних патернів з використанням машинного навчання

Поширення кіберзагроз, особливо складних атак на систему доменних імен (DNS), створило потребу в передових механізмах виявлення для ефективного виявлення та пом'якшення таких загроз. В останні роки машинне навчання (ML) стало ключовою технологією для посилення заходів кібербезпеки, особливо у виявленні аномальних патернів у мережевому трафіку, які можуть свідчити про наявність вразливостей у системі безпеки DNS.

Суть машинного навчання полягає в наданні інструментів для автоматичного виявлення патернів у даних, які відхиляються від норми. У контексті безпеки DNS це передбачає аналіз великих обсягів даних мережевого трафіку для виявлення аномалій, які можуть сигналізувати про такі загрози, як атаки отруєння кешу DNS. Існують різноманітні методи ML, включаючи моделі керованого навчання, такі як дерева рішень і машини опорних векторів, а також методи некерованого навчання, такі як кластеризація і системи виявлення аномалій на основі нейронних мереж.

Моделі керованого навчання навчаються на маркованих наборах даних, що містять нормальні та аномальні шаблони DNS-трафіку. Після навчання ці моделі можуть з високою точністю класифікувати нові дані як нормальні або потенційно шкідливі. Наприклад, дерева рішень довели свою ефективність у визначенні конкретних векторів атак, а машини опорних векторів особливо добре підходять для аналізу DNS-трафіку завдяки їх високій стабільності в багатовимірних просторах.

Моделі неконтрольованого навчання у свою чергу не потребують маркованого набору даних і можуть досліджувати внутрішню структуру даних для виявлення викидів. Наприклад, алгоритми кластеризації групують схожі точки даних і позначають точки даних, які не вписуються в жодну групу, як викиди. Глибоке навчання, підмножина ML, що включає складні нейронні мережі,

продемонструвало чудовий потенціал у виявленні найтонших аномалій у структурі трафіку DNS, навчившись розпізнавати складні шаблони і взаємозв'язки в даних.

Практичне застосування ML в безпеці DNS стикається з низкою проблем, включаючи необхідність великих, точно маркованих наборів даних для навчання моделей, обчислювальну складність навчання і розгортання моделей ML, а також динамічний характер кіберзагроз, які вимагають постійного оновлення моделей. Динамічна природа кіберзагроз, наприклад, створює низку проблем. Незважаючи на ці виклики, ефективність ML у підвищенні безпеки DNS була продемонстрована численними дослідженнями та реальними застосуваннями, оскільки системи на основі ML успішно виявили та пом'якшили багато загроз DNS.

Існують різноманітні рішення у сфері виявлення аномальних патернів з використанням машинного навчання. Кілька найбільш відомих методів та інструментів це:

– різні алгоритми класифікації, такі як наївний Баєсів класифікатор, метод опорних векторів (SVM), дерева рішень та їхні ансамблі, можуть використовуватися для виявлення аномальних патернів в мережевому трафіку або інших даних;

– алгоритми кластеризації та кластерні методи, такі як k-means або DBSCAN, можуть виявляти групи аномальних даних, які відрізняються від типових патернів;

– глибокі нейронні мережі, зокрема з використанням згорткових та рекурентних шарів, можуть ефективно аналізувати складні дані та виявляти аномальні патерни в реальному часі;

– автокодувальні нейронні мережі можуть використовуватися для виявлення аномальних патернів шляхом порівняння вхідних даних з їхніми реконструйованими версіями;

– методи виявлення зміни підпису (Change Point Detection). Ці методи використовуються для виявлення моментів в часовому ряді, коли спостерігається суттєва зміна, що може свідчити про аномальність.

Ці рішення можуть бути використані як самостійні системи виявлення аномалій або вбудовані в існуючі рішення забезпечення безпеки, такі як системи моніторингу мереж або системи виявлення вторгнень. Розглянемо детальніше рекурентну нейронну мережу, для подальшого використання та аналізу.

Рекурентна нейронна мережа (RNN) - це тип нейронної мережі, де вихідні дані попереднього кроку використовуються як вхідні дані для поточного кроку. У традиційних нейронних мережах усі входи та виходи незалежні. Однак, коли потрібно передбачити наступне слово в реченні, важливо враховувати попередні слова, що вимагає їх запам'ятовування (рис. 1.8) [9].

Головною особливістю RNN є її прихований стан, який зберігає інформацію про послідовність. Цей стан також називають станом пам'яті, оскільки він запам'ятовує попередній вхід до мережі. RNN використовує однакові параметри для кожного входу, оскільки виконує однакове завдання на всіх входах або прихованих шарах для створення виходу. Це знижує складність параметрів порівняно з іншими нейронними мережами.

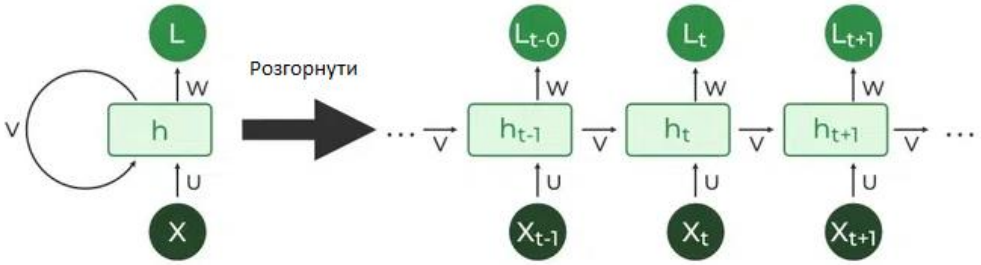



Рисунок 1.8 - Рекурентна нейронна мережа [9]

Представлена діаграма ілюструє архітектуру рекурентної нейронної мережі (RNN).

Основними компонентами моделі є вхідний вектор x(t) на кожному часовому кроці t, прихований стан h(t), що інкапсулює інформацію про попередні входи,

вагові матриці W, U та V, які є параметрами мережі, а також функції активації, зазвичай нелінійні.

Процес обробки даних здійснюється рекурентно. На кожному кроці вхідний вектор x(t) та попередній прихований стан h(t-1) надходять до комірки пам'яті. Використовуючи вагові матриці W та U, а також функцію активації, обчислюється новий прихований стан h(t). Цей стан слугує вхідним для наступного часового кроку та використовується для генерування виходу за допомогою вагової матриці V та функції активації L.

Потрібно згадати, що ключовою особливістю RNN є її здатність зберігати та оновлювати стан, що дозволяє вловлювати довгострокові залежності в послідовних даних.

Основним процесором у рекурентній нейронній мережі (RNN) є рекурентний блок (рис. 1.9), який явно не називається «рекурентним нейроном». Цей пристрій має унікальну здатність підтримувати прихований стан, дозволяючи мережі фіксувати послідовні залежності, запам’ятовуючи попередні вхідні дані під час обробки.

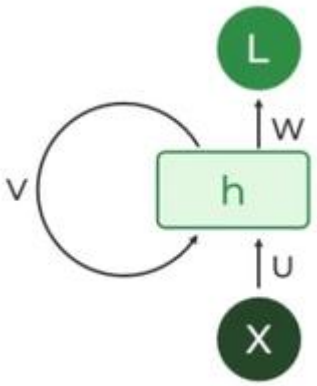



Рисунок 1.9 – Схема рекурентного нейрону [9]

Існує чотири типи RNN на основі кількості входів і виходів у мережі:

– один до одного;
– один до багатьох;
– багато до одного;

– багато до багатьох.

Розглянемо детальніше саме LSMT, адже для вирішення поставленої задачі буде використано вище згадану рекурентну нейронну мережу (RNN), а саме архітектуру LSTM (Long Short-Term Memory).

Специфічна структура LSTM допомагає подолати проблему зникаючого градієнту, характерну для звичайних RNN. У традиційних RNN є лише один прихований стан, що передається через час, що ускладнює вивчення довгострокових залежностей. LSTM вирішують цю проблему завдяки введенню комірки пам'яті, яка може зберігати інформацію протягом тривалого часу. Є 3 види воріт [22]:

– забудьте про ворота (Forget Gate) (рис. 1.10);

Два входи $x_t$ (вхід у певний час) і $h_{t-1}$ (попередній вихід комірки) подаються на вентиль і множаться на вагові матриці з наступним додаванням зсуву. Результат передається через функцію активації, яка дає двійковий вихід. Якщо для певного стану комірки виведення дорівнює 0, частина інформації забувається, а для виходу 1 інформація зберігається для використання в майбутньому. Рівняння для даних воріт:

$$f_t = (W_f[h_{t-1}, x_t] + b_f) \tag{1.1}$$

де $W_f$ - представляє вагову матрицю, пов’язану з воротами забуття

$[h_{t-1}, x_t]$ - позначає конкатенацію поточного введення та попереднього прихованого стану

$b_f$ - зміщення із забутими воротами

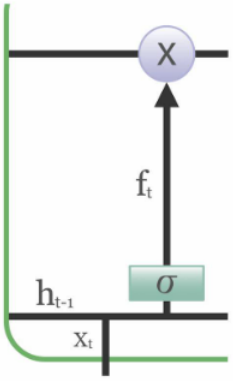



Рисунок 1.10 – Схема забудьте про ворота (Forget Gate) [20]

– вхідні ворота (Input gate) (рис. 1.11);

Додавання корисної інформації до стану комірки виконується вхідним вентилем. По-перше, інформація регулюється за допомогою сигмоїдної функції та фільтрує значення, які потрібно запам'ятати, подібно до пропуску забування за допомогою входів $h_{t-1}$ і $x_t$. Потім за допомогою функції tanh створюється вектор , який дає результат від -1 до +1, який містить усі можливі значення з $h_{t-1}$ і $x_t$ . Нарешті, значення вектора та регульовані значення перемножуються для отримання корисної інформації. Рівняння для вхідного вентиля таке:

$$i_t = (W_i[h_{t-1}, x_t] + b_i) \tag{1.2}$$

$$C_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{1.3}$$

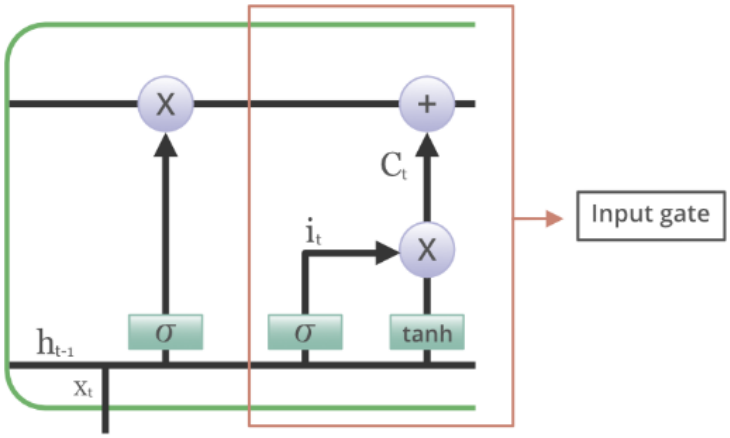



Рисунок 1.11 - Схема вхідні ворота (Input Gate) [20]

– вихідні ворота (Output gate) (рис. 1.12).

Завдання вилучення корисної інформації з поточного стану комірки для представлення як виведення виконується вихідним вентилем. Спочатку генерується вектор шляхом застосування функції $tanh$ до клітинки. Потім інформація регулюється за допомогою сигмоїдної функції та фільтрується за значеннями, які потрібно запам'ятати, за допомогою входів $h_{t-1}$ і $x_t$ . Нарешті, значення вектора та регульовані значення множаться для надсилання як вихідних і вхідних даних до наступної комірки. Рівняння для вихідного вентиля таке:

$$o_t = (W_o[h_{t-1}, x_t] + b_0) \tag{1.4}$$

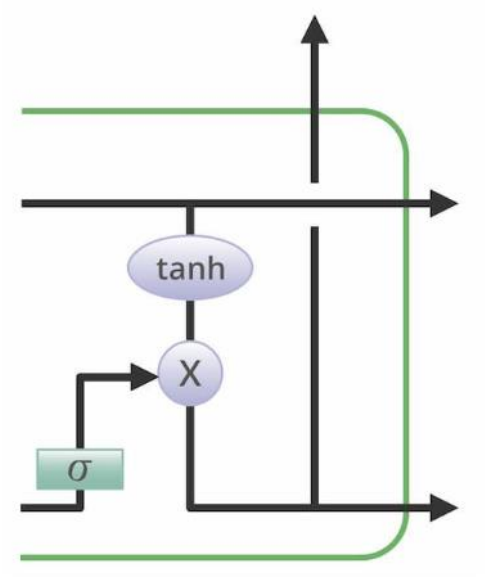



Рисунок 1.12 - Схема вихідні ворота (Output Gate) [20]

Також, існують різноманітні програмні розробки та бібліотеки, які можна використовувати для виявлення аномальних патернів з використанням машинного навчання. Ось деякі з найпопулярніших:

Scikit-learn це одна з найпопулярніших бібліотек для машинного навчання у Python. Вона містить різні алгоритми класифікації, кластеризації, регресії та виявлення аномалій, які можуть бути використані для аналізу даних та виявлення аномальних патернів.

TensorFlow / Keras це потужні бібліотеки для розробки та навчання нейронних мереж. Вони надають широкий набір інструментів для створення різних типів моделей машинного навчання, включаючи аномальну детекцію.

PyOD (Python Outlier Detection) це спеціалізована бібліотека для виявлення аномалій у даних, яка містить різні алгоритми, такі як Isolation Forest, Local Outlier Factor, One-Class SVM та інші.

Anomaly Detection with Python (ADTK) це бібліотека, яка надає простий інтерфейс для виявлення аномалій у часових рядах даних, використовуючи різноманітні статистичні методи та машинне навчання.

H2O.ai це відкрите програмне забезпечення для машинного навчання, яке містить різні алгоритми для виявлення аномалій та аналізу даних.

PHP-ML є бібліотекою машинного навчання для PHP, що підтримує основні алгоритми класифікації, регресії, кластеризації та виявлення аномалій.

Наведемо порівняльну характеристику даних існуючих методів у табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика програмних розробок та бібліотек [21]

| Інструмент | Мова програмування | Підтримувані алгоритми | Особливості |
|---|---|---|---|
| Scikit-learn | Python | Класифікація, кластеризація, регресія, виявлення аномалій | Широкий вибір алгоритмів, проста у використанні, добра документація |
| TensorFlow/Keras | Python | Глибокі нейронні мережі, включаючи згорткові та рекурентні | Потужний інструмент для глибокого навчання, гнучкість у створенні складних моделей |
| PyOD | Python | Isolation Forest, Local Outlier Factor, One-Class SVM та інші | Спеціалізована бібліотека для виявлення аномалій |
| ADTK | Python | Статистичні методи та машинне навчання для часових рядів | Простий інтерфейс для виявлення аномалій у часових рядах |
| H2O.ai | Python, R, Java, Scala | Класифікація, регресія, кластеризація, виявлення аномалій | Масштабована платформа для машинного навчання, підтримка розподілених обчислень |
| PHP-ML | PHP | Класифікація, регресія, кластеризація, виявлення аномалій | Бібліотека машинного навчання для PHP, проста інтеграція, підходить для веб-розробки |

Ці програмні розробки можуть бути використані для реалізації різних методів виявлення аномалій з використанням машинного навчання та їхнього застосування в практичних проектах.

Отже, використання машинного навчання для виявлення аномальних патернів у трафіку DNS є значним досягненням у сфері кібербезпеки. Використовуючи моделі керованого навчання, некерованого навчання та глибокого навчання, фахівці з кібербезпеки можуть покращити виявлення та пом'якшення загроз, пов'язаних з DNS. Незважаючи на труднощі впровадження, переваги систем виявлення аномалій на основі ML у підвищенні безпеки DNS очевидні. Тому продовження досліджень і розробок у цій галузі необхідне для забезпечення безпеки і цілісності інфраструктури Інтернету перед обличчям дедалі витонченіших кіберзагроз.

### 1.5 Висновки та постановка задач

Отже, в даному розділі був здійснений загальний аналіз проблеми DNS Cache Poisoning, а саме аналіз актуальності проблеми DNS Cache Poisoning у контексті сучасних кіберзагроз, розглянуто існуючі загрози DNS-системи та їх можливі наслідки, проведено аналіз методів та інструментів захисту від атак DNS Cache Poisoning та оглянуто існуючих рішень у сфері виявлення аномальних патернів з використанням машинного навчання.

Значущість системи доменних імен (DNS) для функціонування Інтернету робить її привабливою мішенню для зловмисників, що використовують тактики отруєння кешу для перенаправлення користувачів на шкідливі сайти, що може призвести до втрати даних та порушення конфіденційності. Традиційні методи захисту, такі як статичні брандмауери і системи виявлення за сигнатурами, виявляються неефективними проти динамічних і адаптивних характерів цих загроз, що зумовлює необхідність використання передових технологій як машинне навчання для виявлення аномалій і проактивне управління безпекою.

Зловмисники, використовуючи ці атаки, намагаються вплинути на процес перетворення доменних імен у відповідні IP-адреси, що може призвести до серйозних наслідків для безпеки мереж та конфіденційності користувачів.

Для ефективного захисту від таких атак необхідно використовувати комплексний підхід, який включає в себе використання різноманітних методів та

інструментів. Машинне навчання виявляється одним із потужних інструментів, які можуть бути використані для виявлення аномальних патернів у мережевому трафіку та ефективного реагування на них.

Виходячи з отриманих результатів дослідження, було сформовано подальші задачі роботи:

– розробка структури та роботи системи від атак DNS Cache Poisoning; також машинного навчання для виявлення даного типу атак, обґрунтувавши вибір алгоритму та методів машинного навчання для ефективного виявлення аномалій;

– вибір та обґрунтування обраних засобів програмування для розробки;

– розробка графічного інтерфейсу користувача, реалізація програмного засобу та тестування отриманого продукту.

Виконання поставлених завдань сприятиме досягненню основної мети роботи, що полягає у розробці системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів.

# 2 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ЗАХИСТУ ВІД DNS CACHE POISONING

В даному розділі опишемо розробку системи захисту від DNS Cache Poisoning, що включатиме такі етапи: розробку архітектури системи для виявлення аномальних патернів в DNS трафіку, розробку моделі машинного навчання для виявлення даних атак та проведемо інтеграцію розробленої системи з існуючими заходами безпеки DNS.

Також на основі вибраних засобів проектування буде здійснено аналіз вибору засобів програмування та його обґрунтування для подальшої практичної реалізації системи.

## 2.1 Розробка архітектури інтелектуальної системи для виявлення аномальних патернів в DNS-трафіку

Для розробки інтелектуальної архітектури системи, яка виявляє аномальні патерни в DNS-трафіку, необхідно спроектувати діаграму активності для візуалізації та аналізу процесів і взаємодії компонентів системи.

Діаграми діяльності - це тип діаграм, що використовується для моделювання послідовних кроків або дій у процесі. Діаграми діяльності використовуються для візуалізації послідовності подій, дій та обміну інформацією в системі. Діаграми діяльності часто використовуються для моделювання бізнес-процесів, програмного забезпечення або інженерних систем, щоб зрозуміти і відобразити їх логіку і функції.

Оскільки метою даної роботи є розробка системи захисту від DNS Cache Poisoning, головна задача роботи є розробка сервісу, що надасть користувачеві можливість виявляти аномальні патерни в DNS-трафіку.

Виходячи з мети системи, опишемо структуру сервісу наступним чином, розбивши її на етапи:

– отримання DNS-трафіку від користувачів або мережевих пристроїв;
– передача до системи аналізу;

– система використовує алгоритми машинного навчання для виявлення незвичайних патернів у DNS-трафіку;

– виявлення аномальних або підозрілих шаблонів у трафіку;

– створення звітів або сповіщень;

– виконання дій блокування;

– моніторинг та оновлення.

Залежно від серйозності аномалії, система може автоматично вжити заходів для мінімізації потенційних загроз. Про виявлені аномалії можуть бути створені звіти або надіслані оповіщення адміністратору для подальших дій.

Описану структура взаємодії користувача з додатком можна відобразити за допомогою схеми, що наведена на (рис. 2.1).

Крок 1. Процес починається з ініціації системи, яка приводить її в готовність до обробки запитів. Цей крок є відправною точкою для подальших операцій.

Крок 2. Користувач перенаправляється на сторінку авторизації для входу в систему, де йому надається можливість ввести свої облікові дані для доступу до системи. Це забезпечує початковий рівень безпеки, необхідний для запобігання несанкціонованого доступу.

Крок 3. Користувач вводить свої облікові дані, такі як логін та пароль. Система проводить валідацію введених даних шляхом перевірки їх відповідності записам у базі даних користувачів.

Крок 4. Перевірка введених облікових даних на відповідність. Відбувається перевірка коректності введених облікових даних. Результат перевірки визначає подальший хід процесу:

– якщо авторизація успішна, процес переходить до надання доступу до функціоналу системи;

– якщо авторизація неуспішна, користувачу відображається повідомлення про невдалу спробу авторизації.

Крок 5. У випадку невдалого входу користувач отримує повідомлення про помилку і повертається до сторінки авторизації для повторної спроби. Після цього

користувач може повторити спробу авторизації, повернувшись до відповідної форми.

Крок 6. У разі успішної авторизації користувачу надається доступ до функціоналу системи.

Крок 7. Система отримує вхідний DNS-запит від користувача. Цей запит містить інформацію, яку необхідно перевірити на достовірність та безпеку.

Крок 8. Система перевіряє отримані DNS-запити на достовірність, аналізуючи їхній вміст. Це включає перевірку відповідності запиту відомим безпечним паттернам та перевірку на наявність аномалій.

Крок 9. Система використовує алгоритми машинного навчання для аналізу ймовірності того, що даний запит є частиною атаки. Для цього використовуються навчальні моделі, які попередньо навчені на великому обсязі даних про атаки та нормальні запити.

Крок 10. Система визначає, чи є запит частиною атаки "отруєння кешу DNS".

– якщо система ідентифікує атаку, запит буде заблоковано;

– якщо запит визнано безпечним, він буде пропущений для подальшої обробки.

Крок 11. У разі виявлення атаки система блокує DNS-запит. Це запобігає можливості здійснення шкідливих дій, пов'язаних з отруєнням кешу DNS.

Крок 12. Якщо запит визнано безпечним, система пропускає його, дозволяючи обробляти дані відповідно до стандартного процесу.

Крок 13. Процес завершується, і система повертається до початкового стану, готова обробити наступні запити. Це забезпечує безперервний моніторинг та захист від можливих атак у реальному часі.

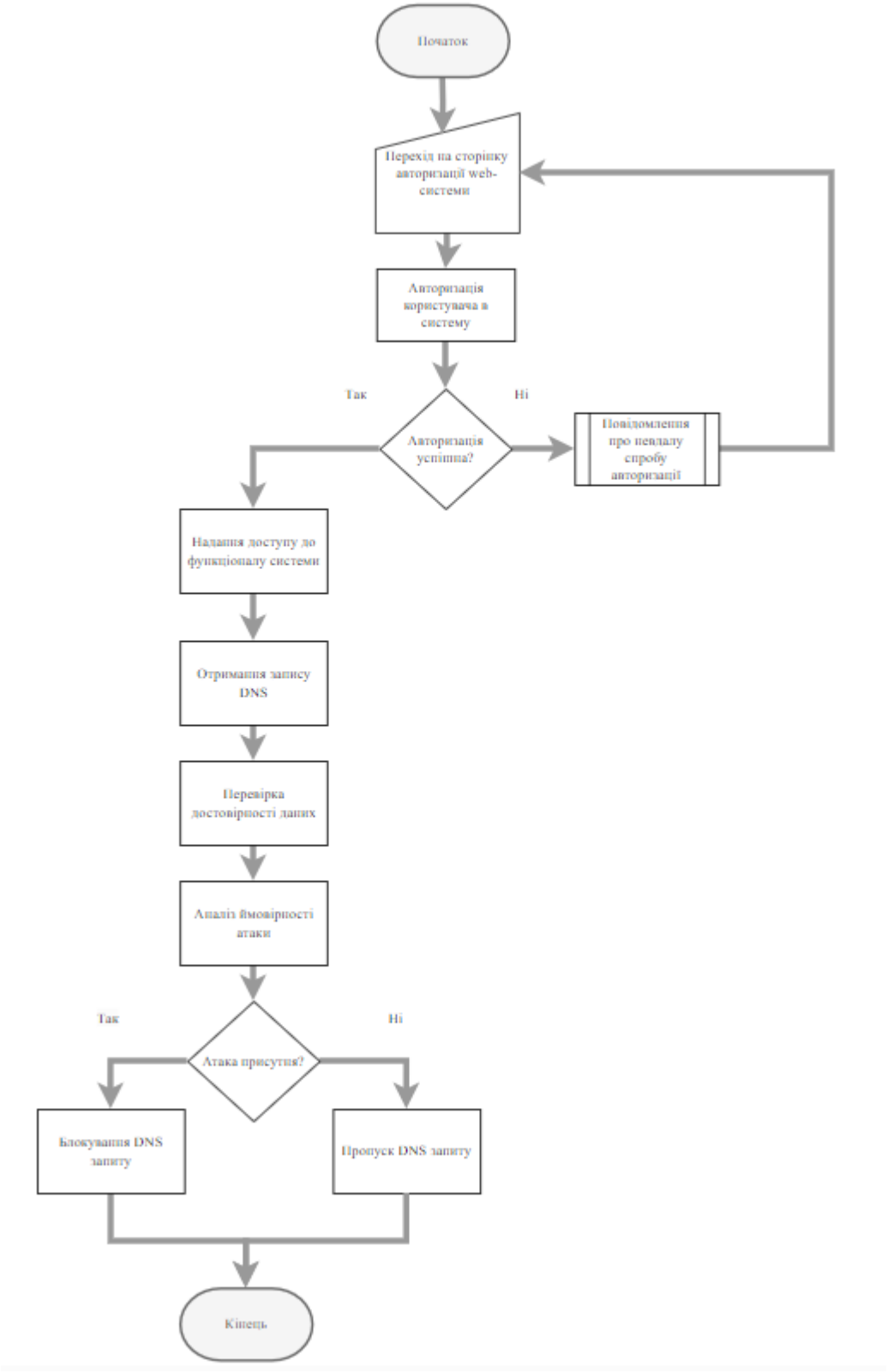



Рисунок 2.1 – Схема взаємодії користувача з розробленою системою

На етапі моніторингу відбувається збір даних про DNS-трафік у мережі за допомогою різних засобів моніторингу та захоплення трафіку.

Вхідні дані отримуються від системи моніторингу та проходять первинну обробку, таку як фільтрація, нормалізація та перетворення даних у формат, придатний для подальшого аналізу.

Кластеризація та виявлення аномалій - це ключовий етап, на якому застосовуються алгоритми машинного навчання та аналізу даних для виявлення відхилень від нормального трафіку. Передбачається групування (кластеризація) даних на основі подібних характеристик.

На кроці перевірки на аномалії система визначає, чи є виявлене відхилення справжньою аномалією, що вимагає втручання, чи це легітимна поведінка. Якщо відхилення класифіковано як аномалію, система ініціює процес оповіщення адміністратора через відповідні канали сповіщення. Адміністратор має можливість проаналізувати інформацію про аномалію та вжити необхідних дій, наприклад, заблокувати певні IP-адреси чи домени. Результати дій адміністратора записуються до бази даних разом з іншими деталями виявленої аномалії. Якщо ж відхилення не було класифіковано як аномалія, дані про нього все одно записуються до бази даних для подальшого аналізу та навчання моделей.

Вихідні дані формуються на основі зібраної інформації про DNS-трафік, виявлені аномалії та дії адміністратора. Ці дані можуть використовуватися для подальшого аналізу, покращення моделей та звітності.

Отже, ця схема охоплює весь процес від моніторингу DNS-трафіку до виявлення аномалій, оповіщення адміністратора та збереження даних для подальшої обробки.

Було описано алгоритм роботи з DNSSEC та DNS та покроково розписано (рис. 2.2, 2.3).

Крок 1. Процес розпочинається з ініціалізації, яка може включати налаштування системи та готовність до обробки DNSSEC записів.

Крок 2. На цьому етапі система отримує запис DNSSEC. Це може бути будь-який DNS-запит або відповідь, який включає інформацію про безпеку домену.

Крок 3. Після отримання запису здійснюється перевірка його достовірності. Це включає перевірку цифрових підписів, щоб підтвердити, що запис не був змінений

і є дійсним.

Крок 4. Далі система аналізує ймовірність атаки. Цей крок передбачає використання алгоритмів машинного навчання або інших методів для визначення, чи є поточний запис підозрілим або аномальним, що може вказувати на атаку "отруєння кешу DNS".

Крок 5. На цьому етапі система приймає рішення про наявність атаки. Виходячи з аналізу, система визначає, чи є запис підозрілим і чи існує загроза атаки.

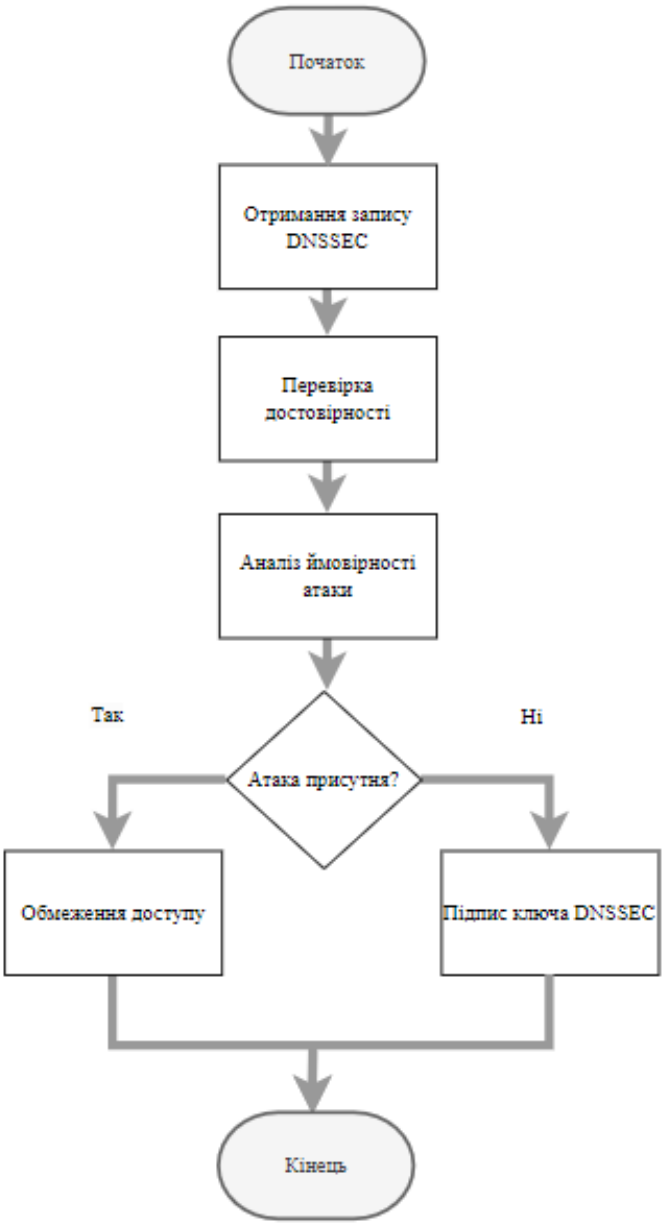



Рисунок 2.2 – Алгоритм роботи з DNSSEC розробленої системи

Крок 6. Якщо система виявляє наявність атаки, вона обмежує доступ, щоб захистити систему від потенційного компромісу. Це може включати блокування шкідливого трафіку або ізоляцію підозрілого запису.

Крок 7. Якщо система не виявляє ознак атаки, вона підписує ключ DNSSEC для поточного запису, підтверджуючи його достовірність і дозволяючи його подальше використання.

Крок 8. Процес завершується після прийняття рішення та виконання відповідних дій. Система повертається у початковий стан, готова обробляти нові записи DNSSEC.

Загалом дана схема відображає процес обробки та перевірки записів DNSSEC з метою виявлення та реагування на атаки "отруєння кешу DNS". Кожен крок забезпечує надійність та безпеку системи, використовуючи методи перевірки достовірності, аналізу аномалій та відповідного реагування на виявлені загрози.

Та опишемо алгоритм роботи з DNS.

Крок 1. Процес розпочинається з ініціалізації, яка передбачає підготовку системи до перевірки DNS-запитів. Це може включати налаштування системи та готовність до обробки вхідного мережевого трафіку.

Крок 2. На цьому етапі здійснюється перевірка отриманого DNS-запиту. Система аналізує запит для визначення його достовірності та цілісності, використовуючи різні методи верифікації, такі як перевірка цифрових підписів або інших механізмів захисту.

Крок 3. Після перевірки DNS-запиту система оцінює, чи присутні ознаки атаки. Це може включати аналіз аномалій, відхилень від звичайної поведінки або інших індикаторів, які можуть свідчити про можливу атаку.

Крок 4. Після перевірки DNS-запиту система оцінює, чи існує запит у системі.

Крок 5. Якщо система визначає, що DNS-запит існує, вона переходить до аналізу існуючого трафіку для подальшої оцінки ймовірності атаки "Cache Poisoning".

Крок 6. Якщо система не виявляє наявність DNS-запиту, результати аналізу зберігаються, і система переходить до стандартної процедури обробки трафіку без

додаткових заходів.

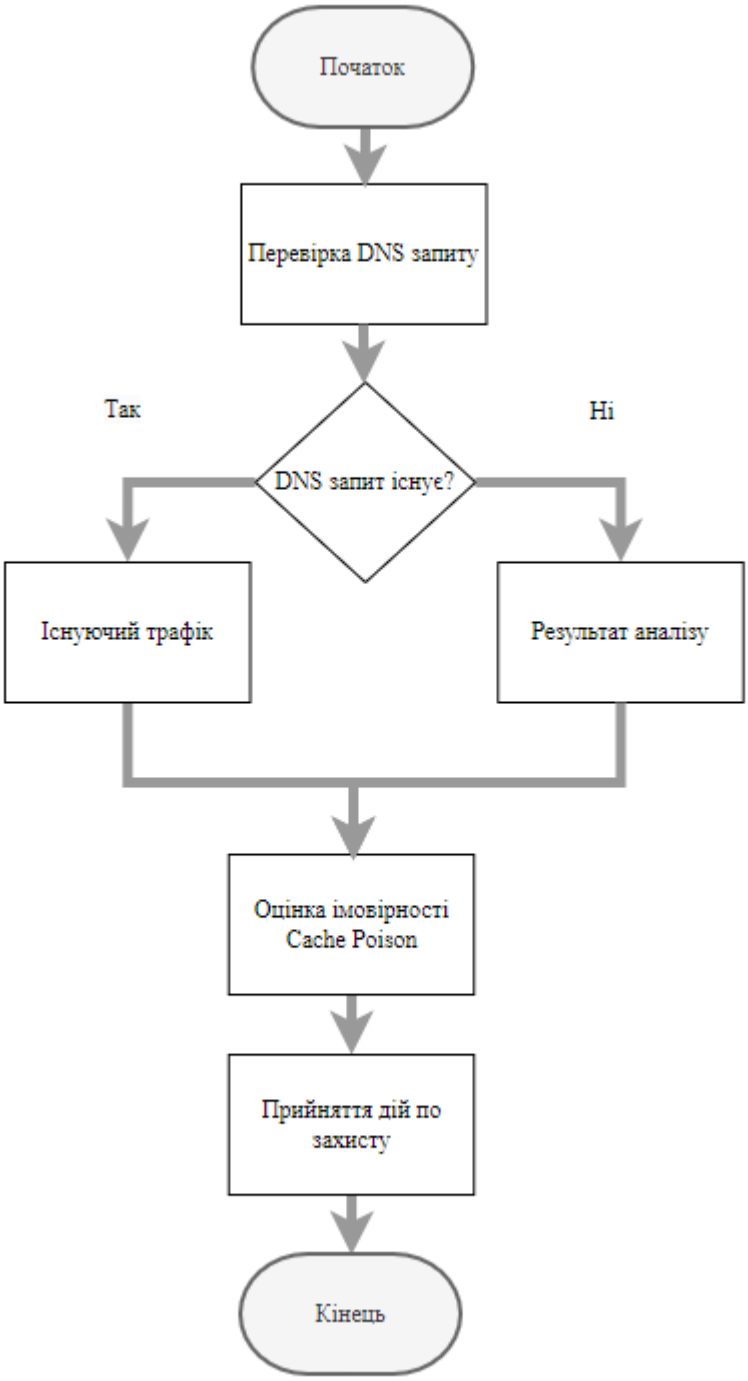



Рисунок 2.3 – Алгоритм роботи з DNS розробленої системи

Крок 7. На цьому етапі, якщо DNS-запит існує, здійснюється детальна оцінка ймовірності "Cache Poisoning". Це включає глибокий аналіз вхідного та існуючого трафіку для визначення потенційних загроз і рівня ризику.

Крок 8. На основі результатів оцінки ймовірності "Cache Poisoning"

приймаються відповідні дії по захисту. Це може включати блокування шкідливого трафіку, оновлення налаштувань безпеки, ізоляцію підозрілих запитів та інші заходи для захисту системи від атаки.

Крок 9. Процес завершується після прийняття відповідних дій. Система повертається у початковий стан, готова обробляти нові DNS-запити та забезпечувати безпеку мережевого трафіку.

Ця схема відображає процес перевірки DNS-запитів з метою виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning). Кожен крок спрямований на забезпечення надійності та безпеки системи через перевірку, аналіз та відповідне реагування на виявлені загрози.

Розробимо захищену базу даних для інтелектуальної архітектури системи, яка виявляє аномальні патерни в DNS-трафіку, що є невід’ємною складовою даного програмного продукту.

Для створення бази даних потрібні такі сутності: СПОВІЩЕННЯ, КОРИСТУВАЧІ, ПОВІДОМЛЕННЯ, DNS-ТРАФІК, DNS-ІДЕНТИФІКАТОР, ІДЕНТИФІКАТОР БЕЗПЕКИ, ТРАФІК БЕЗПЕКИ.

Схему розробленої бази даних наведемо (рис. 2.4, 2.5).

Опишемо властивості (атрибути) вказаних сутностей, що повністю розкривають зміст:

ALERT (Id, Date, Domain);

DNS_II (Id, Json, Type, Date);

DNS_TRAF (Id, Date, Warning, Domain,Type);

MESSAGE (Id, Email);

SEC_II (Id, Json, Type, Date);

SEC_TRAF (Id, Date, Warning, Domain, Class, Ttl, Type, Info).

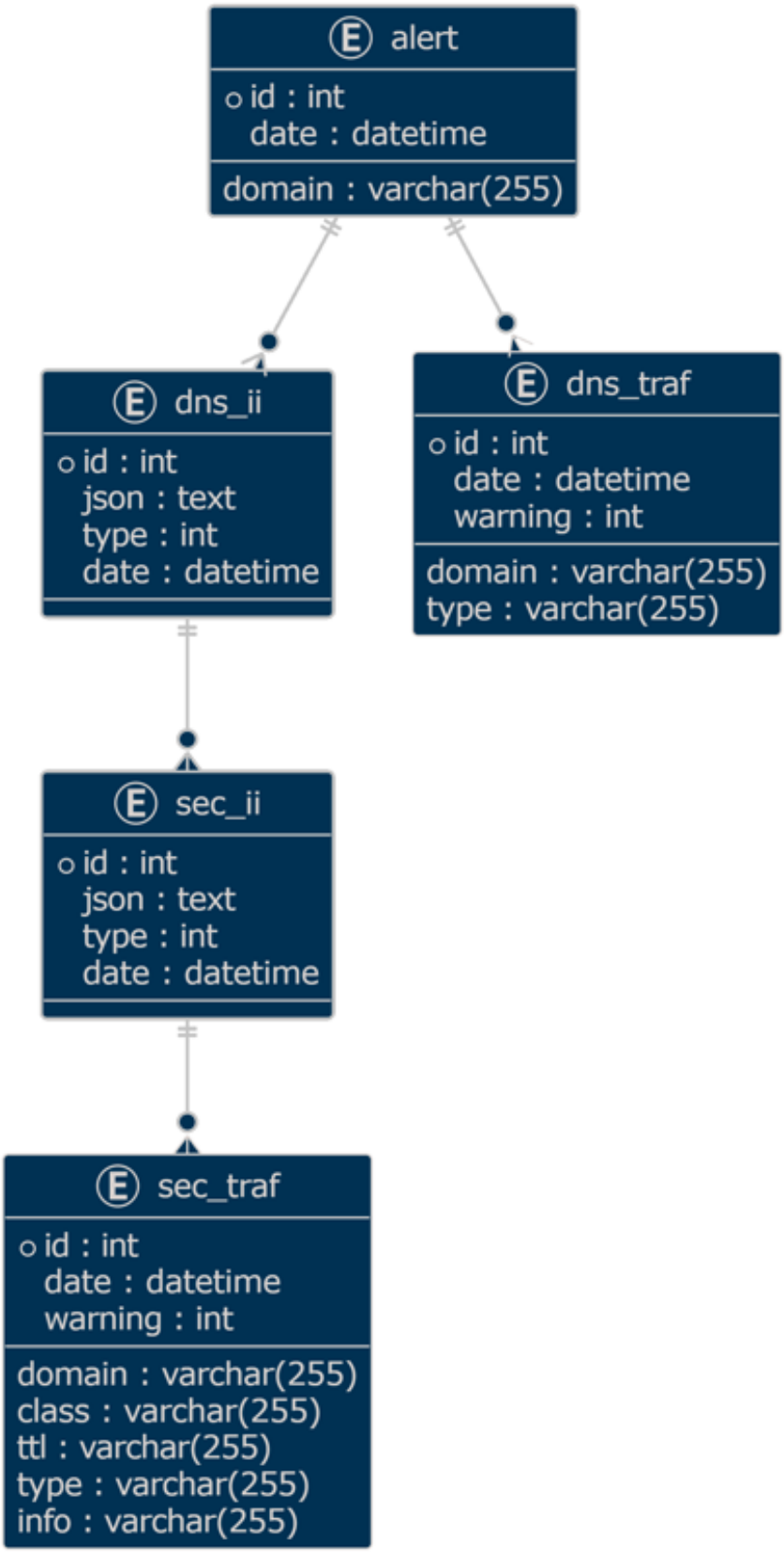



Рисунок 2.4 – Схема бази даних розробленої системи (А)


Рисунок 2.5 – Схема бази даних розробленої системи (Б)

Опишемо властивості (атрибути) вказаних сутностей, що повністю розкривають зміст:

USERS (Id, Login, Pass);

MESSAGE (Id, Email);

Загалом база даних, розроблена для комплексної системи моніторингу безпеки та керування інцидентами, здатної відстежувати та співвідносити різноманітні події безпеки, реєструвати детальну інформацію, керувати доступом користувачів і потенційно надсилати сповіщення чи сповіщення на основі виявлених загроз або аномалій.

Для взаємодії з реляційними базами даних, використовується RedBeanPHP, спрощуючи процес роботи з таблицями та записами через об'єктно-орієнтовану модель програмування.

RedBeanPHP - це легковажна об'єктно-реляційна бібліотека відображення (ORM) для мови програмування PHP. Основною метою RedBeanPHP є забезпечення простого та інтуїтивного способу маніпуляції даними в базі через використання об'єктів. Замість безпосереднього написання SQL-запитів, ця

бібліотека дозволяє працювати з даними на концептуальному рівні, абстрагуючись від деталей реалізації бази даних. Наприклад, для створення нового запису в таблиці достатньо створити екземпляр відповідного об'єкта та викликати метод збереження.

Бібліотека RedBeanPHP забезпечує зручний інтерфейс для виконання основних операцій з базою даних, включаючи створення, читання, оновлення та видалення записів (CRUD). Крім того, підтримує встановлення різних типів зв'язків між таблицями, таких як один-до-одного, один-до-багатьох та багато-до-багатьох.

Однією з ключових переваг RedBeanPHP є її простота у використанні та налаштуванні. Для підключення до бази даних потрібно лише один рядок коду, а для створення складних запитів використовується зрозумілий та лаконічний синтаксис. Це робить бібліотеку привабливою для швидкої розробки додатків та прототипування.

Крім того, RedBeanPHP пропонує інструменти для управління міграціями схеми бази даних, що полегшує процес розробки та розгортання додатків, забезпечуючи відповідність структури бази даних до поточної версії коду.

Щодо безпеки бази даних, RedBeanPHP підтримує використання підготовлених запитів (prepared statements) для запобігання атакам ін'єкцій SQL. Підготовлені запити є одним з найефективніших способів захисту від ін'єкцій, зокрема відокремлюють вхідні дані від структури запиту, що унеможливлює модифікацію синтаксису SQL зловмисним кодом.

Загалом, RedBeanPHP є потужним і гнучким рішенням для роботи з базами даних у PHP-додатках, забезпечуючи зручний об'єктно-орієнтований підхід та спрощуючи процес розробки завдяки своїй простоті та виразності.

## 2.2 Проектування моделі машинного навчання для виявлення атак DNS Cache Poisoning

Для вирішення завдання виявлення атак "отруєння кешу DNS" буде використано метод навчання з вчителем (supervised learning). Зокрема, це задача

бінарної класифікації, де мета полягає у розпізнаванні аномальних патернів у потоці DNS-запитів, які можуть вказувати на атаки отруєння кешу. Підходи навчання без вчителя (unsupervised) для цієї задачі менш ефективні, оскільки важко відокремити аномалії без наявності міток даних про атаки.

Для навчання моделі буде зібрано датасет мережевого трафіку DNS, що містить приклади як нормальних запитів, так і випадків атак "отруєння кешу". Дані будуть зібрані з різних джерел, включаючи захоплений реальний трафік та синтетичні приклади згенеровані за допомогою вразливих DNS-серверів.

Усі приклади будуть розмічені відповідними мітками ("нормальний" або "атака(redflag)"). Потім дані пройдуть етап попередньої обробки, а саме нормалізація числових ознак, кодування категоріальних ознак, видалення дублікатів тощо.

Весь датасет буде розділено на навчальну (80%), валідаційну (10%) та тестову (10%) вибірки для уникнення переналаштування та оцінки узагальнюючої здатності моделі. Було описано алгоритм роботи інтелектуальної системи користувача для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) та відображено за допомогою схеми, що наведена на (рис. 2.6).

Крок 1. Початок ініціалізації процесу.

Крок 2. Відбувається збір реальних даних мережевого трафіку DNS. Генерація синтетичних даних з атаками "отруєння кешу DNS" за допомогою вразливих DNS-серверів.

Крок 3. Попередня обробка даних. Тобто відбувається розмітка даних, а саме розподіл запитів на нормальні та атаки; нормалізація числових ознак - приведення значень до одного масштабу; кодування категоріальних ознак - перетворення текстових даних у числовий формат; видалення дублікатів - усунення повторюваних записів.

Крок 4. Виділення ознак. Використання алгоритму Random Forest для визначення найбільш важливих ознак з мережевого трафіку. Формування векторів ознак для кожного DNS-запиту.

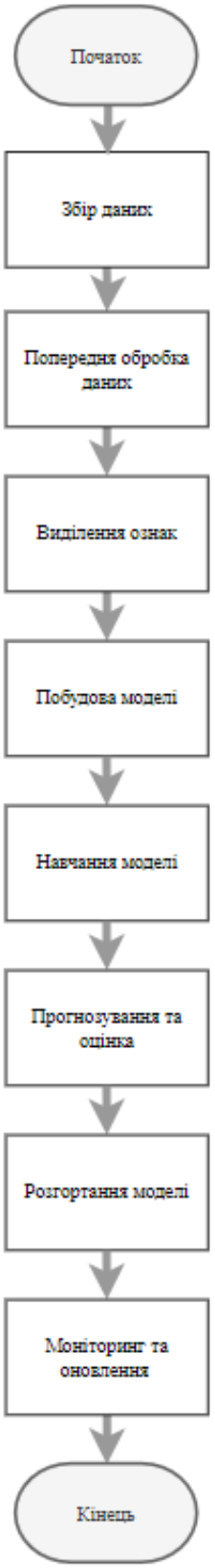



Рисунок 2.6 – Блок-схема алгоритму роботи розробленої системи

Крок 5. Вибір архітектури RNN (LSTM) для обробки послідовних даних. Конфігурація моделі "Багато до одного" (Many-to-One) для аналізу послідовностей запитів.

Крок 6. Навчання моделі на навчальній вибірці. Валідація моделі на валідаційній вибірці для перевірки її узагальнюючої здатності.

Крок 7. Прогнозування на тестовій вибірці для оцінки ефективності моделі. Оцінка точності моделі за допомогою метрик точності, повноти та F1-score.

Крок 8. Інтеграція навченої моделі в систему моніторингу мережевого трафіку для реального часу.

Крок 9. Постійний моніторинг DNS-запитів для виявлення атак. Оновлення моделі на основі нових даних для покращення її точності та ефективності.

Крок 10. Кінець. Завершення процесу і готовність до виявлення атак у реальному часі.

Для вирішення поставленої задачі буде використано рекурентну нейронну мережу (RNN), а саме архітектуру LSTM (Long Short-Term Memory). RNN добре підходять для аналізу послідовних даних, подібних до потоку DNS-запитів. Специфічна структура LSTM допомагає долати проблему зникаючого градієнту, характерну для звичайних RNN.

На основі специфіки завдання виявлення атак "отруєння кешу DNS" за допомогою аналізу потоку мережевих пакетів, найбільш підходящим типом рекурентної нейронної мережі (RNN) є "Багато до одного" (Many-to-One). (рис. 2.7).

Беручи до уваги природу завдання виявлення атак "отруєння кешу DNS" шляхом аналізу мережевого трафіку, даний тип RNN здатний опрацьовувати послідовності вхідних даних довільної довжини та генерувати єдиний прогнозований вихід на основі всієї послідовності.

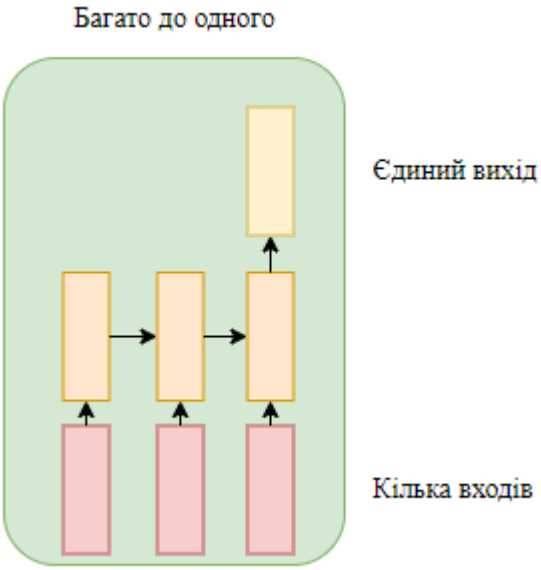



Рисунок 2.7 – Схема RNN багато до одного

У контексті зазначеної проблеми класифікації мережевого трафіку на нормальний та атакований, вхідними даними для архітектури "Багато до одного" слугуватиме серія векторів ознак, отриманих із множини окремих DNS-запитів за певний проміжок часу. Кожен вектор ознак міститиме відібрані найбільш релевантні характеристики, згенеровані попередньо за допомогою алгоритму Random Forest на основі оцінок важливості.

На кожному кроці вхідної послідовності рекурентний шар, зокрема архітектури LSTM (Long Short-Term Memory), опрацьовуватиме відповідний вектор ознак та оновлюватиме свій внутрішній прихований стан. Цей прихований стан акумулює контекстну інформацію з попередніх кроків послідовності, забезпечуючи можливість виявляти часові закономірності та аномальні патерни поведінки.

По завершенні обробки всієї послідовності DNS-запитів, кінцевий прихований стан рекурентного шару буде використано для прогнозування бінарного виходу за допомогою повнозв'язного шару з сигмоїдною функцією активації. Цей вихід представлятиме ймовірність належності всієї послідовності до класу "атака отруєння кешу DNS".

Архітектура RNN типу "Багато до одного" є доцільним вибором, оскільки дозволяє враховувати контекстну інформацію з усієї часової серії взаємопов'язаних

DNS-запитів під час прогнозування аномалій, на відміну від ізольованого розгляду окремих прикладів. Це в свою чергу сприяє більш точному виявленню складних злоякісних патернів поведінки, що можуть розгортатися протягом тривалих періодів часу.

RNN мають таку ж архітектуру введення та виведення, як і будь-яка інша глибока нейронна мережа. Проте різниця полягає у способі передачі інформації від входу до виходу. На відміну від глибоких нейронних мереж, де для кожної щільної мережі використовуються різні матриці ваг, у RNN вага залишається постійною.

Було описано детальну блок-схему LSTM-архітектури та процес навчання для розробленої системи (рис. 2.8).

Крок 1. Початок.

Крок 2. Вхідні дані представляють собою послідовність векторів ознак, отриманих з DNS-запитів за певний проміжок часу.

Крок 3. На кожному кроці вхідної послідовності рекурентний шар LSTM обробляє відповідний вектор ознак та оновлює свій внутрішній прихований стан.

Крок 4. Прихований стан акумулює контекстну інформацію з попередніх кроків послідовності, забезпечуючи можливість виявляти часові закономірності та аномальні патерни поведінки.

Крок 5. Після обробки всієї послідовності кінцевий прихований стан передається на наступний етап.

Крок 6. Повнозв'язний шар використовує кінцевий прихований стан для прогнозування бінарного виходу, що представляє ймовірність атаки.

Крок 7. Оцінка точності моделі за допомогою метрик точності, повноти та F1-score.

Крок 8. Підгонка вагових коефіцієнтів за допомогою алгоритму зворотного поширення помилки для оптимізації моделі.

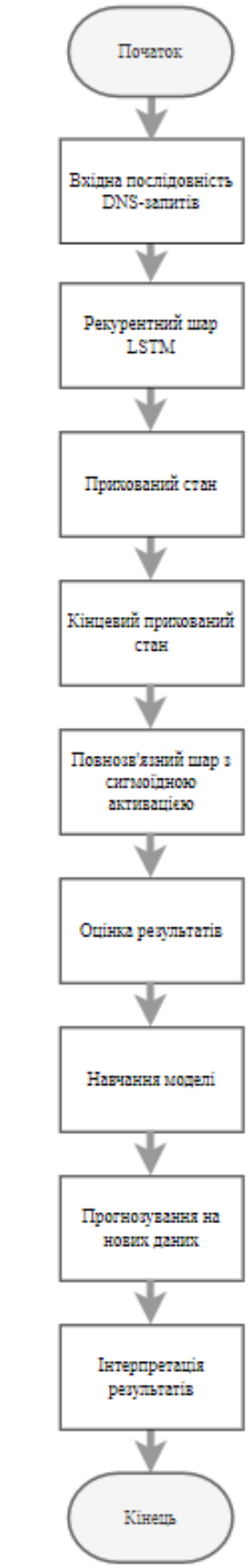



Рисунок 2.8 – Блок-схема LSTM-архітектури розробленої системи

Крок 9. Застосування навченої моделі до нових послідовностей DNS-запитів для виявлення потенційних атак.

Крок 10. Аналіз вихідних даних для виявлення атак "отруєння кешу DNS".

Крок 11. Кінець. Завершення процесу і готовність до виявлення атак у реальному часі.

Для навчання нейронної мережі буде використано базу даних CICIDS, яка містить різноманітні типи мережевих атак, включаючи атаки DNS. Вона є однією з найбільш комплексних і широко використовуваних наборів даних для навчання моделей машинного навчання у сфері кібербезпеки [23]. Приклад даних наведено у таблиці 2.1

Таблиця 2.1 – Приклад даних для навчання LSTM нейронної мережі

| Timestamp | Source IP | Destination IP | Source Port | Destination Port | Protocol | Flow Duration | Total Fwd Packets | Total Backwards Packets | Label |
|---|---|---|---|---|---|---|---|---|---|
| 2021-07-05 08:17:21 | 192.168.10.50 | 205.174.165.68 | 49168 | 80 | TCP | 1.2 | 8 | 10 | BENIGN |
| 2021-07-05 08:17:21 | 192.168.10.51 | 172.16.0.1 | 57621 | 53 | UDP | 0.3 | 2 | 0 | DNS |
| 2021-07-05 08:17:21 | 192.168.10.52 | 192.168.10.1 | 443 | 445 | TCP | 3.5 | 30 | 25 | BENIGN |
| 2021-07-05 08:17:21 | 192.168.10.53 | 192.168.10.2 | 49159 | 135 | TCP | 0.5 | 3 | 2 | DNSPoisoning |
| 2017-07-05 08:17:21 | 192.168.10.54 | 172.16.0.1 | 49000 | 53 | UDP | 0.4 | 2 | 0 | DNS |

Timestamp - час, коли було зафіксовано запит; Source IP - IP-адреса джерела запиту; Destination IP - IP-адреса призначення запиту; Source Port - порт джерела запиту; Destination Port -порт призначення запиту; Protocol - протокол, що використовується для запиту (наприклад, TCP або UDP); Flow Duration - тривалість потоку даних; Total Fwd Packets - загальна кількість пакетів, що були відправлені з джерела до призначення; Total Backward Packets - загальна кількість пакетів, що були відправлені з призначення до джерела; Label - мітка, що вказує на тип трафіку (наприклад, BENIGN для нормального трафіку або DNSPoisoning для атаки).

Наведено блок-схему процесу навчання LSTM нейронної мережі (рис. 2.9).

Крок 1. Ініціалізація процесу.

Крок 2. Завантаження набору даних CICIDS, відбір DNS-запитів та їх розмітка.

Крок 3. Нормалізація числових ознак, кодування категоріальних ознак, видалення дублікатів, розподіл даних на вибірки.

Крок 4. Визначення важливих ознак та формування векторів ознак для кожного DNS-запиту.

Крок 5. Вибір параметрів моделі (кількість шарів, кількість нейронів, функції активації, алгоритми оптимізації).

Крок 6. Навчання на навчальній вибірці, валідація на валідаційній вибірці, налаштування вагових коефіцієнтів.

Крок 7. Прогнозування на тестовій вибірці, оцінка точності моделі за допомогою метрик.

Крок 8. Інтеграція моделі в систему моніторингу мережевого трафіку, постійний моніторинг DNS-запитів.

Крок 9. Аналіз поточних запитів і трафіку, визначення ймовірності атаки "Cache Poisoning".

Крок 10. Блокування шкідливого трафіку, оновлення налаштувань безпеки, ізоляція підозрілих запитів.

Крок 11. Завершення процесу.


Рисунок 2.9 – Блок-схема процесу навчання LSTM нейронної мережі розробленої системи

Повторювана нейронна мережа складається з кількох фіксованих функціональних одиниць активації, по одній для кожного часового кроку. Кожна одиниця має внутрішній стан, відомий як прихований стан одиниці. Цей прихований стан представляє минулі знання, які мережа зберігає на даний момент часу. Прихований стан оновлюється на кожному кроці, щоб відображати зміну знань мережі про минуле.

Формула розрахунку поточного стану:

$$h_t = f(h_{t-1}, x_t) \tag{2.1}$$

де $h_t$ - поточний стан

$h_{t-1}$ - попередній стан

$x_t$ - вхідний стан

Формула для застосування функції активації (tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2.2}$$

де $W_{hh}$ - вага рекурентного нейрона

$W_{xh}$ - вага на вхідному нейроні

Формула розрахунку виходу:

$$y_t = W_{hy}h_t \tag{2.3}$$

де $y_t$ - вихід

$W_{hy}$ - вага на вихідному шарі

У контексті даного завдання виявлення атак "отруєння кешу DNS" всі три види воріт братимуть участь у процесі опрацювання послідовностей векторів ознак, згенерованих із DNS-запитів.

Вхідні ворота визначають, яка інформація із поточного вектора ознак та попереднього прихованого стану буде занесена до стану комірки LSTM. Вони дозволяють моделі вибірково захоплювати релевантні дані, що можуть бути корисними для виявлення аномалій у майбутніх кроках послідовності.

Ворота забування регулюють, яка частина інформації із попереднього стану комірки буде збережена, а яка - відфільтрована та забута. Це допомагає запобігати

накопиченню шумових даних та дозволяє моделі адаптивно "забувати" застарілі або нерелевантні відомості при аналізі потоку DNS-запитів.

Вихідні ворота визначають, яка частина поточного стану комірки буде використана для розрахунку вихідного вектора прихованого стану та передана на наступний крок послідовності. Це дозволяє моделі LSTM виділяти найбільш важливі аспекти внутрішнього стану для прогнозування класифікації трафіку.

Усі три типи воріт тренуються одночасно під час навчання LSTM за допомогою алгоритмів оптимізації, таких як градієнтний спуск. Їх ваги налаштовуються для оптимального виділення та передачі релевантної інформації через часові кроки, що є критично важливим для виявлення атак "отруєння кешу", які можуть проявлятися у складних тимчасових закономірностях DNS-трафіку.

Було описано детальну блок-схему LSTM-архітектури з поясненням роботи воріт для розробленої системи (рис. 2.10).

Крок 1. Початок.

Крок 2. Модель отримує послідовність DNS-запитів, представлених у вигляді векторів ознак. Ці вектори можуть містити різноманітну інформацію про кожен DNS-запит, таку як IP-адреса, ім'я домену, час запиту тощо. Вхідні дані передаються до вхідних воріт LSTM, що дозволяє моделі почати процес обробки послідовності.

Крок 3. На цьому етапі вхідні ворота LSTM отримують поточний вектор ознак $x_t$ та попередній прихований стан $h_{t-1}$. Вхідні ворота обчислюють значення за формулою (1.2), де $i_t$ визначає, яка інформація з поточного вектора ознак та попереднього прихованого стану буде занесена до стану комірки LSTM. Цей процес дозволяє моделі вибірково захоплювати релевантні дані для подальшого аналізу.

Крок 4. На даному етапі здійснюється оновлення поточного стану комірки. Попередній стан комірки $C_{t-1}$ та значення з вхідних воріт $i_t$ використовуються для обчислення нового стану комірки за формулою:

$$C_t = f_t C_{t-1} + i_t C_t \quad (2.1)$$

Це оновлення забезпечує акумуляцію релевантної інформації для подальшого аналізу аномалій у DNS-трафіку.

Крок 5. Ворота забування, отримуючи поточний вектор ознак $x_t$ та попередній прихований стан $h_{t-1}$, обчислюють значення за формулою (1.1). Ці ворота регулюють, яка частина інформації з попереднього стану комірки буде збережена, а яка буде відфільтрована та забута. Це дозволяє уникнути накопичення шумових даних та забезпечує адаптивне "забування" застарілої інформації.

Крок 6. На цьому кроці модель використовує вихід з воріт забування $f_t$, вхідні ворота $i_t$ та оновлений стан комірки $C_t$ для оновлення прихованого стану за формулою:

$$h_t = o_t \tanh(C_t) \tag{2.2}$$

Це оновлення дозволяє моделі зберігати критично важливу інформацію для подальшого прогнозування.

Крок 7. Вихідні ворота, отримуючи поточний вектор ознак $x_t$ та попередній прихований стан $h_{t-1}$, обчислюють значення за формулою (1.4). Ці ворота визначають, яка частина поточного стану комірки буде використана для розрахунку виходу та передана на наступний крок послідовності.

Крок 8. На цьому кроці модель використовує поточний стан комірки $h_t$ для розрахунку вихідного вектора прихованого стану. Цей вихідний вектор передається на наступний крок послідовності, забезпечуючи контекстну інформацію для подальшого аналізу.

Крок 9. Вихідний вектор прихованого стану $h_t$ використовується для прогнозування виходу. Вихід обчислюється за формулою:

$$y_t = W_{hy} h_t \tag{2.3}$$

Цей вихід представляє ймовірність того, що поточна послідовність DNS-запитів належить до класу "атака отруєння кешу DNS".


Рисунок 2.10 – Блок-схема LSTM-архітектури з роботою воріт розробленої системи

Крок 10. На цьому етапі здійснюється аналіз вихідних даних для виявлення атак "отруєння кешу DNS". Використовуючи результати прогнозування, модель визначає наявність аномалій у DNS-трафіку, що дозволяє вчасно виявляти та реагувати на потенційні загрози.

Крок 11. На завершальному етапі процес виявлення аномалій закінчується. Модель акумулює та аналізує отримані дані, що дозволяє забезпечити високий рівень безпеки та захисту від атак "отруєння кешу DNS".

Отже, дана блок-схема демонструє послідовність процесів, що здійснюються у розробленій системі виявлення та реагування на атаки DNS Cache Poisoning з використанням LSTM (Long Short-Term Memory) мережі для аналізу та прогнозування аномалій.

## 2.3 Обґрунтування вибору алгоритмів та методів машинного навчання для ефективного виявлення аномалій

Вибір конкретного алгоритму та моделі машинного навчання для виявлення атак DNS Cache Poisoning може залежати від кількох факторів, таких як обсяг та характеристики даних, доступність ресурсів для навчання та виконання моделі, а також вимоги до точності та швидкодії. Розглянемо можливі алгоритми:

– наївний Баєсів класифікатор (Naive Bayes) - простий, але ефективний алгоритм, особливо коли даних немає дуже багато. Також добре працює з категоріальними ознаками, такими як DNS-запити та відповіді;

– дерево рішень (Decision Tree) - алгоритм добре підходить для задач класифікації на основі декількох ознак. Легко інтерпретується та може допомогти виявити важливі залежності між ознаками;

– випадковий ліс (Random Forest) - метод, який використовує кілька дерев рішень для зменшення перенавчання та покращення узагальнюючої здатності моделі.

З огляду на простоту та ефективність, було обрано алгоритм Random Forest. Випадковий ліс є потужним алгоритмом, який добре працює навіть з великими обсягами даних та високо узагальнюючими можливостями. Може обробляти як

кілька ознак, так і велику кількість даних ефективно. Має вбудовані механізми для запобігання перенавчанню, що дозволяє підтримувати високу ефективність моделі навіть на нових даних.

Використання випадкового лісу з класифікатором Random Forest є гарним варіантом для виявлення атак DNS Cache Poisoning через його ефективність, стійкість до перенавчання та можливість інтерпретації результатів.

Ліси, згенеровані з цього набору, потім навчаються за допомогою пакетної або бутстреп-агрегації. Для підвищення точності цього алгоритму також використовується ансамблевий мета-алгоритм, який називається bagging. Bagging - це комплексна модель навчання, де кількатижневі моделі навчаються на різних підмножинах навчальних даних. Кожна підмножина відбирається із заміною, і прогнозування робиться шляхом усереднення прогнозу тижневих моделей для проблеми регресії та врахування більшості голосів для проблеми класифікації. Алгоритм Random Forest визначає результати на основі прогнозів, зроблених у дереві рішень. Щоб передбачити результат, алгоритм усереднює результати з різних дерев. Зі збільшенням кількості дерев результати стають більш точними. У порівнянні з відомим алгоритмом дерева рішень, алгоритм випадкового лісу знімає обмеження використання одного дерева, зменшує реорганізацію набору даних і підвищує точність отриманих результатів.

Кожне дерево будується з використанням випадкової підмножини набору даних для вимірювання випадкової підмножини ознак у кожному розділі. Ця випадковість вводить варіативність між окремими деревами, зменшуючи ризик переобладнання та покращуючи загальну ефективність прогнозування. Під час прогнозування алгоритм агрегує результати всіх дерев шляхом голосування (для завдань класифікації) або шляхом усереднення (для завдань регресії). Цей процес спільного прийняття рішень, який підтримується кількома деревами з їхньою інформацією, надає приклад стабільних і точних результатів (рис. 2.11).

Характеристики алгоритму випадкового лісу:

– він є більш точним, ніж алгоритм дерева рішень;
– забезпечує ефективний спосіб роботи з відсутніми даними;

– може давати обґрунтовані оцінки без налаштування гіперпараметрів;

– вирішує проблему реконструкції дерева рішень;

– у кожному рандомізованому скінченному дереві вибирається випадкова підмножина ознак у точках розгалуження вузлів.

Крок 1. На першому кроці готується набір даних для тренування моделі. Цей набір містить приклади з різними характеристиками, які використовуються для навчання дерев рішень.

Крок 2. Далі, кожен підмножина даних передається для навчання окремих дерев рішень. Кожне дерево рішень навчається на різних частинах даних, що дозволяє моделі бути більш стійкою до перенавчання. На діаграмі показано три дерева рішень, але в реальних задачах їх кількість може бути значно більшою.

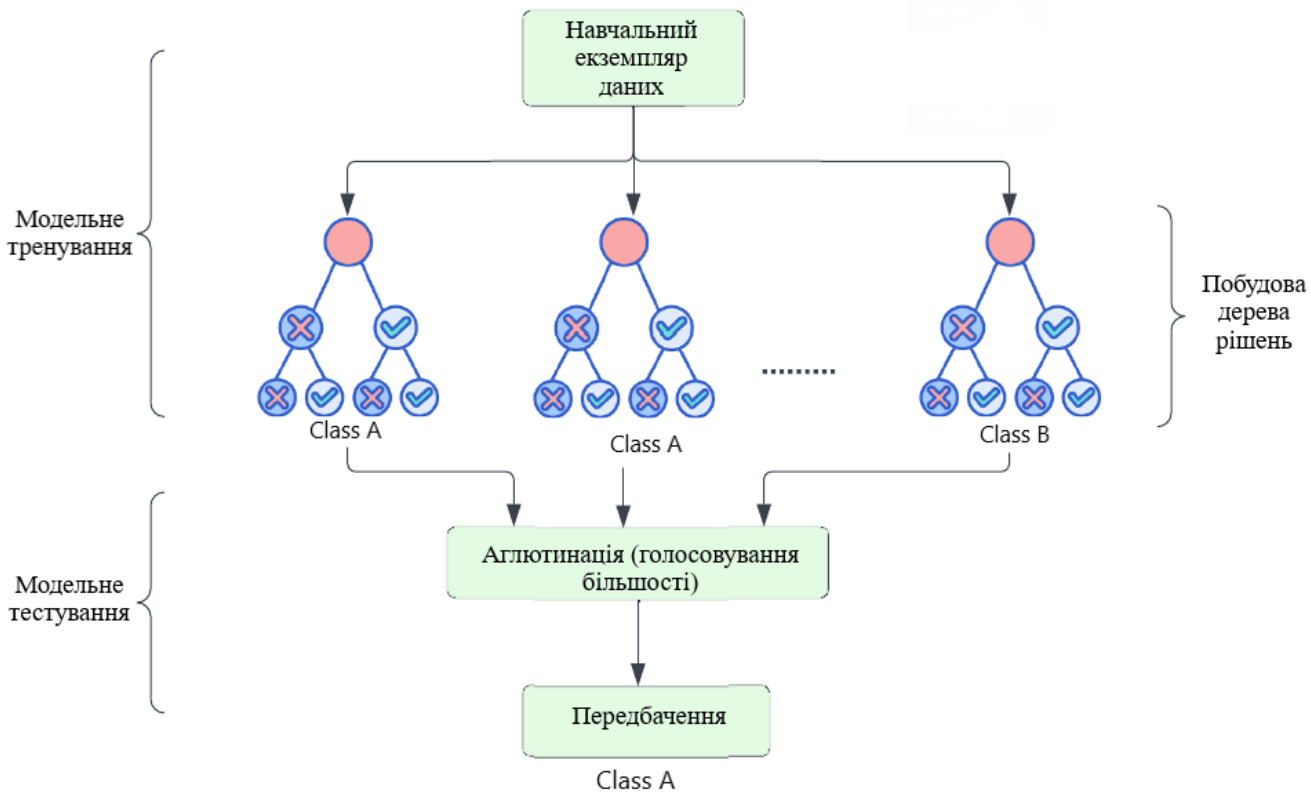



Рисунок 2.11 – Схема алгоритму випадкового лісу [24]

Крок 3. Після того, як кожне дерево рішень зробило своє передбачення, результати об'єднуються за допомогою методу аглютинації (bagging). Кожне дерево голосує за свій варіант класу, і остаточний клас визначається більшістю голосів (дерево 1 голосує за клас A, дерево 2 голосує за клас A, дерево N голосує за клас B). Більшість голосів (2 з 3) визначає, що остаточний клас для цього прикладу є клас A.

Крок 4. Остаточний результат передбачення виходить після об'єднання голосів всіх дерев. У даному випадку, модель передбачає, що вхідний приклад належить до класу A.

Загалом діаграма ілюструє процес навчання та тестування моделі машинного навчання з використанням техніки пакетування (голосування більшості) з деревами рішень. Процес можна розділити на дві основні фази: навчання моделі та тестування моделі.

Під час фази навчання моделі екземпляр навчальних даних надається як вхідні дані. Будується кілька дерев рішень, кожне з яких навчається на випадковій підмножині навчальних даних. Дерева рішень роблять прогнози для даного екземпляра, призначаючи його або до класу A, або до класу B. Цей процес повторюється для кількох дерев рішень, як вказано еліпсисом.

На етапі пакетування (більшість голосів) прогнози з усіх дерев рішень агрегуються, і клас з більшістю голосів вибирається як остаточний вихід прогнозу.

Під час фази тестування моделі нові екземпляри вводяться в навчену модель і застосовується той самий процес упаковки. Окремі дерева рішень роблять свої прогнози, і більшість голосів визначає остаточний результат прогнозу для кожного випадку.

Цей підхід, відомий як ансамблеве навчання, поєднує кілька слабких учнів (окремі дерева рішень) для створення сильного учня (мі шок моделі), який потенційно може підвищити точність і надійність прогнозування порівняно з використанням одного дерева рішень.

Дерева рішень є основними будівельними блоками алгоритму рандомізованого лісу. Дерева рішень - це методи прийняття рішень, які формують деревоподібну

структуру. Дерево рішень зазвичай має в собі три елементи: вузла рішення, листового вузла і кореневого вузла. Його алгоритм розділяє набір навчальних даних на гілки, які, в свою чергу, розгалужуються на інші гілки. Це виконується до тих пір, поки не буде досягнута листкова вершина. Листяні вузли не можуть бути далі розділені. Вузли рішень забезпечують зв'язок з листками (рис. 2.12).

Крок 1. На першому кроці відбувається вибір кореневого вузла, який є початковою точкою дерева рішень. Кореневий вузол містить основне питання або умову, яка ділить набір даних на дві або більше підгрупи.

Крок 2. Виходячи з кореневого вузла, створюються вузли рішення. Кожен вузол рішення представляє нову умову або питання, яке поділяє підгрупу даних, отриману від кореневого вузла, на ще менші підгрупи. У даній схемі є два вузли рішення на другому рівні.

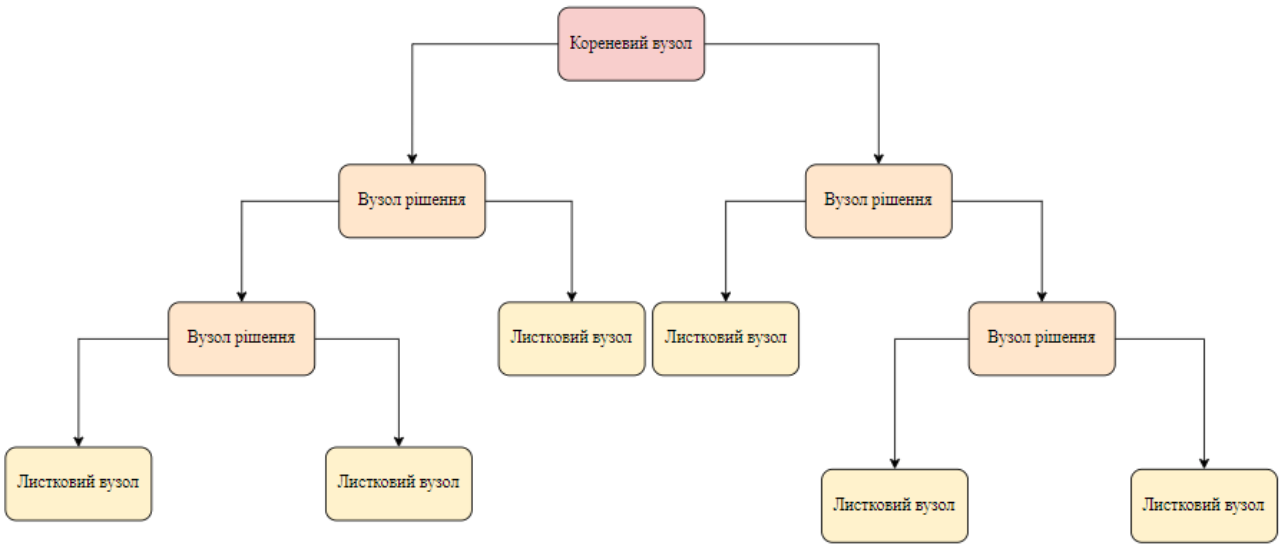



Рисунок 2.12 – Логіка алгоритму дерево рішень

Крок 3. Для лівої гілки дерева, вузол рішення ділиться на листкові вузли. Листкові вузли є кінцевими вузлами дерева і представляють остаточні рішення або

класифікації. У даній схемі лівий вузол рішення ділиться на один листковий вузол та один додатковий вузол рішення.

Крок 4. Для правої гілки дерева, вузол рішення також ділиться на два листкові вузли. Це завершує поділ даних у правій гілці на цьому рівні.

Крок 5. Додатковий вузол рішення, який знаходиться у лівій гілці, ділиться на два листкові вузли. Ці листкові вузли є кінцевими точками цього піддерева і представляють остаточні класифікації або рішення для даної підгрупи даних.

Крок 6. Додатковий вузол рішення, який знаходиться у правій гілці, ділиться на два листкові вузли. Це завершальний етап для правої гілки на другому рівні.

Результат отримуємо за допомогою ентропії. Для визначення інформаційного простору використовують ентропію та умовну ентропію цільової змінної. У цьому випадку умовна ентропія віднімається від ентропії. Збільшені знання використовуються для навчання дерева рішень. Це може зменшити певну невизначеність у дереві рішень. Високий приріст інформації означає, що висока невизначеність (інформаційна ентропія) усувається. Ентропія та інформаційний приріст є важливими для біфуркації, яка є ключовою дією при побудові дерева рішень. Випадкові ліси використовують методи групування для генерації необхідних прогнозів. Пакетне навчання використовує різні вибірки навчальних даних, а не один зразок. Дерево рішень видає різні результати в залежності від навчальних даних, переданих алгоритму Random Forest. Ці результати ранжуються, і найвищий з них обирається як кінцевий результат. Випадковий ліс розподіляє вузли відповідно до випадковості. Остаточний прогноз вибирається на основі результатів чотирьох дерев. Результат, обраний більшістю, є остаточним вибором.

Регресія - це ще одне завдання, яке виконує алгоритм випадкового лісу. У моделі випадкового лісу передаються значення характеристичних і незалежних змінних. Запуск методу випадкового лісу (рис. 2.13).

Дана схема ілюструє процес роботи класифікатора випадкового лісу, починаючи з тренування окремих дерев рішень, отримання їх прогнозів і закінчуючи об'єднанням цих прогнозів для отримання остаточного результату. Використання

методу більшості голосів дозволяє досягти високої точності та стійкості моделі до перенавчання.

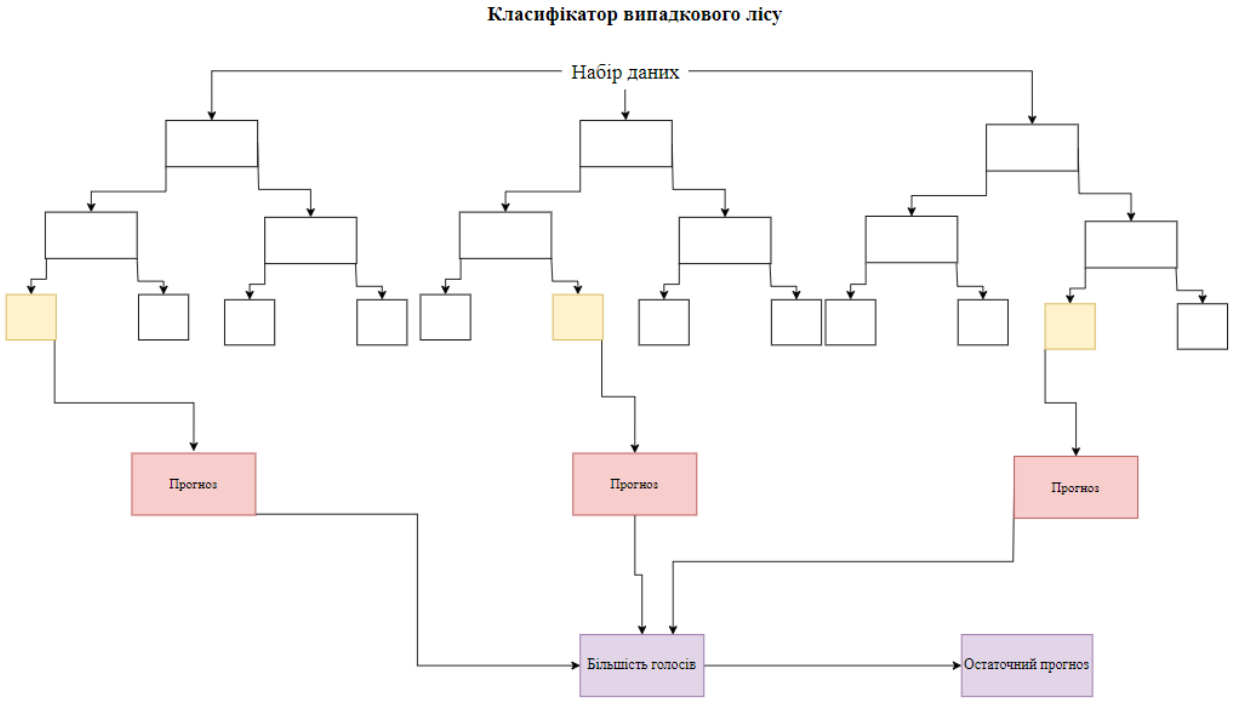



Рисунок 2.13 – Діаграма класифікації випадкового лісу

Загалом діаграма представляє архітектуру класифікатора випадкового лісу, який є методом ансамблевого навчання для завдань класифікації. Класифікатор випадкового лісу працює шляхом побудови кількох дерев рішень із набору даних і комбінування їхніх прогнозів більшістю голосів, щоб зробити остаточний прогноз.

Процес починається з введення набору даних у класифікатор випадкового лісу. Потім набір даних випадково відбирається із заміною для створення кількох підмножин, відомих як початкові зразки. Кожен початковий зразок використовується для навчання окремого дерева рішень, у результаті чого створюється набір дерев рішень, який часто називають «лісом».

Кожне дерево рішень у лісі вирощується шляхом рекурсивного поділу даних на основі найбільш інформативної функції на кожному вузлі, використовуючи підмножину випадково вибраних функцій. Цей процес триває, доки не буде

виконано критерій зупинки, наприклад максимальна глибина дерева або мінімальна кількість зразків у вузлі.

Після того, як усі дерева рішень у лісі навчені, вони роблять індивідуальні прогнози для заданого вхідного сигналу. Потім ці прогнози об'єднуються більшістю голосів, де остаточним прогнозом є клас, який отримує найбільше голосів з окремих дерев рішень.

Крок маркера більшості голосів об'єднує окремі прогнози з кожного дерева рішень, визначаючи клас, який отримав найбільшу кількість голосів. Після цього кінцевий прогноз виводиться як результат класифікатора випадкового лісу.

Даний ансамблевий підхід допомагає зменшити переобладнання та підвищити загальну точність і надійність класифікатора, оскільки окремі дерева рішень навчаються на різних підмножинах даних і різних підмножинах ознак, що фіксують різні погляди на проблему.

У регресії випадкового лісу кожне дерево дає певну оцінку. Результатом регресії є середній прогноз окремих дерев. Це зворотний процес до класифікації випадкового лісу, який дає класифікатор дерева рішень. Крім того, регресія випадкового лісу і лінійна регресія дотримуються однієї концепції, тобто вони схожі за структурою, але відрізняються за функцією.

Порівняємо обраний алгоритм машинного навчання з існуючими аналогами (табл. 2.2).

Таблиця 2.2 - Випадковий ліс з іншими алгоритмами машинного навчання [25]

| Особливість | Випадковий ліс | Інші алгоритми машинного навчання |
|---|---|---|
| Ансамбльний підхід | Використовує ансамбль дерев рішень, поєднуючи їхні результати для прогнозів, сприяючи надійності та точності. | Зазвичай покладається на одну модель (наприклад, лінійну регресію, машину опорних векторів) без підходу до ансамблю, що потенційно призводить до меншої стійкості до шуму. |

Продовження таблиці 2.2

| | | |
|---|---|---|
| Стійкість до переобладнання | Стійкість до переобладнання завдяки агрегації різноманітних дерев рішень, запобігаючи запам'ятовуванню даних навчання. | Деякі алгоритми можуть бути схильні до переобладнання, особливо при роботі зі складними наборами даних, оскільки вони можуть надмірно адаптуватися до навчального шуму. |
| Обробка відсутніх даних | Виявляє стійкість у обробці відсутніх значень, використовуючи доступні функції для прогнозів, сприяючи практичності в реальних сценаріях. | Інші алгоритми можуть вимагати імпутації або видалення відсутніх даних, що потенційно може вплинути на навчання та продуктивність моделі. |
| Змінна важливість | Забезпечує вбудований механізм для оцінки важливості змінної, допомагаючи у виборі ознак та інтерпретації впливових факторів. | Багато алгоритмів можуть не мати чіткої оцінки важливості функції, що ускладнює ідентифікацію важливих змінних для прогнозів. |
| Потенціал розпаралелювання | Використовує розпаралелювання, що дозволяє одночасно навчати дерева рішень, що призводить до швидшого обчислення для великих наборів даних. | Деякі алгоритми можуть мати обмежені можливості розпаралелювання, що потенційно може призвести до довшого часу навчання для великих наборів даних. |

Алгоритм випадкового лісу (Random Forest) є одним із найпотужніших та гнучких інструментів у сфері машинного навчання. Його вибір для розв'язання завдань класифікації та регресії обґрунтовано рядом ключових переваг, які забезпечують високу точність, надійність та ефективність моделі. Розглянемо основні причини, чому Random Forest є доцільним вибором у контексті даного дослідження.

Однією з найважливіших особливостей Random Forest є використання ансамбльного підходу, який передбачає побудову великої кількості дерев рішень та поєднання їхніх результатів для отримання кінцевого прогнозу. Такий підхід дозволяє значно підвищити точність моделі та її стійкість до шуму у даних. На

відміну від традиційних алгоритмів машинного навчання, які зазвичай покладаються на одну модель (наприклад, лінійну регресію або машину опорних векторів), Random Forest використовує колективний інтелект дерев рішень, що знижує ризик помилок і забезпечує більш надійні результати. У контексті розробки інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning), використання Random Forest сприяє підвищенню надійності та точності виявлення аномальних патернів у мережевому трафіку.

Random Forest демонструє високу стійкість до переобладнання (overfitting) завдяки агрегації різноманітних дерев рішень. Кожне дерево рішень у лісі навчається на випадковій підмножині даних, що запобігає моделі від надмірного запам'ятовування деталей навчального набору. Це особливо важливо при роботі зі складними наборами даних, де деякі алгоритми можуть надмірно адаптуватися до навчального шуму, втрачаючи здатність до узагальнення. У випадку з виявленням атак "отруєння кешу DNS", це означає, що модель зможе ефективно розрізняти нормальний трафік від шкідливого, зберігаючи свою продуктивність на різних наборах даних.

Random Forest проявляє стійкість у обробці відсутніх значень. Завдяки використанню доступних функцій для прогнозів, модель здатна ефективно працювати навіть у випадках, коли деякі дані відсутні. Це сприяє практичності алгоритму в реальних сценаріях, де відсутність даних є поширеною проблемою. Інші алгоритми часто вимагають імпутації або видалення відсутніх даних, що може негативно вплинути на процес навчання та продуктивність моделі. Для системи виявлення та реагування на атаки "отруєння кешу DNS" така стійкість до відсутніх даних є критично важливою, оскільки дані мережевого трафіку можуть бути неповними або пошкодженими.

Однією з корисних функцій Random Forest є можливість оцінки важливості змінних. Ця характеристика дозволяє дослідникам визначити, які ознаки найбільше впливають на прогнозування результату, що є надзвичайно корисним для вибору ознак та інтерпретації моделі. Багато інших алгоритмів не мають чіткої оцінки важливості функцій, що ускладнює ідентифікацію важливих змінних для прогнозів.

У випадку з виявленням атак "отруєння кешу DNS", це дозволяє ідентифікувати найважливіші параметри мережевого трафіку, які вказують на можливу атаку.

Random Forest підтримує розпаралелювання, що дозволяє одночасно навчати дерева рішень. Це значно прискорює процес обчислення для великих наборів даних. Інші алгоритми можуть мати обмежені можливості розпаралелювання, що призводить до довшого часу навчання, особливо при роботі з великими обсягами даних. Для системи виявлення та реагування на атаки "отруєння кешу DNS" це означає швидшу обробку великих обсягів мережевого трафіку, забезпечуючи своєчасне виявлення загроз.

Отже, виходячи з результатів дослідження цього методу, можна сказати, що алгоритм Random Forest є гнучким алгоритмом машинного навчання. Даний метод добре підходить для розробки інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS", Random Forest  та є оптимальним вибором, що забезпечує надійність, точність та ефективність виявлення аномальних патернів у мережевому трафіку.

### 2.4 Обґрунтування вибору засобів програмування

Внутрішня архітектура системи побудована на принципах MVC (рис. 2.14). MVC - це архітектурний патерн, який використовується при проектуванні та розробці програмного забезпечення. Патерн розділяє систему на три взаємопов'язані частини: модель даних, представлення (інтерфейс користувача) і модуль управління. MVC базується на наступних трьох елементах:

– модель - набір низькорівневих функцій для роботи з базами даних, сеансами, аутентифікацією тощо;

– представлення - файли представлення. На практиці це набір HTML-файлів;

– контролер - сполучна ланка між моделлю та представленням. Він приймає запити, обробляє їх і повертає згенеровану сторінку користувачеві.


Рисунок 2.14 – Шаблон проектування MVC [25]

Дана модель використовується щоб розділити і відокремити дані від інтерфейсу користувача (подання). Це гарантує, що зміни в інтерфейсі користувача мають мінімальний вплив на дані і що зміни в моделі даних можуть бути внесені без зміни інтерфейсу користувача.

MVP - це патерн проектування, похідний від MVC, де візуальне відображення та обробка подій відбувається різними класами, тобто представленням та сервером (рис. 2.15).

Дана діаграма представляє архітектурний шаблон Model-View-Presenter (MVP), який є різновидом архітектурного шаблону Model-View-Controller (MVC).

Діаграма складається з трьох основних компонентів:

– модель - це центральний компонент, який представляє дані та бізнес-логіку додатку. Відповідає за керування станом даних, їх валідацію та обробку. Модель не має жодних залежностей від інших компонентів і є незалежним від користувацького інтерфейсу;

– вид компонент, який відповідає за представлення даних користувачеві та забезпечує користувацький інтерфейс. Вид відображає дані, отримані від користувача, і передає події взаємодії;


Рисунок 2.15 – Шаблон проектування MVP [26]

– користувач є проміжним шаром між видом та моделлю. Отримує дані від моделі, підготовлює їх для представлення у виді та реагує на події, згенеровані видом. Користувач також відповідає за оновлення View у відповідь на зміни в моделі.

Стрілки на діаграмі вказують на напрямок взаємодії між компонентами. Вид надсилає запити на оновлення даних і події взаємодії користувача до корисутвача. Користувач, зі свого боку, отримує дані від моделі та передає їх до виду для відображення. Модель не має прямого зв'язку з видом, що забезпечує розподіл відповідальностей та полегшує тестування.

Шаблон MVP відокремлює подання даних від їх логіки, що сприяє модульності, підвищенню можливості повторного використання коду та полегшує тестування окремих компонентів.

Було обрано MySQL через її продуктивність та інші параметри, перелічені нижче.

MySQL - це безкоштовна система управління реляційними базами даних. Ця система управління базами даних (СУБД) з відкритим вихідним кодом [27].

MySQL дозволяє здійснювати доступ та маніпулювання даними, що зберігаються в базі даних, захищає дані від пошкодження та порушення цілісності,

а також надає метадані, необхідні для опису даних Основна відмінність між СУБД та СКБД полягає в тому, що остання має справу лише з реляційними базами даних і підтримує зберігання даних у вигляді структур, таких як таблиці або зв'язки між таблицями.

В даний час MySQL є однією з найбільш широко використовуваних систем управління базами даних. MySQL підтримує різні мови програмування і в основному використовується для створення динамічних веб-сторінок. MySQL підтримує безпечне середовище для зберігання, управління та пошуку даних з багатою базою даних. MySQL надає багатий набір функцій, які підтримують безпечне середовище для зберігання, управління та пошуку даних.

Вона характеризується швидкістю, гнучкістю та простотою використання і була розроблена для підвищення продуктивності транзакцій у великих базах даних.

Вихідний код сервера компілюється на багатьох платформах. Можливості сервера найкраще реалізуються в UNIX-системах, де багатоканальна підтримка покращує загальну продуктивність системи (рис. 2.16). Переваги сервера MySQL:

– його легко встановити та використовувати;

– необмежена кількість користувачів може одночасно працювати з базою даних;

– таблиці можуть містити до 50 мільйонів рядків;

– висока швидкість виконання команд;

– проста та ефективна система безпеки.

MySQL може підтримувати роботу баз даних значного розміру; згідно з документацією, що постачається з MySQL, деякі бази даних, що використовуються MySQL AB (розробником MySQL), можуть зберігати до 50 мільйонів записів [29].

Підключення MySQL має мережеву структуру. Кілька користувачів можуть отримати доступ до MySQL одночасно з будь-якої точки Інтернету. MySQL має ряд інтерфейсів прикладного програмування.

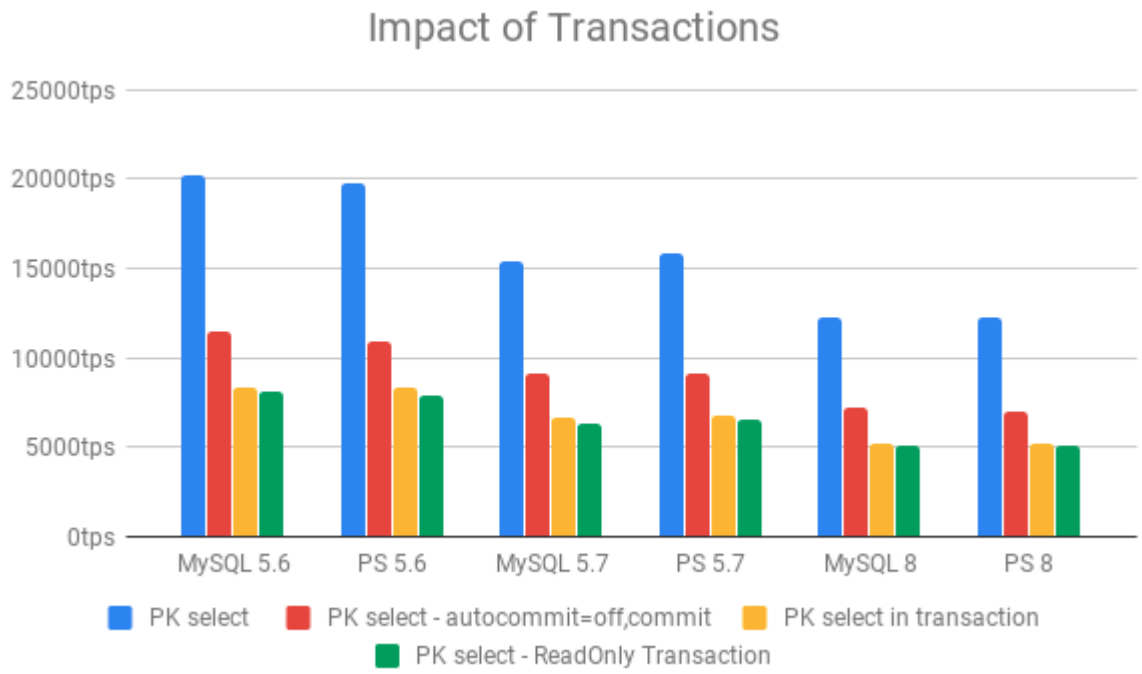



Рисунок 2.16 – Швидкодія MySQL [28]

MySQL має систему контролю доступу до даних, яка шифрує дані під час передачі. Дуже високий рівень безпеки забезпечує база даних mysql, яка створюється при встановленні пакета і містить п'ять таблиць. За допомогою цих таблиць можна визначити, які користувачі з якого домену можуть працювати над якими таблицями і які команди можуть використовувати. Завдяки відкритому вихідному коду до пакета можна додавати необхідний функціонал і розширювати функціональність за бажанням.

Крім того, наразі існують версії програми для більшості поширених комп'ютерних платформ. Це означає, що вам не обов'язково використовувати певну операційну систему. Ви можете вибрати, з якою операційною системою ви хочете працювати - наприклад, Linux або Windows - але якщо ви зміните операційну систему, ви не втратите жодних даних і вам не знадобляться додаткові інструменти для перенесення даних.

Розглянемо популярні мови програмування для розробки [30]:

– Java - потужна та масштабована мова програмування, яка часто використовується для створення корпоративних веб-додатків та серверних частин веб-сайтів. Java відома своєю кросплатформеністю та безпекою;

– Python - мова програмування з простим та зрозумілим синтаксисом, яка широко використовується для веб-розробки, наукових обчислень, машинного навчання та автоматизації. Популярні веб-фреймворки на Python включають Django та Flask;

– Ruby - динамічна мова програмування з акцентом на простоту та продуктивність. Популярний веб-фреймворк Ruby on Rails часто використовується для швидкої розробки веб-додатків;

– C# - мова програмування від Microsoft, яка часто використовується для розробки веб-додатків на платформі .NET, а також для створення ігор та мобільних додатків;

– JavaScript був призначений для клієнтської сторони веб-сторінок, з появою Node.js його також широко використовують для розробки серверної частини веб-додатків.

Порівняємо мови програмування для обґрунтування вибору PHP у таблиці 2.3.

Таблиця 2.3 – Порівняльна характеристика мов програмування [31]

| Мова | Швидкість | Безпека | Веб-інтеграція | Портативність | Підтримка бази даних |
|---|---|---|---|---|---|
| PHP | Висока | Хороша | Відмінна | Висока | Відмінна (MySQL) |
| Java | Помірна | Відмінна | Хороша | Висока | Хороша |
| Python | Помірна | Хороша | Хороша | Висока | Хороша |
| Ruby | Помірна | Хороша | Хороша | Висока | Хороша |
| C# | Висока | Відмінна | Хороша | Низька | Хороша |
| JavaScript | Висока | Відмінна | Відмінна | Висока | Помірна |

При виборі мови програмування ключовим параметром залишалася ефективність. Це дуже важливий фактор при програмуванні в багатокористувацькому середовищі, в тому числі в Інтернеті. Дуже важливою перевагою PHP є його "двигун": PHP не має ні компілятора, ні інтерпретатора. Мова

є широкомовним інтерпретатором, і цей апарат двигуна PHP дозволяє йому обробляти скрипти на дуже високих швидкостях.

За деякими оцінками, більшість PHP-скриптів (особливо невеликого розміру) обробляються швидше, ніж аналогічні програми, написані на Perl. PHP також надає розробникам і адміністраторам гнучкі та ефективні інструменти безпеки. Вони поділяються на дві категорії: інструменти системного рівня та інструменти рівня додатків.

PHP реалізує механізми безпеки, які під управлінням адміністратора і при правильному налаштуванні забезпечують максимальну свободу і безпеку. Безпечний режим дозволяє користувачам накладати ряд важливих обмежень при використанні PHP. Наприклад, можна обмежити максимальний час виконання і використання пам'яті (неконтрольоване споживання пам'яті негативно впливає на продуктивність сервера). Як і у випадку з cgi-bin, адміністратори можуть встановлювати обмеження на директорії, в яких користувачі можуть переглядати і виконувати PHP-скрипти, а також можуть використовувати PHP-скрипти для перегляду конфіденційної інформації на сервері [31].

До стандартного набору функцій PHP включено багато надійних механізмів шифрування. PHP також сумісний з багатьма сторонніми програмами і легко інтегрується з безпечними технологіями електронної комерції. Ще однією перевагою є те, що вихідний текст PHP-скриптів можна відображати в браузері. Реалізація ГПС на стороні сервера запобігає нетривіальному перехопленню скриптів користувачами, які володіють достатніми знаннями для виконання команди View Source. Як вбудована мова, PHP є дуже гнучкою відповідно до потреб розробників. PHP часто рекомендують використовувати в поєднанні з HTML, але його також можна інтегрувати з JavaScript, WML, XML та іншими мовами. Крім того, добре структурований PHP-додаток можна легко розширювати за потреби (але це справедливо для всіх основних мов програмування).

Оскільки PHP-скрипти повністю компілюються на стороні сервера перед відправкою клієнту, немає проблеми залежності від браузера. Фактично, скрипти ГВЧ можна надсилати на будь-який пристрій з браузером, включаючи традиційні

ПК, а також мобільні телефони та електронні ноутбуки. Програмісти, які працюють з утилітами, можуть запускати PHP в режимі командного рядка.

PHP не містить специфічного для веб-сервера коду, тому користувачі не обмежені певним сервером (який може бути для них чужим): Apache, Microsoft IIS, Netscape Enterprise Server, Stronghold Zeus - PHP працює на всіх цих серверах. Оскільки ці сервери працюють на різних платформах, PHP, як правило, є портативною і незалежною мовою.

Також використано DnsClass, який дозволяє імітувати реальні DNS-запити та тестувати реакцію системи на аномалії та атаки, зокрема DNS Cache Poisoning. Цей клас створений для забезпечення зручного середовища для тестування безпеки та надійності системи DNS.

Основні можливості класу DnsClass включають:

– симуляцію DNS-запитів;
– тестування вразливостей;
– перевірку підписів DNSSEC.

Дозволяє перевіряти правильність підписів DNSSEC та визначати наявність або відсутність захисту DNSSEC для кожного запиту. За допомогою класу DnsClass можна проводити тести на вразливості системи, зокрема на атаки DNS Cache Poisoning, та перевіряти, як система реагує на аномальні ситуації.

Клас дозволяє імітувати реальні DNS-запити, що дозволяє тестувати систему на навантаження та перевіряти її реакцію на цей трафік.

Система забезпечує зручний та надійний інструментарій для тестування безпеки та надійності DNS, дозволяючи виявляти та виправляти потенційні вразливості перед їх використанням у реальних умовах експлуатації.

### 2.5 Висновки до розділу

У цьому розділі було проведено розробку архітектури та реалізацію інтелектуальної системи для виявлення аномальних патернів в DNS-трафіку. Починаючи з розробки моделі машинного навчання для виявлення атак DNS Cache Poisoning на основі нейронної мережі RNN, було обрано та навчено модель на

основі алгоритму Random Forest, що показав високу ефективність у виявленні таких атак. Інтеграція розробленої системи з існуючими заходами безпеки DNS була успішною, що дозволить покращити рівень захисту мережевої інфраструктури від атак DNS Cache Poisoning.

Обґрунтування вибору алгоритмів та методів машинного навчання базувалося на їхній ефективності та потенціалі виявлення аномалій у DNS-трафіку. Алгоритм Random Forest був обраний через його здатність до роботи з великими обсягами даних та високу ефективність навчання, що забезпечує надійне виявлення атак.

Обгрунтування вибору засобів програмування полягало в їхній зручності та доступності для розробки та інтеграції системи. Використання мови програмування PHP разом з бібліотекою PHP-ML дозволило забезпечити ефективну і швидку розробку системи, яка відповідає вимогам безпеки та функціональності.

Результати роботи в цьому розділі вказують на успішну реалізацію інтелектуальної системи для виявлення аномальних патернів в DNS-трафіку, яка може бути використана для підвищення безпеки мережевих систем та захисту від атак DNS Cache Poisoning.

# 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ДЛЯ ВИЯВЛЕННЯ АТАК DNS CACHE POISONING

В даному розділі опишемо основні етапи практичної реалізації та тестування розробленої системи для виявлення атак DNS Cache Poisoning. Спроектовано та описано розробку графічного користувацького інтерфейсу, програмну реалізацію розробленої системи, наведено інструкцію користувача для роботи з додатком, проведено тестування розробленого програмного продукту.

## 3.1 Розробка графічного інтерфейсу програмної розробки для інтелектуальної системи

В сучасному цифровому світі, де мережева безпека стає все більшою проблемою, ефективний та зручний інтерфейс відіграє критичну роль у забезпеченні надійності та ефективності захисту мережі. Інтерфейс програмної розробки для інтелектуальної системи, спрямованої на виявлення аномальних патернів в DNS-трафіку, не є винятком.

Було почато розробку інтуїтивно зрозумілого та функціонального графічного інтерфейсу, який дозволить адміністраторам мережевих систем ефективно взаємодіяти з інтелектуальною системою захисту від атак "отруєння кешу DNS". Цей інтерфейс буде наділено широким спектром можливостей, від відстеження DNS-трафіку до управління записами DNSSEC, і він буде побудований з урахуванням кращих практик розробки програмного забезпечення та дизайну інтерфейсів.

Під час розробки інтерфейсу дотримуємося принципів зручності, ефективності та безпеки. Кожен елемент інтерфейсу проектується з урахуванням потреб користувача та має на меті спростити робочий процес адміністрування мережі та виявлення потенційних загроз безпеці. Наша мета - створити інтерфейс, який не лише виконує свої основні функції, але й забезпечує задоволення від користування та впевненість у його надійності.

Створення зручного та інтуїтивного інтерфейсу вимагає дотримання кількох ключових правил. По-перше, основою вдалого дизайну є "невидимість" - інтерфейс має бути настільки природним та зрозумілим, що користувачі можуть переміщатися ним без жодних перешкод чи плутанини.

По-друге, контент повинен бути відформатованим відповідно до екрану пристрою, на якому він відображається, уникаючи необхідності масштабування чи горизонтальної прокрутки для перегляду всього вмісту.

По-третє, важливо забезпечити достатній контрастний контент, такий як текст, на тлі, щоб покращити його читабельність.

Нарешті, інтерфейс має бути адаптивним, легко читатися та керуватися незалежно від розміру екрану чи орієнтації пристрою, на якому він використовується користувачами.

Розглянемо проектування деяких вікон додатку для розуміння його структури.

Сторінка авторизації адміністратора являє собою форму з полями для введення облікових даних користувача та перевірки доступу до його акаунту. По центру вікна буде розміщена кнопка “Увійти”, окрім цього передбачений надпис “Дані невірні”, “Заповніть дане поле” (рис. 3.1).

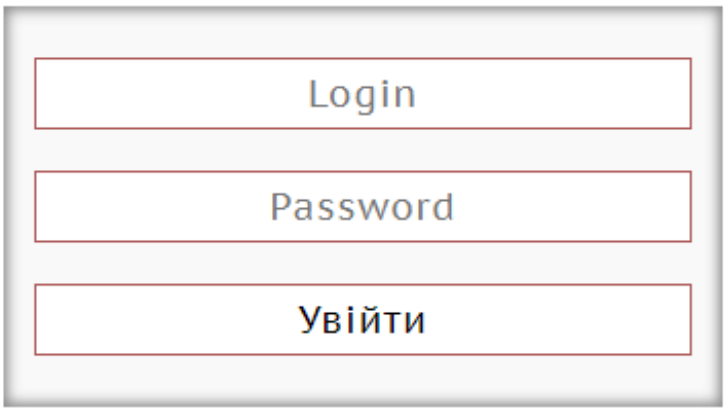



Рисунок 3.1 – Сторінка авторизації адміністратора

Наступне вікно - це головне вікно додатку, що має доступ до всього функціоналу системи, а саме “DNS-трафік”, “Записи DNSSEC”, “Статистика поведінки”, “Прийнятні обмеження”, “Повідомлення”, “Профіль”, “Вихід” (рис. 3.2).


| Домен | Тип | Дата | |
|---|---|---|---|
| ransomwarehub.net | SRV | 2024-05-03 17:34:49 | |
| slack.com | PTR | 2024-05-03 16:53:26 | |
| zoom.us | SRV | 2024-05-03 16:53:26 | |
| malware.net | TXT | 2024-05-03 16:53:06 | |
| ransomwarehub.net | PTR | 2024-05-03 16:53:06 | |
| malware.net | TXT | 2024-05-03 16:53:06 | |
| wordpress.com | AAAA | 2024-05-03 16:52:06 | |
| yahoo.com | CNAME | 2024-05-03 16:52:06 | |
| github.com | TXT | 2024-05-03 16:51:47 | |
| stackoverflow.com | A | 2024-05-03 16:51:47 | |
| apple.com | MINFO | 2024-05-03 16:51:26 | |
| microsoft.com | MX | 2024-05-03 16:51:26 | |
| amazon.com | SRV | 2024-05-03 16:51:07 | |
| ebay.com | HINFO | 2024-05-03 16:51:07 | |
| instagram.com | SOA | 2024-05-03 16:50:45 | |
| reddit.com | PTR | 2024-05-03 16:50:45 | |
| linkedin.com | CNAME | 2024-05-03 16:50:20 | |
| wikipedia.org | NS | 2024-05-03 16:50:20 | |
| twitter.com | TXT | 2024-05-03 16:49:27 | |
| youtube.com | AAAA | 2024-05-03 16:49:27 | |
| google.com | A | 2024-05-03 16:49:25 | |
| facebook.com | MX | 2024-05-03 16:49:25 | |

Рисунок 3.2 – Головне вікно системи DNS-трафіку

Основна частина сторінки має графік по центру графік розподілу DNS-трафіку. Нижче графіка наведена таблиця з записами DNS запитів, включаючи “Домен”, “Тип”, “Дата”. Червоними прапорцями помічено аномальні патерни небезпечних запитів, тобто їх потенційну небезпеку. Загалом, таблиця надає детальну інформацію про конкретні запити DNS, а діаграма забезпечує загальний огляд розподілу небезпечного трафіку.

Отже, ключовими вимогами до розробленого інтерфейсу системи аналізу DNS-трафіку є інтуїтивна зрозумілість, адаптивність до різних пристроїв та розмірів екрану, чіткий та лаконічний дизайн без зайвих елементів. Ретельне проектування та розробка графічного інтерфейсу на початкових етапах спрощують процес та дозволяють врахувати всі необхідні деталі та функціональні вимоги до системи.

Чітка візуалізація статистики небезпечного трафіку у вигляді діаграми, таблиці з детальною інформацією про окремі запити DNS та можливість швидко ідентифікувати підозрілі домени за допомогою відповідних позначок є ключовими особливостями інтерфейсу, що забезпечують ефективний аналіз та реагування на потенційні кіберзагрози.

## 3.2 Програмна реалізація системи для виявлення та реагування на атаки DNS Cache Poisoning з використанням машинного навчання

Програмна реалізація є одним із ключових етапів розробки системи. На цьому етапі, на основі спроектованих алгоритмів і з урахуванням особливостей обраних засобів створюється програмний код. У цьому розділі ми покроково розглянемо деякі основні моменти реалізованого коду, які забезпечують запланований функціонал додатку.

Представлений код у контролері WelcomeController з простору імен controllers, реалізує кілька ключових функцій для управління обліковими записами користувачів, таких як аутентифікація, редагування профілю та вихід із системи. Метод loginAction обробляє запит на вхід користувача до системи, перевіряючи наявність логіну і пароля в даних POST-запиту, очищаючи введені дані, здійснюючи пошук користувача в базі даних за логіном, а також перевіряючи відповідність

введених даних збереженим у базі. У разі успішної аутентифікації користувачеві присвоюється ідентифікатор сесії, і він перенаправляється до адміністративної частини сайту, тоді як у разі невірних даних відбувається повернення на головну сторінку із повідомленням про помилку:

```
<?php
namespace controllers;
class WelcomeController extends Controller {
  public function indexAction(){
            //
  }
   public function loginAction(){
            if( empty($_POST['login']) || empty($_POST['pass']) ){
                     redirect('/');
            }
            $login = clear($_POST['login'], 'text');
            $user  = \R::findOne( 'users', 'login = ?', [$login] );
            if( $login == $user['login'] && md5($_POST['pass']) == $user['pass'] ){
                     $_SESSION['id'] = $user['id'];
                     $_SESSION['login'] = $user['login'];
                     redirect('/admin');
            }else{
                     $_SESSION['error'] = 'Дані невірні!';
                     redirect('/');
            }
  }
```

Метод editAction обробляє запит на редагування профілю користувача, здійснюючи очищення введених даних, пошук користувача в базі даних за ідентифікатором сесії, оновлення логіну та пароля (за потреби), збереження змін і перенаправлення користувача із повідомленням про успішне оновлення даних. Метод logoutAction реалізує вихід користувача із системи, видаляючи ідентифікатор користувача і логін із сесії та перенаправляючи його на головну сторінку. Цей контролер використовує сесії для зберігання стану користувача та бібліотеку \R::findOne для взаємодії з базою даних, забезпечуючи основні функції для роботи з обліковими записами:

```
   public function editAction(){
            $login = clear($_POST['name'], 'text');
            $pass = clear($_POST['pass'], 'text');
            $user  = \R::findOne( 'users', 'id = ?', [$_SESSION['id']] );
            $user->login = $login;
            if( $pass ){
                     $user->pass = md5($_POST['pass']);
            }
            \R::store( $user );
```

```
            $_SESSION['error'] = 'Дані оновлено';
            redirect();
    }
     public function logoutAction(){
            unset( $_SESSION['id'] );
            unset( $_SESSION['name'] );
            redirect('/');
     }
}
```

Розробка головної сторінки. Меню навігації, представлено блоком з класом menu, містить кілька посилань, які ведуть до різних підрозділів адміністративної панелі, зокрема “DNS-трафік”, “Записи DNSSEC”, “Статистика поведінки”, “Прийнятні обмеження”, “Повідомлення”, “Профіль”, “Вихід”:

```
<div class="wrap">
    <div class="menu">
            <a href="/admin/dnsTraf">DNS-трафік</a>
            <a href="/admin/secTraf">Записи DNSSEC</a>
            <a href="/admin/stat">Статистика поведінки</a>
            <a href="/admin/limit">Прийняті обмеження</a>
            <a href="/admin/message">Повідомлення</a>
            <a href="/admin/profile">Профіль</a>
            <a href="/welcome/logout">Вихід</a>
    </div>
```

Лівий підрозділ містить заголовок "Навчання" і кілька посилань для роботи з DNS та DNSSEC даними, зокрема оновлення, додавання нових записів та перегляд історичних даних:

```
    <div class="main">
            <div class="left">
                    <span class="big">Навчання</span>
                    <a href="/neuron/">Оновити</a>
                    <a href="/neuron/loadDns">Додати DNS</a>
                    <a href="/neuron/loadDnsHistory">Дані DNS</a>
                    <a href="/neuron/loadSec">Додати DNSSEC</a>
                    <a href="/neuron/loadSecHistory">Дані DNSSEC</a>
            </div>
```

Правий підрозділ фокусується на "DNS-трафіку". Він містить контейнер із ідентифікатором drow для відображення центрального контенту та динамічно формується список DNS-записів. Якщо змінна $dns не порожня, відображається список записів, кожен з яких містить доменне ім'я, тип, дату та іконку попередження (якщо запис містить попередження):

```
<div class="right">
```

```
<span class="big">DNS-трафік</span>
        <div class="center" id="drow"></div>
        <?php if ( !empty($dns) ): ?>
                <div class="data-list">
                        <div class="item">
                                <span class="part">Домен</span>
                                <span class="part">Тип</span>
                                <span class="part">Дата</span>
                                <span class="small"></span>
                                </div>
```

Даний код описує контролер AdminController в PHP, який відповідає за виконання різних дій в адміністративній частині системи. Метод dnsTrafAction() отримує всі записи DNS-трафіку з бази даних, сортує їх за датою в порядку спадання і створює асоціативний масив з підрахунком кількості попереджень (warning) для кожного домену. Після цього масив перетворюється у формат JSON і передається у вигляді змінної в шаблон:

```
class AdminController extends Controller
    public function dnsTrafAction(){
        $dns = \R::findAll( 'dns_traf', 'ORDER BY date DESC' );
        $arr = [];
        foreach($dns as $item){
            if(  $item->warning == 1 ){
                if( empty($arr['DNS'][$item['domain']]) ){
                    $arr['DNS'][$item['domain']] = 1;
                }else{
                    $arr['DNS'][$item['domain']] += 1;
                }
            }
        }
        $json = json_encode($arr);
        $this->set(compact('dns', 'json'));
  }
```

Метод secTrafAction() аналогічний dnsTrafAction, але працює з даними DNSSEC-трафіку. Він також створює асоціативний масив з підрахунком попереджень для кожного домену і передає його у форматі JSON в шаблон:

```
    public function secTrafAction(){
        $sec = \R::findAll( 'sec_traf', 'ORDER BY date DESC' );
        $arr = [];
        foreach($sec as $item){
            if(  $item->warning == 1 ){
                if( empty($arr['DNSSEC'][$item['domain']]) ){
                    $arr['DNSSEC'][$item['domain']] = 1;
                }else{
                    $arr['DNSSEC'][$item['domain']] += 1;
                }
            }
```

```
        }
        $json = json_encode($arr);
        $this->set(compact('sec', 'json'));
    }
```

Метод profileAction() отримує дані про поточного користувача з бази даних за його ID, який зберігається в сесії, і передає ці дані в шаблон. Метод statAction() отримує всі записи DNS-трафіку і DNSSEC-трафіку, які містять попередження, і об'єднує їх у один масив, який потім передається в шаблон. Метод limitAction() отримує всі записи з таблиці alert (сповіщення) з бази даних, сортує їх за датою в порядку спадання і передає їх у шаблон:

```
public function profileAction(){
        $user = \R::findOne( 'users', 'id = ? ', [ $_SESSION['id'] ] );
        $this->set(compact('user'));
    }
     public function statAction(){
        $dns = \R::find( 'dns_traf', 'WHERE warning = 1 ORDER BY date DESC' );
        $sec = \R::find( 'sec_traf', 'WHERE warning = 1 ORDER BY date DESC' );
        $res = array_merge($dns, $sec);
        $this->set(compact('res'));
    }
     public function limitAction(){
        $alert = \R::find( 'alert', 'ORDER BY date DESC' );
        $this->set(compact('alert'));
    }
```

Метод alertRemoveAction() видаляє конкретне сповіщення з таблиці alert за його ID, який передається через GET-запит, і перенаправляє користувача назад на попередню сторінку. Метод messageAction() отримує всі повідомлення з таблиці message і передає їх у шаблон. Метод addEmailAction() створює новий запис у таблиці message з електронною поштою, яку отримує з POST-запиту, і зберігає цей запис у базі даних, після чого перенаправляє користувача назад на попередню сторінку. Метод emailRemoveAction() видаляє конкретне повідомлення з таблиці message за його ID, який передається через GET-запит, і перенаправляє користувача назад на попередню сторінку:

```
     public function alertRemoveAction(){
        $alert = \R::findOne( 'alert', 'id = ?', [$_GET['id']] );
        \R::trash($alert);
        redirect();
    }
     public function messageAction(){
        $message = \R::findAll( 'message' );
```

```
            $this->set(compact('message'));
        }
         public function addEmailAction(){
            $message = \R::dispense( 'message' );
            $message->email = $_POST['email'];
            \R::store( $message );
            redirect();
        }

         public function emailRemoveAction(){
            $message = \R::findOne( 'message', 'id = ?', [$_GET['id']] );
            \R::trash($message);
            redirect();
        }
    }
```

Конструктор класу ініціалізує з'єднання з базою даних та зберігає інформацію про поточний маршрут, встановлюючи також назву представлення. Метод getView() створює об'єкт представлення і викликає його метод render(), передаючи змінні до представлення. Метод set($vars) встановлює змінні, які будуть передані до представлення. Додатково, клас має методи для перевірки, чи був запит здійснений через AJAX, завантаження конкретного представлення та обробки DNS-трафіку:

```
    public function __construct($route) {
            Model::instance();
            $this->route = $route;
            $this->view = $route['action'];
        }
        public function getView(){
            $vObj = new View($this->route, $this->layout, $this->view);
            $vObj->render($this->vars);
        }
        public function set($vars){
            $this->vars = $vars;
        }
        public function isAjax() {
            return isset($_SERVER['HTTP_X_REQUESTED_WITH']) && $_SERVER['HTTP_X_REQUESTED_WITH'] === 'XMLHttpRequest';
        }
         public function loadView($view, $vars = []){
            extract($vars);
            require ROOT . "/views/{$this->route['prefix']}{$this->route['controller']}/{$view}.php";
        }
```

Наприклад, метод dnsView() створює сокет для прослуховування DNS-трафіку та виводить інформацію про отримані пакети:

```
    public function dnsView($view, $vars = []){
            $socket = socket_create(AF_INET, SOCK_RAW, IPPROTO_UDP);
            if ($socket === false) {
                    echo "Не вдалося створити сокет: " . socket_strerror(socket_last_error()) . "\n";
```

```
                exit;
        }
        if (!socket_bind($socket, '0.0.0.0', 53)) {
                echo "Не вдалося пов'язати сокет: " . socket_strerror(socket_last_error()) . "\n";
                exit;
        }
        echo "Сокет успішно створено та призначено.\n";
        while (true) {
                if (socket_recvfrom($socket, $buffer, 2048, 0, $source_ip, $source_port)) {
                        echo "Отримано пакет від " . $source_ip . ":" . $source_port . "\n";
                        echo "Дані: " . bin2hex($buffer) . "\n";
                } else {
                        echo "Помилка читання даних з сокета: " . socket_strerror(socket_last_error()) . "\n";
                }
        }
        socket_close($socket);
}
```

Метод secView() виконує запит до API DNSViz для перевірки статусу DNSSEC для вказаного домену і виводить інформацію про наявність або відсутність DNSSEC, а також отримує DNS-записи для домену та виводить їх:

```
public function dnsView($view, $vars = []){
    public function secView($domain){
        $url = "https://dnsviz.net/api/v1/".urlencode($domain)."/json";
        $response = file_get_contents($url);
        $data = json_decode($response, true);
        if(isset($data['status']['secure'])){
            if($data['status']['secure'] == true){
                echo "DNSSEC встановлено для домена ".$domain;
            } else {
                echo "DNSSEC не встановлено для домена ".$domain;
            }
        } else {
            echo "Не вдалося отримати дані DNSSEC для домена ".$domain;
        }
        $dns_records = dns_get_record($domain, DNS_ALL);
        echo "DNS-записи для домена ".$domain.":\n";
        $domain = "example.com"; // ваш домен для перевірки
    }
```

Представлений контролер NeuronController, який відповідає за обробку різних дій пов'язаних з нейронною мережею. Кожен метод цього контролера відповідає конкретній дії, яку можна виконати в системі. Метод addDnsAction() обробляє POST-запит, отримує дані JSON і тип, додає їх до бази даних за допомогою ORM-бібліотеки RedBeanPHP, а потім перенаправляє користувача на сторінку з нейроном.

Методи loadDnsHistoryAction(), loadSecAction(), addSecAction() і loadSecHistoryAction() здійснюють аналогічні дії з історією та обробкою DNS та

DNSSEC даних, які використовуються для моніторингу та аналізу мережевого трафіку в системі:

```
public function addDnsAction(){
        $json = $_POST['json'];
        $type = $_POST['type'];
        $dns = \R::dispense( 'dns_ii' );
        $dns->json = $json;
        $dns->type = $type;
        \R::store( $dns );
        $_SESSION['error'] = 'Дані створено';
        redirect('/neuron/');
  }
   public function loadDnsHistoryAction(){
        $dns = \R::findAll( 'dns_ii', 'ORDER BY date DESC' );
        $this->set(compact('dns'));
  }
   public function addSecAction(){
        $json = $_POST['json'];
        $type = $_POST['type'];
        $sec = \R::dispense( 'sec_ii' );
        $sec->json = $json;
        $sec->type = $type;
        \R::store( $sec );
        $_SESSION['error'] = 'Дані створено';
        redirect('/neuron/');
  }
   public function loadSecHistoryAction(){
        $sec = \R::findAll( 'sec_ii', 'ORDER BY date DESC' );
        $this->set(compact('sec'));
   }
```

Використання випадкового лісу з класифікатором Random Forest є гарним варіантом для виявлення атак DNS Cache Poisoning через його ефективність, стійкість до перенавчання та можливість інтерпретації результатів:

```
require_once 'vendor/autoload.php';
use Phpml\Dataset\ArrayDataset;
use Phpml\Classification\RandomForest;
use Phpml\FeatureExtraction\TokenCountVectorizer;
use Phpml\CrossValidation\StratifiedRandomSplit;
$dataset = new ArrayDataset($samples, $labels);
$split = new StratifiedRandomSplit($dataset, 0.3);
$randomForest = new RandomForest();
$randomForest->train($split->getTrainSamples(), $split->getTrainLabels());
$predictedLabels = $randomForest->predict($split->getTestSamples());
foreach ($predictedLabels as $index => $predictedLabel) {
   echo 'Predicted: ' . $predictedLabel . ', Actual: ' . $split->getTestLabels()[$index] . PHP_EOL;
}
```

Після навчання моделі ми можемо зберегти її в файл, щоб потім використовувати навчену модель без необхідності повторного навчання:

```
// Ініціалізація та навчання моделі Random Forest
$model = new RandomForest();
// Навчаємо модель...
// Збереження навченої моделі в файл
$modelManager = new ModelManager();
$modelManager->saveToFile($model, 'random_forest_model.phpml');
```

Цей код створить файл random_forest_model.phpml, в якому буде збережена навчена модель Random Forest. Після збереження можна використовувати цю модель для передбачення на нових даних без необхідності повторного навчання.

Інтеграція системи з існуючими заходами безпеки DNS є важливим кроком для забезпечення повноцінного захисту інфраструктури від атак DNS Cache Poisoning.

Спочатку необхідно реалізувати підключення до існуючого сервісу DNS. Потрібно взаємодіяти з DNS-серверами для перевірки та фільтрації запитів DNS на основі результатів вашої системи виявлення атак:

```
$dnsServerAddress = '127.0.0.1';
$dnsServerPort = 53;
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
socket_connect($socket, $dnsServerAddress, $dnsServerPort);
```

Далі отримуємо та обробляємо DNS-запити, що надходять на сервер. Перевіряється кожен запит на наявність ознак атаки DNS Cache Poisoning за допомогою навченої моделі:

```
while (true) {
    socket_recvfrom($socket, $request, 1024, 0, $clientAddress, $clientPort);
    // Отримання DNS-запиту та аналіз.
    // Використання моделі для визначення, чи є цей запит атакою DNS Cache Poisoning
    $isAttack = $model->predict($request);
    // Обробка результатів та відповідь DNS-сервера
    if ($isAttack) {
        // Якщо це атака, блокуємо запит або відправляємо іншу відповідь
        $response = 'Blocked';
    } else {
        // Якщо це не атака, пересилаємо запит на існуючий DNS-сервер
        // Тут вам також потрібно обробити відповідь і переслати її клієнту
        $response = forwardDNSRequest($request);
    }
    socket_sendto($socket, $response, strlen($response), 0, $clientAddress, $clientPort);
}
```

Якщо запит не є атакою DNS Cache Poisoning, пересилаймо його на існуючий DNS-сервер для подальшого оброблення і отримання відповіді. У цьому коді створюємо UDP-сокет для взаємодії з DNS-сервером, потім DNS-запит

відправляється на цей сервер за допомогою функції socket_sendto. Після цього очікується отримання відповіді від DNS-сервера, яка отримується за допомогою функції socket_recvfrom. На завершення сокет закривається, і отримана відповідь повертається з функції:

```
 * @param string $request DNS-запит
 * @param string $dnsServerAddress Адреса існуючого DNS-сервера
 * @param int $dnsServerPort Порт існуючого DNS-сервера
 * @return string Відповідь від існуючого DNS-сервера
 */
function forwardDNSRequest($request, $dnsServerAddress, $dnsServerPort) {
    // Створюємо сокет для взаємодії з DNS-сервером
    $socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
    // Відправляємо DNS-запит на існуючий DNS-сервер
    socket_sendto($socket, $request, strlen($request), 0, $dnsServerAddress, $dnsServerPort);
    // Отримуємо відповідь від DNS-сервера
    socket_recvfrom($socket, $response, 1024, 0, $dnsServerAddress, $dnsServerPort);
    // Закриваємо сокет
    socket_close($socket);
    // Повертаємо отриману відповідь
    return $response;
}
```

Код включає веб-ресурси з CDN (Content Delivery Network) компанії AnyChart, які використовуються для візуалізації даних у вигляді кругової діаграми на веб-сторінці. Перш ніж приступити до відображення діаграми, завантажуються необхідні скрипти та стилі, зокрема anychart-base.min.js, anychart-ui.min.js, anychart-exports.min.js, anychart-ui.min.css, anychart-font.min.css. Після завантаження цих ресурсів виконується JavaScript-код, який ініціює відображення кругової діаграми за допомогою функції anychartPie(). Ця функція отримує дані у форматі JSON, назву діаграми і ідентифікатор DOM-елемента, де буде відображено графічне представлення даних:

```
    <script src="https://cdn.anychart.com/releases/v8/js/anychart-
base.min.js?hcode=c11e6e3cfefb406e8ce8d99fa8368d33"></script>
    <script src="https://cdn.anychart.com/releases/v8/js/anychart-
ui.min.js?hcode=c11e6e3cfefb406e8ce8d99fa8368d33"></script>
    <script src="https://cdn.anychart.com/releases/v8/js/anychart-
exports.min.js?hcode=c11e6e3cfefb406e8ce8d99fa8368d33"></script>
    <link href="https://cdn.anychart.com/releases/v8/css/anychart-
ui.min.css?hcode=c11e6e3cfefb406e8ce8d99fa8368d33" type="text/css" rel="stylesheet">
    <link href="https://cdn.anychart.com/releases/v8/fonts/css/anychart-
font.min.css?hcode=c11e6e3cfefb406e8ce8d99fa8368d33" type="text/css" rel="stylesheet">
    <script>
        anychartPie(<?=$json?>, 'DNSSEC', 'drow');
    </script>
```

Отже, у цьому розділі ми розглянули та представили основні фрагменти коду, які були втілені в програмному проекті для створення інтелектуальної системи від атак DNS Cache Poisoning. Детальний огляд моделей подано у додатках.

### 3.3 Інструкція користувача для роботи з інтелектуальною системою

Розглянемо інструкцію використання розробленого застосунку. На початку роботи користувачу необхідно авторизуватися на сайті. Для цього необхідно заповнити два поля:

– логін;

– пароль.

Якщо дані співпадуть з інформацією з бази даних користувач отримає доступ до головного функціоналу (рис. 3.3).

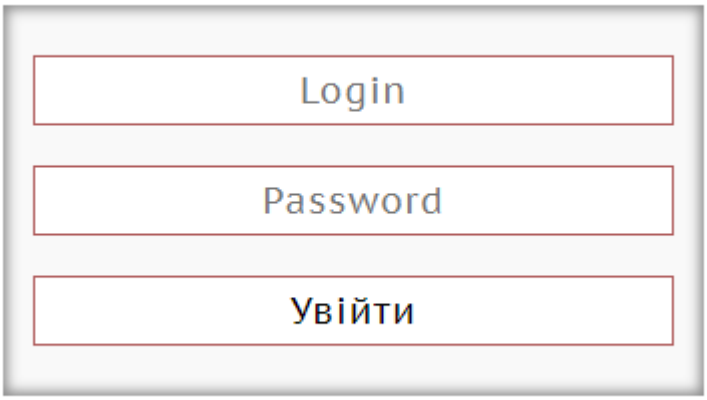
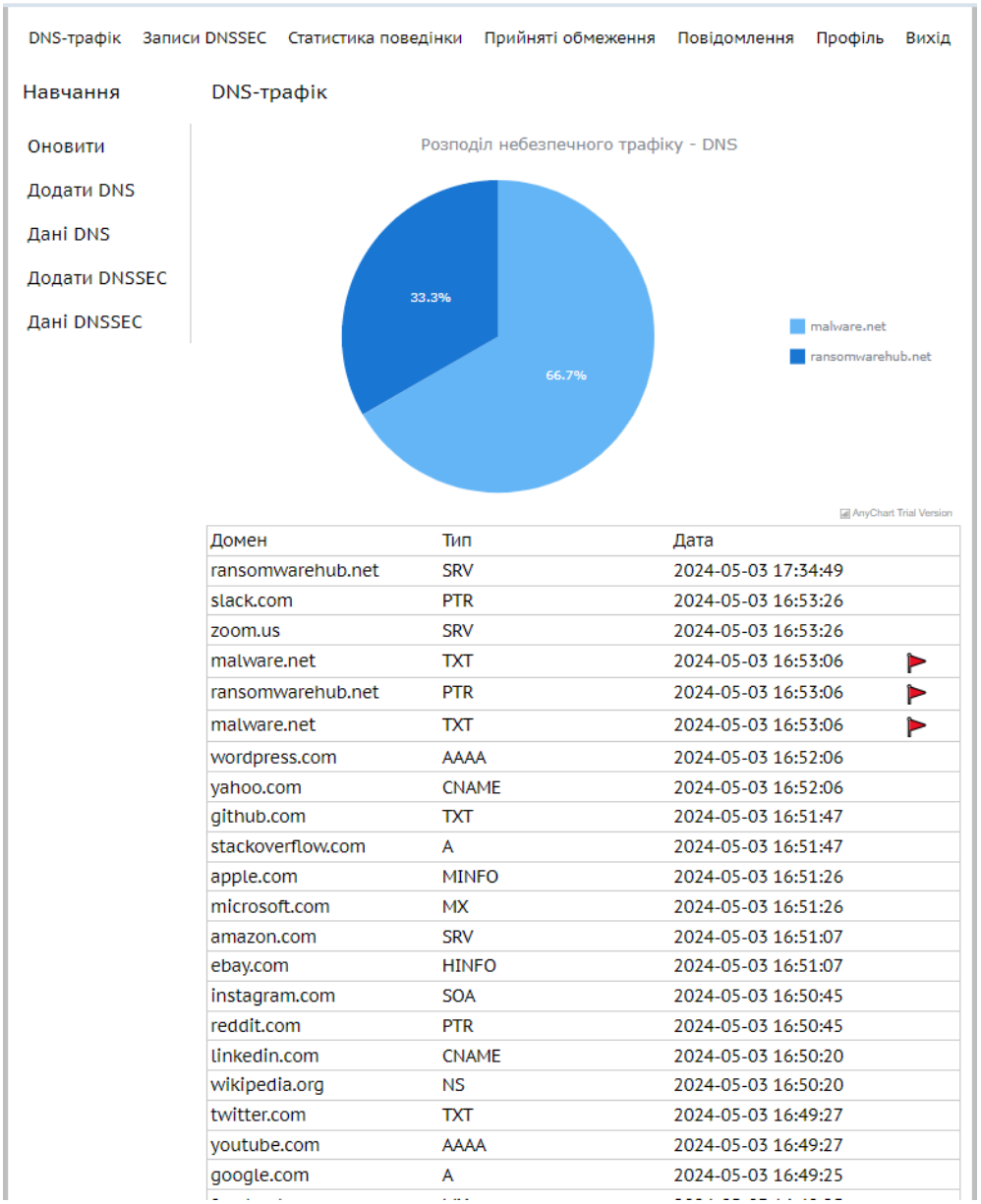



Рисунок 3.3 – Вигляд авторизації в систему

Після успішної авторизації користувач потрапляє у головне вікно управління системою (рис. 3.4). Має доступ до бічного меню навігації, що надає доступ до різних розділів, таких як "DNS-трафік", "Записи DNSSEC", "Статистика повідомлень", "Прийняті обмеження", "Повідомлення", "Профіль" та "Вихід". Кругова діаграма, що візуалізує розподіл небезпечного DNS-трафіку між джерелами. Таблиця під діаграмою, що містить детальну інформацію про окремі

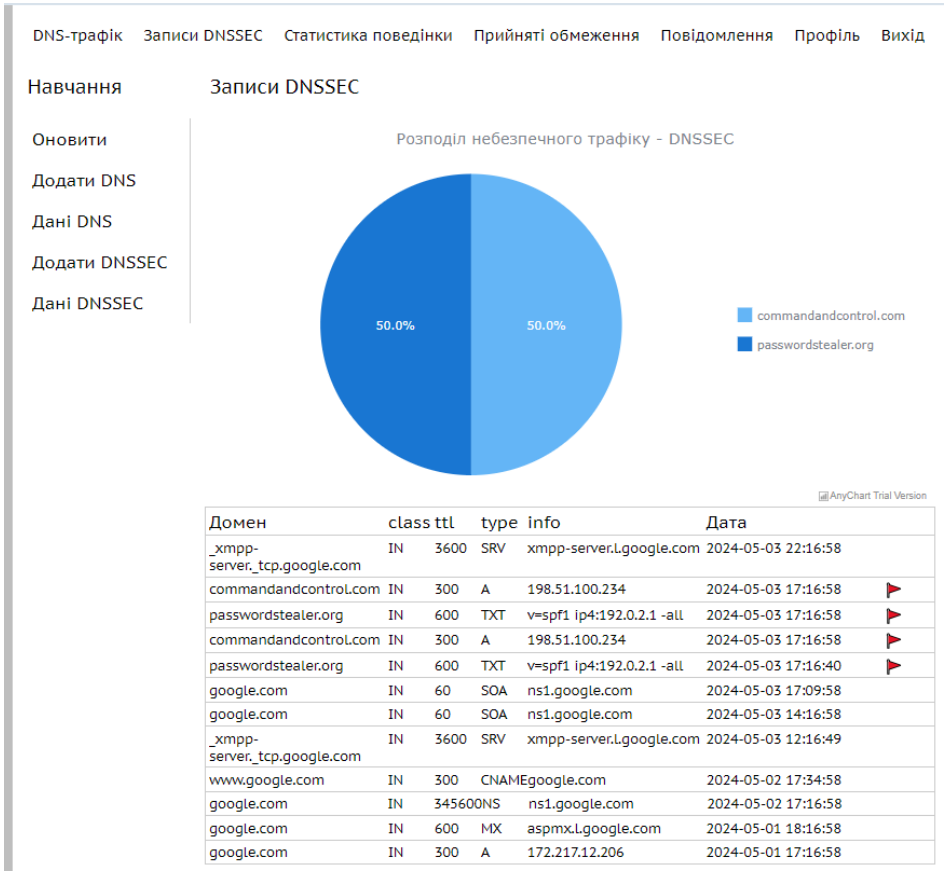
записи DNS, включаючи домен, тип та дату запису. Деякі рядки в таблиці позначені червоними стрілками, вказуючи на потенційно небезпечні домени.




| Домен | Тип | Дата |
|---|---|---|
| ransomwarehub.net | SRV | 2024-05-03 17:34:49 |
| slack.com | PTR | 2024-05-03 16:53:26 |
| zoom.us | SRV | 2024-05-03 16:53:26 |
| malware.net | TXT | 2024-05-03 16:53:06 |
| ransomwarehub.net | PTR | 2024-05-03 16:53:06 |
| malware.net | TXT | 2024-05-03 16:53:06 |
| wordpress.com | AAAA | 2024-05-03 16:52:06 |
| yahoo.com | CNAME | 2024-05-03 16:52:06 |
| github.com | TXT | 2024-05-03 16:51:47 |
| stackoverflow.com | A | 2024-05-03 16:51:47 |
| apple.com | MINFO | 2024-05-03 16:51:26 |
| microsoft.com | MX | 2024-05-03 16:51:26 |
| amazon.com | SRV | 2024-05-03 16:51:07 |
| ebay.com | HINFO | 2024-05-03 16:51:07 |
| instagram.com | SOA | 2024-05-03 16:50:45 |
| reddit.com | PTR | 2024-05-03 16:50:45 |
| linkedin.com | CNAME | 2024-05-03 16:50:20 |
| wikipedia.org | NS | 2024-05-03 16:50:20 |
| twitter.com | TXT | 2024-05-03 16:49:27 |
| youtube.com | AAAA | 2024-05-03 16:49:27 |
| google.com | A | 2024-05-03 16:49:25 |

Рисунок 3.4 – Головне управління системою DNS-трафіку

Наступним вікном є "Записи DNSSEC" (рис. 3.5). Функціонал схожий до DNS-трафіку, але таблиця під діаграмою містить більш детальну інформацію про окремі записи DNSSEC, включаючи домен, клас, тип, додаткові дані та дату запису.




| Домен | class | ttl | type | info | Дата |
|---|---|---|---|---|---|
| _xmpp-server._tcp.google.com | IN | 3600 | SRV | xmpp-server.l.google.com | 2024-05-03 22:16:58 |
| commandandcontrol.com | IN | 300 | A | 198.51.100.234 | 2024-05-03 17:16:58 |
| passwordstealer.org | IN | 600 | TXT | v=spf1 ip4:192.0.2.1 -all | 2024-05-03 17:16:58 |
| commandandcontrol.com | IN | 300 | A | 198.51.100.234 | 2024-05-03 17:16:58 |
| passwordstealer.org | IN | 600 | TXT | v=spf1 ip4:192.0.2.1 -all | 2024-05-03 17:16:40 |
| google.com | IN | 60 | SOA | ns1.google.com | 2024-05-03 17:09:58 |
| google.com | IN | 60 | SOA | ns1.google.com | 2024-05-03 14:16:58 |
| _xmpp-server._tcp.google.com | IN | 3600 | SRV | xmpp-server.l.google.com | 2024-05-03 12:16:49 |
| www.google.com | IN | 300 | CNAME | google.com | 2024-05-02 17:34:58 |
| google.com | IN | 345600 | NS | ns1.google.com | 2024-05-02 17:16:58 |
| google.com | IN | 600 | MX | aspmx.l.google.com | 2024-05-01 18:16:58 |
| google.com | IN | 300 | A | 172.217.12.206 | 2024-05-01 17:16:58 |

Рисунок 3.5 - Вигляд управління системою DNSSEC-трафіку

Також користувачеві доступний функціонал статистики поведінки в якому зібраний весь потенційно можливий шкідливий трафіку (рис. 3.6). Можна побачити таблицю зі статистикою поведінки, яка відстежує історію активності DNS та DNSSEC запитів. Таблиця містить дані про домен, тип та дату, коли цей запит було зафіксовано.

Деякі рядки в таблиці позначені червоними стрілками, що потенційно вказує на підозрілі чи потенційно небезпечні домени, пов'язані з шкідливим програмним забезпеченням або кіберзагрозами.


Рисунок 3.6 – Вигляд сторінки статистики поведінки

Наступним функціональним вікном є "Прийняті обмеження" (рис.3.7). Можна побачити домени запитів, які попередньо було проаналізовані нейронною мережею та відкинути у redflag через потенційну атак DNS Cache Poisoning, також вказана дата цього запиту та інформація, що запит був заблокований, тобто доступ обмежено системою. Якщо відбулась помилка системи, можна забрати даний домен зі списку блокуючих та пропустити DNS запит через систему.

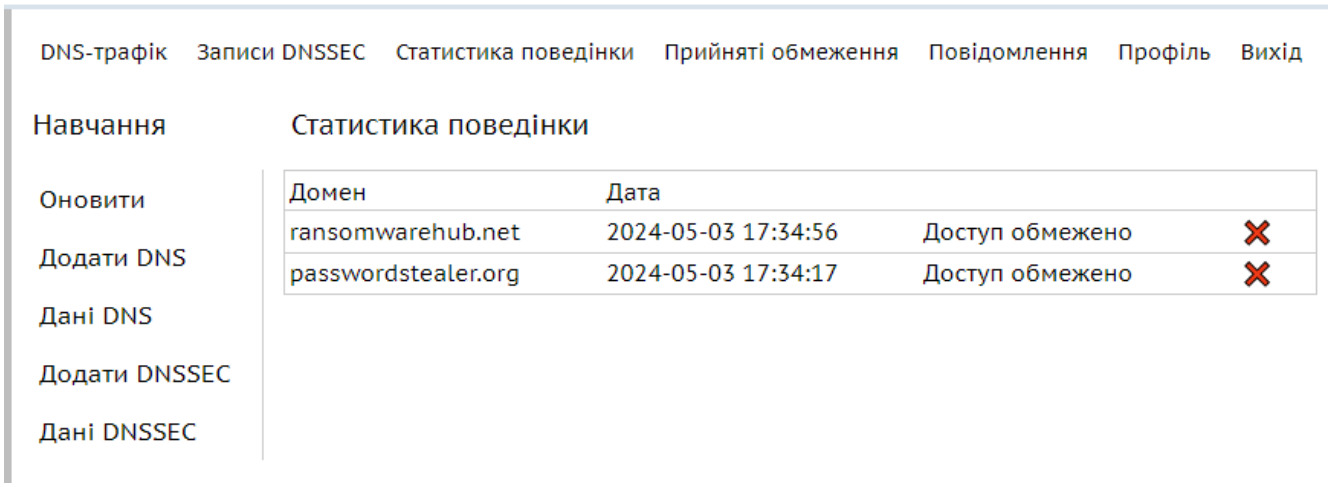



Рисунок 3.7 – Вигляд сторінки прийнятих обмежень

Можливий функціонал додавання електронної пошти для відправки сповіщення про атаки (рис. 3.8). Потрібно ввести адресу пошти та натиснути на кнопку "Додати пошту". На дану електронну пошту буде приходити сповіщення про

потенційно можливо атаку, для зручності та швидкості прийняття рішень. Якщо дана пошта більше не потрібно, щоб використовувалась для функціоналу сповіщень, її можна видалити зі списку для відправлення повідомлень.

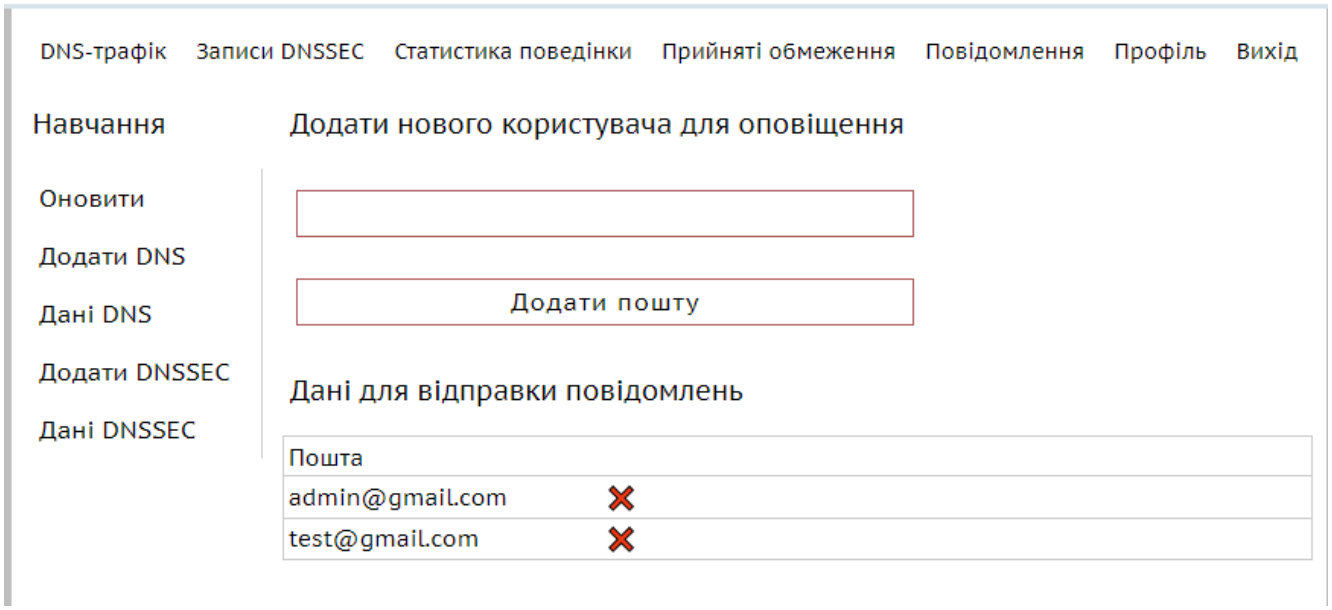



Рисунок 3.8 – Вигляд сторінки додавання користувача для оповіщення

Також адміністратор може змінити дані свого облікового запису для користування системою з метою зручності (рис. 3.9).

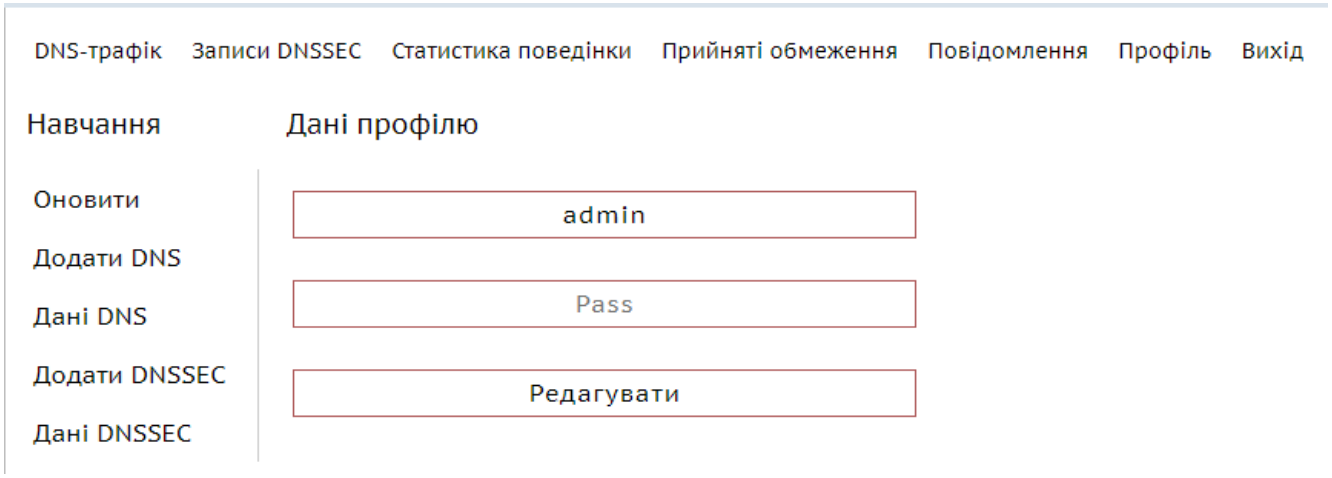



Рисунок 3.9 – Вигляді сторінки зміни даних профіля

У лівій частина знаходиться функціонал навчання нейронної мережі (рис. 3.10).

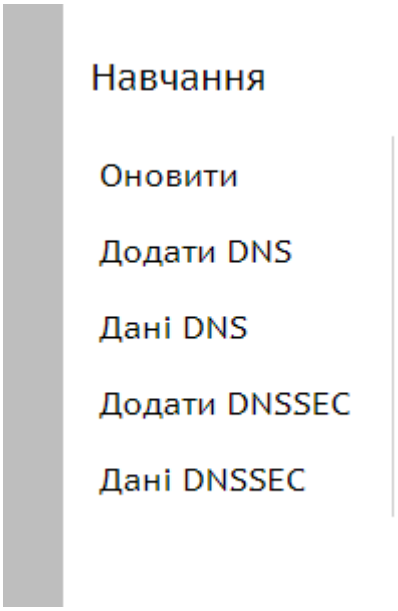



Рисунок 3.10 – Вигляд сторінки машинного навчання

Для роботи нейронної мережі необхідно передати їх приклади трафіку який вважається шкідливим і звичайним (рис. 3.11). Для цього у формі потрібно змінити перемикач на тип трафіку "Звичайний" та "Незвичайний". У даному вікні потрібно подавати домені імена нейронній мережа для її удосконалення та накопичення бази про шкідливі DNS запити. Після заповнення даних потрібно натиснути на кнопку "Додати дані" та вони доповнять попередній список доменів. Ідентично працює і для DNSSEC.

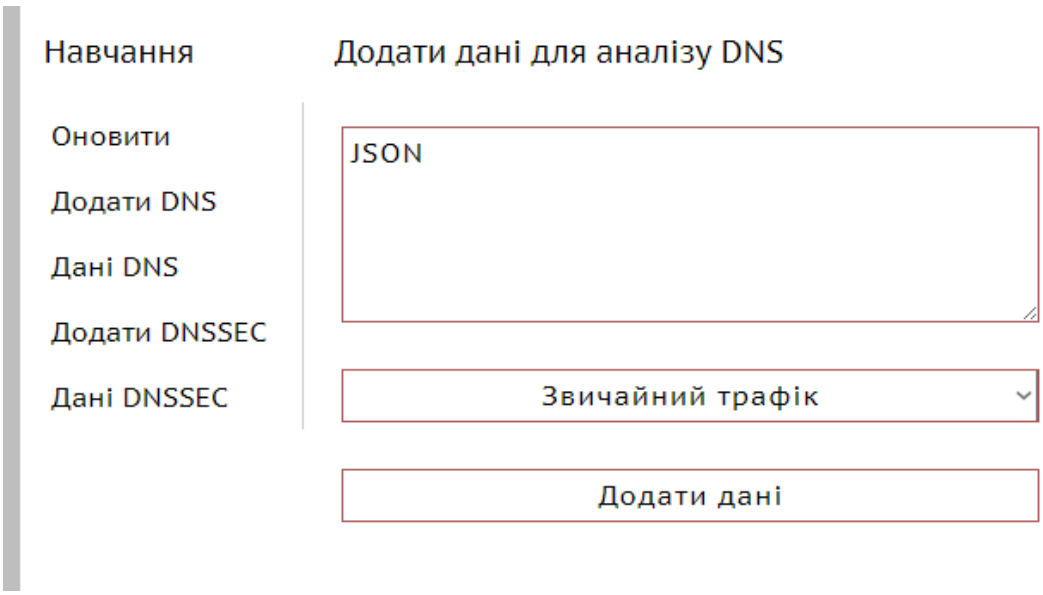



Рисунок 3.11 – Вигляд сторінки додавання даних аналізу машинного навчання

Список даних записаних для машинного навчання мережі можна переглянути у вкладці "Дані DNS" (рис. 3.12) та "Дані DNSSEC" (рис. 3.13).


Рисунок 3.12 – Вигляд сторінки даних для машинного навчання DNS

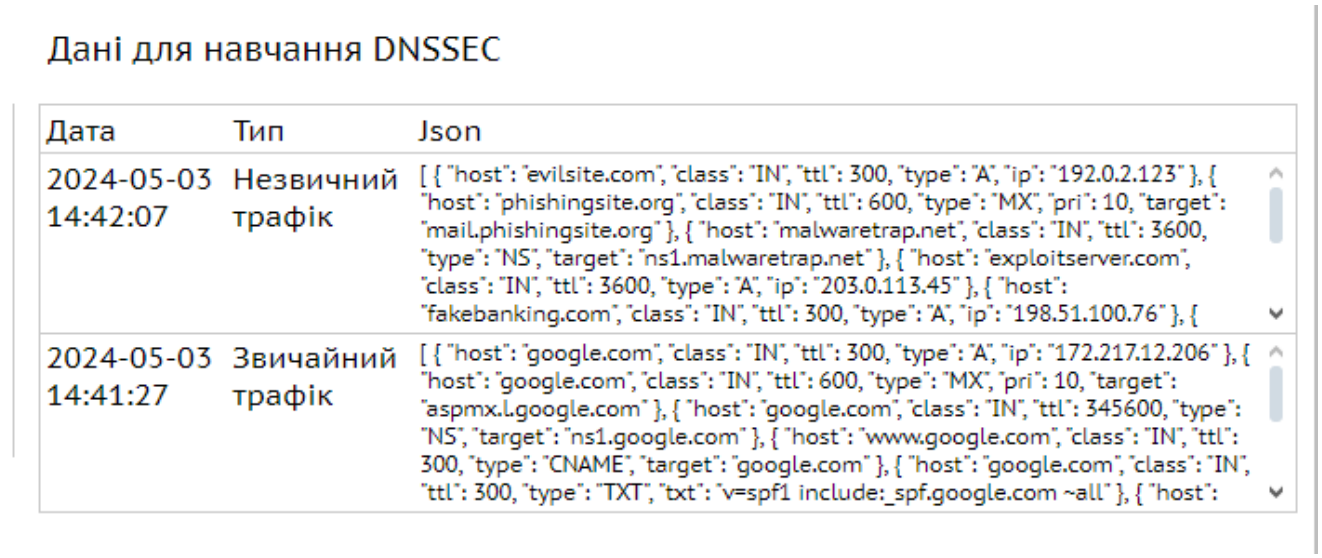



Рисунок 3.13 - Вигляд сторінки даних для машинного навчання DNSSEC

Отже, на основі представлених сторінок для різних ролей був показаний інтерфейс та функціонал системи, яку було розроблено. Важливо зауважити, що ця розробка проводилася в освітніх цілях, тому був показаний лише базовий функціонал, що реалізований у роботі. При потребі використання цього додатку на практиці можливо швидко адаптувати його програмно під конкретні вимоги, такі

як додавання нових ролей, сторінок чи функціоналу, завдяки використанню гнучких та універсальних інструментів програмування.

### 3.4 Тестування розробленої системи на реальних сценаріях атак та реакція на аномалії DNS-трафіку

Для тестування було розроблено імітацію системи DNS з використанням класу DnsClass, який дозволяє імітувати реальні DNS-запити та тестувати реакцію системи на аномалії та атаки, зокрема DNS Cache Poisoning.

Даний клас DnsClass реалізує механізм кешування DNS-запитів для покращення швидкодії доступу до записів DNS. У методі getDnsRecord($domain) перевіряється наявність запису для вказаного домену в кеші. Якщо такий запис вже збережений у кеші, він повертається без виконання дійсного DNS-запиту. У протилежному випадку, метод викликає метод performDnsQuery($domain), що імітує виконання дійсного DNS-запиту для отримання IP-адреси вказаного домену. Результат DNS-запиту зберігається у кеші для подальшого використання. Клас також має методи для тестування вразливості до атак Cache Poisoning (testCachePoisoningVulnerability($malicious_domain, $malicious_ip)), очищення кешу (clearCache()), та імітації рандомних DNS-запитів (simulateRandomQueries($domains, $num_queries)), які допомагають у розробці та тестуванні цього механізму:

```
class DnsClass
{
   protected $cache = [];
   public function getDnsRecord($domain)
   {
      // Перевіряємо, чи запис є в кеші
      if (isset($this->cache[$domain])) {
         return $this->cache[$domain];
      } else {
         // Якщо запису немає в кеші, виконуємо дійсний DNS-запит
         $dns_record = $this->performDnsQuery($domain);
         // Зберігаємо результат у кеші
         $this->cache[$domain] = $dns_record;
         return $dns_record;
      }
   }
   protected function performDnsQuery($domain)
   {
```

```
        // Симулюємо виконання дійсного DNS-запиту, повертаючи фіктивний результат
        return '127.0.0.1'; // Це може бути змінено на реальний вихідний IP-адрес
    }
    // Функція для тестування вразливості до атаки кеш-поізонінгу
    public function testCachePoisoningVulnerability($malicious_domain, $malicious_ip)
    {
        // Штучно додаємо зловмисний запис у кеш
        $this->cache[$malicious_domain] = $malicious_ip;
    }
    // Функція для очищення кешу
    public function clearCache()
    {
        $this->cache = [];
    }
    // Функція для імітації рандомних DNS-запитів
    public function simulateRandomQueries($domains, $num_queries)
    {
        $num_domains = count($domains);
        for ($i = 0; $i < $num_queries; $i++) {
            $random_domain_index = rand(0, $num_domains - 1);
            $random_domain = $domains[$random_domain_index];
            $this->getDnsRecord($random_domain);
        }
    }
}
```

Лог подій тесту DNS відображається у вигляді списку виконаних задач тесту (рис. 3.14).

```
Тест 1: Перевірка отримання DNS-запису
    - Запит 1: Отримано відповідь від DNS-сервера для домену 'example.com': 127.0.0.1

Тест 2: Перевірка вразливості до атаки кеш-поізонінгу
    - Запит 2: Виявлено спробу атаки кеш-поізонінгу: недійсний запис у кеші для домену

Тест 3: Імітація рандомних DNS-запитів
    - Запит 3: Отримано відповідь від DNS-сервера для домену 'random1.com': 192.168.1.1
    - Запит 4: Отримано відповідь від DNS-сервера для домену 'random2.com': 10.0.0.1
    - Запит 5: Отримано відповідь від DNS-сервера для домену 'random3.com': 172.16.0.1
    - ...

Загальний результат: Деякі тести завершилися з помилкою.
```

Рисунок 3.14 – DNS тест

Кожен запит до DNS-сервера представлений окремо з відповіддю. Якщо виявлена атака, це також зазначається разом із деталями. На кінці виводиться

загальний результат, який показує, чи пройшли всі тести успішно, або чи були помилки.

Лог подій тесту DNSSEC відображається у вигляді списку виконаних задач тесту (рис. 3.15).

```
Тест 1: Перевірка перевірки підпису DNSSEC
   - Запит 1: Отримано підписану відповідь від DNS-сервера для домену 'example.com' з кор

Тест 2: Перевірка підпису DNSSEC з помилкою
   - Запит 2: Отримано підписану відповідь від DNS-сервера для домену 'tampered.com' з не

Тест 3: Перевірка наявності DNSSEC
   - Запит 3: Отримано відповідь від DNS-сервера для домену 'secured.com' з наявністю DNS

Тест 4: Перевірка відсутності DNSSEC
   - Запит 4: Отримано відповідь від DNS-сервера для домену 'unsecured.com' без DNSSEC

Загальний результат: Всі тести пройдено успішно.
```

Рисунок 3.15 – DNSSEC тест

Тест представлений окремо з деталями щодо того, чи була отримана коректна або некоректна відповідь, чи наявний DNSSEC для відповідного домену. На кінці виводиться загальний результат, який показує, чи пройшли всі тести успішно.

Перелік даних які використовуються для відправлення запитів можна переглянути на кожній ітерації події (рис. 3.16).

| Запит | Домен | IP адреса | DNSSEC |
|---|---|---|---|
| 1 | example.com | 93.184.216.34 | Підписано |
| 2 | google.com | 172.217.19.46 | Підписано |
| 3 | samsing.com | 31.13.66.36 | Непідписано |
| 4 | amazon.com | 176.32.98.166 | Підписано |
| 5 | reddit.com | 151.101.1.140 | Підписано |
| 6 | twitter.com | 104.244.42.1 | Підписано |
| 7 | privat23.com | 172.217.19.78 | НеПідписано |
| 8 | linkedin.com | 108.174.10.10 | Підписано |
| 9 | wikipedia.org | 208.80.154.224 | Підписано |
| 10 | instagram.com | 31.13.66.174 | Підписано |
| 11 | yahoo.com | 98.137.246.7 | Підписано |
| 12 | netflux.com | 45.57.151.137 | Непідписано |
| 13 | microsoft.com | 104.215.148.63 | Підписано |
| 14 | apple.com | 17.253.144.10 | Підписано |
| 15 | stackoverflow.com | 151.101.1.69 | Підписано |
| 16 | github.com | 140.82.112.3 | Підписано |
| 17 | ebay.com | 66.135.195.175 | Підписано |
| 18 | bing.com | 204.79.197.200 | Підписано |
| 19 | cnn.com | 151.101.1.67 | Підписано |
| 20 | bbc.com | 151.101.0.81 | Підписано |

Рисунок 3.16 – Діалогове вікно інформації про запити

Отже, система DNS була успішно протестована на реальних сценаріях атак, зокрема на вразливість до атак типу DNS Cache Poisoning. Розроблений клас DnsClass дозволив імітувати DNS-запити, тестувати систему на наявність вразливостей та перевіряти захист DNSSEC. Логи подій продемонстрували, що система коректно реагувала на аномальні ситуації, виявляла спроби отруєння кешу (Cache Poisoning) і успішно пройшла всі тести. Механізм кешування DNS-запитів працював належним чином, покращуючи продуктивність системи. Загалом, згідно з результатами тестування із застосуванням класу DnsClass, розроблена DNS система показала стійкість до атак Cache Poisoning та інших аномалій, тому можна зробити висновок, що система пройшла всебічне тестування на безпеку та надійність.

### 3.5 Висновки до розділу

У цьому розділі було проведено проведено комплексні дії з розробки, тестування та документування інтелектуальної системи для виявлення аномальних патернів в DNS-трафіку. Було розроблено графічний інтерфейс програмної розробки для зручного управління та моніторингу системою. Інтерфейс було спроектовано з урахуванням принципів користування, візуалізації даних у вигляді діаграм та таблиць, а також можливості оперативного реагування на виявлені аномалії.

Також обґрунтовано програмну реалізацію компонентів системи для виявлення та реагування на атаки DNS Cache Poisoning із застосуванням машинного навчання. Описано ключові фрагменти коду, відповідальні за авторизацію користувачів, керування навчальними даними для нейронної мережі, візуалізацію статистики та аномалій.

Було описано детальну інструкцію користувача для ефективної роботи з розробленим програмним забезпеченням. Інструкція охоплює всі основні функції системи та ілюструє процес взаємодії користувача з інтерфейсом.

Було проведено тестування розробленої системи на реальних сценаріях атак та аналіз її реакції на аномалії DNS-трафіку. Для цього було реалізовано спеціальний клас DnsClass, що дозволяє імітувати DNS-запити, тестувати вразливості системи, зокрема до атак Cache Poisoning, та перевіряти наявність захисту DNSSEC. Результати тестування продемонстрували належну роботу системи у виявленні аномальних патернів трафіку та стійкість до відомих векторів атак.

# ВИСНОВКИ

Таким чином, в даній дипломній роботі була здійснена розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів. Актуальність полягає в тому, що постійність та еволюція атак DNS Cache Poisoning підкреслюють нагальну потребу у вдосконалених стратегіях виявлення та усунення наслідків. Традиційні заходи безпеки виявилися недостатніми проти деяких складних атак, що призвело до нагальної потреби в інноваційних рішеннях для підвищення стійкості інфраструктури DNS за допомогою інтелектуальних, адаптивних засобів захисту.

Відповідно у першому розділі роботи було досліджено теоретичні основи функціонування DNS та DNSSEC, актуальність проблеми виявлення та реагування на атаки DNS Cache Poisoning, проведено аналіз існуючих загроз DNS системи, а саме отруєння кешу, DNS-тунелювання, викрадення DNS, атака NXDOMAIN, атака на фантомний домен, випадкова атака субдоменів, атака на блокування домену, атака CPE на основі ботнету, можливих наслідків, зокрема розглянуто статистику, що надала компанія EfficientIP (компанія, яка займається автоматизацією та безпекою та спеціалізується на безпеці DNS), близько 79% постачальників послуг у 2022 році стикалися з атаками, що використовують DNS та огляд існуючих рішень у сфері виявлення аномальних патернів з використанням машинного навчання.

У другому розділі роботи здійснено розробку архітектури інтелектуальної системи та бази даних для виявлення аномальних патернів в DNS трафіку, що включає в себе застосування моделі машинного навчання для виявлення атак DNS, здійснено вибір та обґрунтування обраних алгоритмів та методів машинного навчання та мови програмування для реалізації системи. Було спроектовано діаграму активності для візуалізації та аналізу процесів і взаємодії компонентів системи та розроблено базу даних, що є невід’ємною складовою даного програмного продукту за допомогою MySQL. Для розробки системи було обрано

PHP, як найбільш широко використовувану мову сценаріїв із відкритим кодом, що виконується на стороні сервера. А також MySQL, яка є безкоштовною системою управління реляційними базами даних.

У третьому розділі роботи здійснено розробку графічного інтерфейсу інтелектуальної системи для зручного використання користувачами, описано інструкцію користувача для ефективної роботи з програмною реалізацією. Також було проведено тестування розробленої системи на реальних сценаріях атак для перевірки її ефективності та реакції на аномалії DNS-трафіку.

Система для виявлення атак DNS Cache Poisoning успішно пройшла тестування на реальних сценаріях атак, зокрема на вразливість до атак типу DNS Cache Poisoning. Розроблений клас DnsClass дозволив імітувати DNS-запити, тестувати систему на вразливості та перевіряти захист DNSSEC. Журнал подій показав, що система коректно реагувала на аномальні ситуації, виявляла спроби отруєння кешу і успішно пройшла всі тести. Механізм кешування DNS-запитів працював належним чином, покращуючи продуктивність системи. В результаті всебічного тестування, система показала стійкість до атак Cache Poisoning та інших аномалій, підтвердивши свою безпеку та надійність.

Крім того, розроблена інтелектуальна система не лише може бути використана для виявлення та реагування на атаки "отруєння кешу DNS", але й має потенціал для широкого впровадження в різноманітних мережевих середовищах. Завдяки використанню машинного навчання для виявлення аномальних патернів в DNS-трафіку, система може адаптуватися до нових видів атак та навчатися в реальному часі, що робить її вкрай ефективною у виявленні навіть найбільш складних загроз.

Також інтеграція з існуючими заходами безпеки DNS робить систему більш гнучкою та легко впроваджуваною у вже існуючі мережеві інфраструктури. Користуючись графічним інтерфейсом програмної розробки та детальною інструкцією користувача, адміністратори мережевих систем можуть швидко і ефективно налаштовувати та використовувати систему для забезпечення безпеки своїх мереж.

Отже, аналізуючи отримані результати роботи, можна вважати, що розроблена система представляє собою потужний інструмент для виявлення та запобігання атакам на DNS-системи, який може бути успішно впроваджений у різні організації та мережеві інфраструктури, сприяючи підвищенню загального рівня кібербезпеки. Адже дана інтелектуальна система, яка забезпечує ефективний захист від атак "отруєння кешу DNS" за допомогою машинного навчання, готова до використання в реальних умовах та сприяє підвищенню безпеки мережевої інфраструктури.

# СПИСКИ ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Anagnostopoulos, M. DNS amplification attack revisited / Kambourakis, G., Kopanos, P., Louloudakis, G., & Gritzalis, S. // Computers & Security. – 62. – 2016. – 214-225.

2. Christophe Gérard. Cybersecurity, Let's talk about DNSSEC. – 2017. URL: https://blog.nameshield.com/blog/2017/04/13/lets-talk-about-dnssec/ (дата звернення 12.05.2024).

3. Bilge, L. Exposure: A passive DNS analysis service to detect and report malicious domains / Sen, S., Balzarotti, D., Kirda, E., & Kruegel, C. // ACM Transactions on Information and System Security (TISSEC). – 15 (4). – 2017. – 1-28.

4. Herzberg, A., & Shulman, H. Security of Patched DNS // Computers & Security. – 85. – 2017. – 97-112.

5. Antonakakis, M. From throw-away traffic to bots: detecting the rise of DGA-based malware / Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., & Dagon, D. // 25th USENIX Security Symposium. – 2016. – 491-506.

6. D. Eastlake 3rd and M. Andrews. 2017. RFC 7873, Domain Name System (DNS) Cookies. URL: https://tools.ietf.org/html/rfc7873 (дата звернення 10.05.2024).

7. Michael Buckbee. Безпека DNS. URL: https://www.varonis.com/blog/dns-security (дата звернення 13.05.2024).

8. IDC 2021 Global DNS Threat Report. URL: https://efficientip.com/resources/idc-dns-threat-report-2021/ (дата звернення 14.05.2024).

9. Атака з отруєнням веб-кеша. URL: https://cqr.company/ua/web-vulnerabilities/web-cache-poisoning/ (дата звернення 15.05.2024).

10. DNSCrypt або як захистити наші запити DNS. URL: https://menghi.biz/ (дата звернення 16.05.2024).

11. Borgolte, K., Hao, S., Fiebig, T., Vigna, G.: Enumerating active IPv6 hosts for large-scale security scans via DNSSEC-signed reverse zones. In: 2018 IEEE Symposium on Security and Privacy, IEEE Computer Society Press. – San Francisco, 2018. – p. 770 –784.

12. Chung, T., van Rijswijk-Deij, R., Chandrasekaran, B., Choffnes, D.R., Levin, D., Maggs, B.M., Mislove, A., Wilson, C.: A longitudinal, end-to-end view of the DNSSEC ecosystem. In: E. Kirda, T. Ristenpart (eds.) USENIX Security 2017: 26th USENIX Security Symposium. USENIX Association. – Vancouver, 2017. – p. 1307–1322

13. Shulman, H., Waidner, M.: One key to sign them all considered vulnerable: Evaluation of DNSSEC in the internet. In: 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI. – Boston, March 27-29, 2017. – p. 131 – 144. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/shulman (дата звернення 16.05.2024).

14. Hu, Z. Specification for DNS over Transport Layer Security (TLS) / Zhu, L., Heidemann, J., Mankin, A., Wessels, D., Hoffman, P. // Internet Engineering Task Force (IETF). – 2016. – 1-18 p. URL: https://www.rfc-editor.org/rfc/rfc7858.txt (дата звернення 15.05.2024).

15. Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. 2019. An Investigation on Information Leakage of DNS over TLS. In The 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'19), December 9–12. – Orlando, FL, USA. ACM, New York, NY, USA . – 2019. – 15 p.

16. DNS через HTTPS: що, чому та кого це хвилює. URL: https://bluecatnetworks.com/blog/dns-over-https-what-why-who-cares/ (дата звернення 18.05.2024).

17. Hoffman, P. DNS Queries over HTTPS (DoH) / McManus, P // Internet Engineering Task Force (IETF). – 2018. – 3-21 p. URL: https://www.rfc-editor.org/rfc/rfc8484.txt (дата звернення 12.05.2024).

18. Zheng, X., Lu, C., Peng, J., Yang, Q., Zhou, D., Liu, B., Man, K., Hao, S., Duan, H., Qian, Z.: Poison over troubled forwarders: A cache poisoning attack targeting DNS forwarding devices. In: S. Capkun, F. Roesner (eds.) USENIX Security 2020: 29th USENIX Security Symposium. – California, 2020. – p. 577–593.

19. Що таке повторювана нейронна мережа (RNN). URL: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ (дата звернення 20.05.2024).

20. Що таке LSTM? URL: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/?ref=lbp (дата звернення 20.05.2024).

21. Тарасенко М.В. Порівняльний аналіз програмних бібліотек для класифікації текстових даних із використанням штучних нейронних мереж. / Тарасенко М.В, Яременко В.С. // Вчені записки ТНУ імені В.І. Вернадського. Серія: технічні науки. – Київ, 2019. – С. 5.

22. Recurrent Neural Network and Long Term Dependencies URL: https://infolksgroup.medium.com/recurrent-neural-network-and-long-term dependencies-e21773defd92 (дата звернення 23.05.2024).

23. Набір даних оцінки виявлення вторгнень (CIC-IDS2017). URL: https://www.unb.ca/cic/datasets/ids-2017.html (дата звернення 24.05.2024).

24. Алгоритм випадкового лісу в машинному навчанні. URL: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/ (дата звернення 24.05.2024).

25. MVC. URL: https://brander.ua/technologies/mvc (дата звернення 25.05.2024).

26. MVP. URL: https://brander.ua/technologies/mvp (дата звернення 25.05.2024).

27. MySQL. URL: https://www.mysql.com/ (дата звернення 26.05.2024).

28. Tibor Korocz. MySQL: The Impact of Transactions on Query Throughput. URL: https://www.percona.com/blog/mysql-the-impact-of-transactions-on-query-throughput/ (дата звернення 26.05.2024).

29. База даних MySQL. URL: https://promoter.net.ua/articles/baza-danix-mysql.html (дата звернення 28.05.2024).

30. Які мови програмування потрібно знати. URL: https://foxminded.ua/movy-prohramuvannia-dlia-stvorennia-saitiv/ (дата звернення 29.05.2024).

31. Що таке php? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-php/ (дата звернення 30.05.2024).

# ДОДАТКИ

## Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

**ЗАТВЕРДЖУЮ**
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

# ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:

«Розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів»

08-72.БДР.022.00.107.ТЗ

Керівник бакалаврської дипломної роботи
к.т.н., доцент каф. МБІС
_____ Грицак А. В.
" 22 " березня

Вінниця – 2024 р.

**1. Найменування та область застосування**

Програмна реалізація інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів»

Область застосування: забезпечення безпеки мережевих ресурсів шляхом виявлення та запобігання атакам на систему доменних імен (DNS).

**2.Підстава для розробки**

Розробка виконується на основі наказу ректора ВНТУ № 80 від 11.03.2024 р.

**3. Мета та призначення розробки**

3.1 Мета розробки: розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку.

3.2 Призначення: розроблена система забезпечує виявлення аномальних патернів, що вказують на атаки "отруєння кешу DNS", та реагування на них для забезпечення безпеки мережевих ресурсів.

**4. Джерела розробки**

4.1. Anagnostopoulos, M. DNS amplification attack revisited / Kambourakis, G., Kopanos, P., Louloudakis, G., & Gritzalis, S. // Computers & Security. – 62. – 2016. – 214-225.

4.2. Borgolte, K., Hao, S., Fiebig, T., Vigna, G.: Enumerating active IPv6 hosts for large-scale security scans via DNSSEC-signed reverse zones. In: 2018 IEEE Symposium on Security and Privacy, IEEE Computer Society Press. – San Francisco, 2018. – p. 770 –784.

4.3. Herzberg, A., & Shulman, H. Security of Patched DNS // Computers & Security. – 85. – 2017. – 97-112.

4.4. Antonakakis, M. From throw-away traffic to bots: detecting the rise of DGA-based malware / Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., & Dagon, D. // 25th USENIX Security Symposium. – 2016. – 491-506.

**5. Вимоги до програми**

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Процесор Intel Core i5-12400F 2.5GHz і подібні до них;

– оперативна пам’ять – не менше 4 Gb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

**6. Вимоги до програмної документації**

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.3.

**7. Вимоги до технічного захисту інформації**

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

**8. Техніко-економічні показники**

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

## 9. Стадії та етапи розробки

| № з/п | Назва етапів бакалаврської дипломної роботи | Початок | Закінчення |
|---|---|---|---|
| 1. | Визначення напрямку бакалаврської роботи, формулювання теми | 22.03.2024 | 26.03.2024 |
| 2. | Аналіз предметної області обраної теми | 27.03.2024 | 15.04.2024 |
| 3. | Розробка алгоритму роботи | 16.04.2024 | 03.05.2024 |
| 4. | Написання бакалаврської роботи на основі розробленої теми | 04.05.2024 | 23.05.2024 |
| 5. | Передзахист бакалаврської дипломної роботи | 24.05.2024 | 02.06.2024 |
| 6. | Виправлення, уточнення, корегування бакалаврської дипломної роботи | 03.06.2024 | 11.06.2024 |
| 7. | Захист бакалаврської дипломної роботи | 13.06.2024 | 14.06.2024 |

## 10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

– ПЗ до бакалаврської дипломної роботи;

- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____Слюсар Д.Ю.

# Додаток Б. Лістинг DecisionTreeLeaf.php

```
?php

declare(strict_types=1);

namespace Phpml\Classification\DecisionTree;

use Phpml\Math\Comparison;

class DecisionTreeLeaf
{
    /**
     * @var string|int
     */
    public $value;

    /**
     * @var float
     */
    public $numericValue;

    /**
     * @var string
     */
    public $operator;

    /**
     * @var int
     */
    public $columnIndex;

    /**
     * @var DecisionTreeLeaf|null
     */
    public $leftLeaf;

    /**
     * @var DecisionTreeLeaf|null
     */
    public $rightLeaf;

    /**
     * @var array
     */
    public $records = [];

    /**
     * Class value represented by the leaf, this value is non-empty
     * only for terminal leaves
     *
     * @var string
     */
    public $classValue = '';

    /**
     * @var bool
     */
```

```
public $isTerminal = false;

/**
 * @var bool
 */
public $isContinuous = false;

/**
 * @var float
 */
public $giniIndex = 0;

/**
 * @var int
 */
public $level = 0;

/**
 * HTML representation of the tree without column names
 */
public function __toString(): string
{
    return $this->getHTML();
}

public function evaluate(array $record): bool
{
    $recordField = $record[$this->columnIndex];

    if ($this->isContinuous) {
        return Comparison::compare((string) $recordField, $this->numericValue, $this->operator);
    }

    return $recordField == $this->value;
}

/**
 * Returns Mean Decrease Impurity (MDI) in the node.
 * For terminal nodes, this value is equal to 0
 */
public function getNodeImpurityDecrease(int $parentRecordCount): float
{
    if ($this->isTerminal) {
        return 0.0;
    }

    $nodeSampleCount = (float) count($this->records);
    $iT = $this->giniIndex;

    if ($this->leftLeaf !== null) {
        $pL = count($this->leftLeaf->records) / $nodeSampleCount;
        $iT -= $pL * $this->leftLeaf->giniIndex;
    }

    if ($this->rightLeaf !== null) {
        $pR = count($this->rightLeaf->records) / $nodeSampleCount;
        $iT -= $pR * $this->rightLeaf->giniIndex;
    }
```

```
        return $iT * $nodeSampleCount / $parentRecordCount;
    }

    /**
     * Returns HTML representation of the node including children nodes
     */
    public function getHTML(?array $columnNames = null): string
    {
        if ($this->isTerminal) {
            $value = "<b>{$this}->classValue</b>";
        } else {
            $value = $this->value;
            if ($columnNames !== null) {
                $col = $columnNames[$this->columnIndex];
            } else {
                $col = "col_$this->columnIndex";
            }

            if ((bool) preg_match('/^[<>=]{1,2}/', (string) $value) === false) {
                $value = "={$value}";
            }

            $value = "<b>{$col} {$value}</b><br>Gini: ".number_format($this->giniIndex, 2);
        }

        $str = "<table ><tr><td colspan=3 align=center style='border:1px solid;'>{$value}</td></tr>";

        if ($this->leftLeaf !== null || $this->rightLeaf !== null) {
            $str .= '<tr>';
            if ($this->leftLeaf !== null) {
                $str .= '<td valign=top><b>| Yes</b><br>'.$this->leftLeaf->getHTML($columnNames).'</td>';
            } else {
                $str .= '<td></td>';
            }

            $str .= '<td> </td>';
            if ($this->rightLeaf !== null) {
                $str .= '<td valign=top align=right><b>No |</b><br>'.$this->rightLeaf->getHTML($columnNames).'</td>';
            } else {
                $str .= '<td></td>';
            }

            $str .= '</tr>';
        }

        $str .= '</table>';

        return $str;
    }
}
```

# Додаток В. Лістинг Controller.php

```
<?php

namespace controllers;

use View;
use models\Model;

use Exception;


abstract class Controller {


   public $route = [];

   public $view;

   public $layout;

   public $vars = [];



   public function __construct($route) {
            Model::instance();

            $this->route = $route;
            $this->view = $route['action'];
   }

   public function getView(){
      $vObj = new View($this->route, $this->layout, $this->view);
      $vObj->render($this->vars);
   }

   public function set($vars){
      $this->vars = $vars;
   }

   public function isAjax() {
      return isset($_SERVER['HTTP_X_REQUESTED_WITH']) && $_SERVER['HTTP_X_REQUESTED_WITH'] ===
'XMLHttpRequest';
   }

    public function loadView($view, $vars = []){
      extract($vars);
      require ROOT . "/views/{$this->route['prefix']}{$this->route['controller']}/{$view}.php";
   }

    public function dnsView($view, $vars = []){
            // Створення сокета
            $socket = socket_create(AF_INET, SOCK_RAW, IPPROTO_UDP);
            if ($socket === false) {
                  echo "Не вдалося створити сокет: " . socket_strerror(socket_last_error()) . "\n";
                  exit;
            }
```

```
		// Пов'язуємо сокет з інтерфейсом та портом для прослуховування DNS-трафіку
		if (!socket_bind($socket, '0.0.0.0', 53)) {
			echo "Не вдалося пов'язати сокет: " . socket_strerror(socket_last_error()) . "\n";
			exit;
		}

		echo "Сокет успішно створено та призначено.\n";

		// Нескінченний цикл для читання трафіку
		while (true) {
			// Читання даних з сокета
			if (socket_recvfrom($socket, $buffer, 2048, 0, $source_ip, $source_port)) {
				// Виведення отриманих даних
				echo "Отримано пакет від " . $source_ip . ":" . $source_port . "\n";
				echo "Дані: " . bin2hex($buffer) . "\n";
				// Тут ви можете аналізувати дані та виконувати інші операції
			} else {
				echo "Помилка читання даних з сокета: " . socket_strerror(socket_last_error()) .
"\n";
			}
		}

		// Закриття сокета
		socket_close($socket);
	}

	public function secView($domain){
		/*"example.com"*/

		// Формуємо URL для запиту до API DNSViz
		$url = "https://dnsviz.net/api/v1/".urlencode($domain)."/json";

		// Виконуємо запит до API
		$response = file_get_contents($url);

		// Розшифровуємо JSON-відповідь
		$data = json_decode($response, true);

		// Перевіряємо наявність помилок DNSSEC
		if(isset($data['status']['secure'])){
			if($data['status']['secure'] == true){
				echo "DNSSEC встановлено для домена ".$domain;
			} else {
				echo "DNSSEC не встановлено для домена ".$domain;
			}
		} else {
			echo "Не вдалося отримати дані DNSSEC для домена ".$domain;
		}


		// Отримуємо DNS-записи для вказаного домену
		$dns_records = dns_get_record($domain, DNS_ALL);

		// Виводимо отримані DNS-записи
		echo "DNS-записи для домена ".$domain.":\n";

		$domain = "example.com"; // ваш домен для перевірки
	}
```

# Додаток Г. Лістинг Apriori.php

```
<?php
declare(strict_types=1);
namespace Phpml\Association;
use Phpml\Helper\Predictable;
use Phpml\Helper\Trainable;
class Apriori implements Associator
{
    use Trainable;
    use Predictable;

    public const ARRAY_KEY_ANTECEDENT = 'antecedent';

    public const ARRAY_KEY_CONFIDENCE = 'confidence';

    public const ARRAY_KEY_CONSEQUENT = 'consequent';

    public const ARRAY_KEY_SUPPORT = 'support';

    /**
     * Minimum relative probability of frequent transactions.
     *
     * @var float
     */
    private $confidence;

    /**
     * The large set contains frequent k-length item sets.
     *
     * @var mixed[][][]
     */
    private $large = [];

    /**
     * Minimum relative frequency of transactions.
     *
     * @var float
     */
    private $support;

    /**
     * The generated Apriori association rules.
     *
     * @var mixed[][]
     */
    private $rules = [];

    /**
     * Apriori constructor.
     */
    public function __construct(float $support = 0.0, float $confidence = 0.0)
    {
        $this->support = $support;
        $this->confidence = $confidence;
    }

    /**
```

```
 * Get all association rules which are generated for every k-length frequent item set.
 *
 * @return mixed[][]
 */
public function getRules(): array
{
    if (count($this->large) === 0) {
        $this->large = $this->apriori();
    }

    if (count($this->rules) > 0) {
        return $this->rules;
    }

    $this->rules = [];

    $this->generateAllRules();

    return $this->rules;
}

/**
 * Generates frequent item sets.
 *
 * @return mixed[][][]
 */
public function apriori(): array
{
    $L = [];

    $items = $this->frequent($this->items());
    for ($k = 1; isset($items[0]); ++$k) {
        $L[$k] = $items;
        $items = $this->frequent($this->candidates($items));
    }

    return $L;
}

/**
 * @param mixed[] $sample
 *
 * @return mixed[][]
 */
protected function predictSample(array $sample): array
{
    $predicts = array_values(array_filter($this->getRules(), function ($rule) use ($sample): bool {
        return $this->equals($rule[self::ARRAY_KEY_ANTECEDENT], $sample);
    }));

    return array_map(static function ($rule) {
        return $rule[self::ARRAY_KEY_CONSEQUENT];
    }, $predicts);
}

/**
 * Generate rules for each k-length frequent item set.
 */
private function generateAllRules(): void
```

```
{
    for ($k = 2; isset($this->large[$k]); ++$k) {
        foreach ($this->large[$k] as $frequent) {
            $this->generateRules($frequent);
        }
    }
}

/**
 * Generate confident rules for frequent item set.
 *
 * @param mixed[] $frequent
 */
private function generateRules(array $frequent): void
{
    foreach ($this->antecedents($frequent) as $antecedent) {
        $confidence = $this->confidence($frequent, $antecedent);
        if ($this->confidence <= $confidence) {
            $consequent = array_values(array_diff($frequent, $antecedent));
            $this->rules[] = [
                self::ARRAY_KEY_ANTECEDENT => $antecedent,
                self::ARRAY_KEY_CONSEQUENT => $consequent,
                self::ARRAY_KEY_SUPPORT => $this->support($frequent),
                self::ARRAY_KEY_CONFIDENCE => $confidence,
            ];
        }
    }
}

/**
 * Generates the power set for given item set $sample.
 *
 * @param mixed[] $sample
 *
 * @return mixed[][]
 */
private function powerSet(array $sample): array
{
    $results = [[]];
    foreach ($sample as $item) {
        foreach ($results as $combination) {
            $results[] = array_merge([$item], $combination);
        }
    }

    return $results;
}

/**
 * Generates all proper subsets for given set $sample without the empty set.
 *
 * @param mixed[] $sample
 *
 * @return mixed[][]
 */
private function antecedents(array $sample): array
{
    $cardinality = count($sample);
    $antecedents = $this->powerSet($sample);
```

```
    return array_filter($antecedents, static function ($antecedent) use ($cardinality): bool {
        return (count($antecedent) != $cardinality) && ($antecedent != []);
    });
}

/**
 * Calculates frequent k = 1 item sets.
 *
 * @return mixed[][]
 */
private function items(): array
{
    $items = [];

    foreach ($this->samples as $sample) {
        foreach ($sample as $item) {
            if (!in_array($item, $items, true)) {
                $items[] = $item;
            }
        }
    }

    return array_map(static function ($entry): array {
        return [$entry];
    }, $items);
}

/**
 * Returns frequent item sets only.
 *
 * @param mixed[][] $samples
 *
 * @return mixed[][]
 */
private function frequent(array $samples): array
{
    return array_values(array_filter($samples, function ($entry): bool {
        return $this->support($entry) >= $this->support;
    }));
}

/**
 * Calculates frequent k item sets, where count($samples) == $k - 1.
 *
 * @param mixed[][] $samples
 *
 * @return mixed[][]
 */
private function candidates(array $samples): array
{
    $candidates = [];

    foreach ($samples as $p) {
        foreach ($samples as $q) {
            if (count(array_merge(array_diff($p, $q), array_diff($q, $p))) != 2) {
                continue;
            }
```

```
            $candidate = array_values(array_unique(array_merge($p, $q)));

            if ($this->contains($candidates, $candidate)) {
                continue;
            }

            foreach ($this->samples as $sample) {
                if ($this->subset($sample, $candidate)) {
                    $candidates[] = $candidate;

                    continue 2;
                }
            }
        }
    }

    return $candidates;
}

/**
 * Calculates confidence for $set. Confidence is the relative amount of sets containing $subset which also contain
 * $set.
 *
 * @param mixed[] $set
 * @param mixed[] $subset
 */
private function confidence(array $set, array $subset): float
{
    return $this->support($set) / $this->support($subset);
}

/**
 * Calculates support for item set $sample. Support is the relative amount of sets containing $sample in the data
 * pool.
 *
 * @see \Phpml\Association\Apriori::samples
 *
 * @param mixed[] $sample
 */
private function support(array $sample): float
{
    return $this->frequency($sample) / count($this->samples);
}

/**
 * Counts occurrences of $sample as subset in data pool.
 *
 * @see \Phpml\Association\Apriori::samples
 *
 * @param mixed[] $sample
 */
private function frequency(array $sample): int
{
    return count(array_filter($this->samples, function ($entry) use ($sample): bool {
        return $this->subset($entry, $sample);
    }));
}

/**
```

```
     * Returns true if set is an element of system.
     *
     * @see \Phpml\Association\Apriori::equals()
     *
     * @param mixed[][] $system
     * @param mixed[]   $set
     */
    private function contains(array $system, array $set): bool
    {
        return (bool) array_filter($system, function ($entry) use ($set): bool {
            return $this->equals($entry, $set);
        });
    }

    /**
     * Returns true if subset is a (proper) subset of set by its items string representation.
     *
     * @param mixed[] $set
     * @param mixed[] $subset
     */
    private function subset(array $set, array $subset): bool
    {
        return count(array_diff($subset, array_intersect($subset, $set))) === 0;
    }
    private function equals(array $set1, array $set2): bool
    {
        return array_diff($set1, $set2) == array_diff($set2, $set1);
    }
}
```

# Додаток Ґ. Лістинг secTraf.php

```
<div class="wrap">

<div class="menu">
	<a href="/admin/dnsTraf">DNS-трафік</a>
	<a href="/admin/secTraf">Записи DNSSEC</a>
	<a href="/admin/stat">Статистика поведінки</a>
	<a href="/admin/limit">Прийняті обмеження</a>
	<a href="/admin/message">Повідомлення</a>
	<a href="/admin/profile">Профіль</a>
	<a href="/welcome/logout">Вихід</a>
</div>

<div class="main">
	<div class="left">
		<span class="big">Навчання</span>
		<a href="/neuron/">Оновити</a>

		<a href="/neuron/loadDns">Додати DNS</a>
		<a href="/neuron/loadDnsHistory">Дані DNS</a>

		<a href="/neuron/loadSec">Додати DNSSEC</a>
		<a href="/neuron/loadSecHistory">Дані DNSSEC</a>
	</div>
	<div class="right">
		<span class="big">Записи DNSSEC</span>

			<div class="center" id="drow"></div>

			<?php if ( !empty($sec) ): ?>
				<div class="data-list">
					<div class="item">
						<span class="hol">Домен</span>
						<span class="small">class</span>
						<span class="small">ttl</span>
						<span class="small">type</span>
						<span class="hol">info</span>
						<span class="hol">Дата</span>
						<span class="small"></span>
					</div>

					<?php foreach ($sec as $item): ?>
						<div class="item small-font">
							<span class="hol"><?=$item->domain?></span>
							<span class="small"><?=$item->class?></span>
							<span class="small"><?=$item->ttl?></span>
							<span class="small"><?=$item->type?></span>
							<span class="hol"><?=$item->info?></span>
							<span class="hol"><?=$item->date?></span>
							<span class="small">
								<?php if ( $item->warning == 1 ): ?>
									&#128681;
								<?php endif; ?>
							</span>
						</div>
					<?php endforeach; ?>
				</div>
```

```
                                    <?php endif; ?>

                        </div>
                </div>


        </div>



        <script src="https://cdn.anychart.com/releases/v8/js/anychart-
base.min.js?hcode=c11e6e3cfefb406e8ce8d99fa8368d33"></script>
        <script src="https://cdn.anychart.com/releases/v8/js/anychart-
ui.min.js?hcode=c11e6e3cfefb406e8ce8d99fa8368d33"></script>
        <script src="https://cdn.anychart.com/releases/v8/js/anychart-
exports.min.js?hcode=c11e6e3cfefb406e8ce8d99fa8368d33"></script>
        <link href="https://cdn.anychart.com/releases/v8/css/anychart-
ui.min.css?hcode=c11e6e3cfefb406e8ce8d99fa8368d33" type="text/css" rel="stylesheet">
        <link href="https://cdn.anychart.com/releases/v8/fonts/css/anychart-
font.min.css?hcode=c11e6e3cfefb406e8ce8d99fa8368d33" type="text/css" rel="stylesheet">

        <script>
            anychartPie(<?=$json?>, 'DNSSEC', 'drow');
        </script>
```

# Додаток Д. Ілюстративний матеріал

## Розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів

Виконав ст. групи 1КІТС-20б Слюсар Д.Ю.
Керівник: к.т.н., доцент кафедри МБІС Грицак А.В.

## Актуальність:

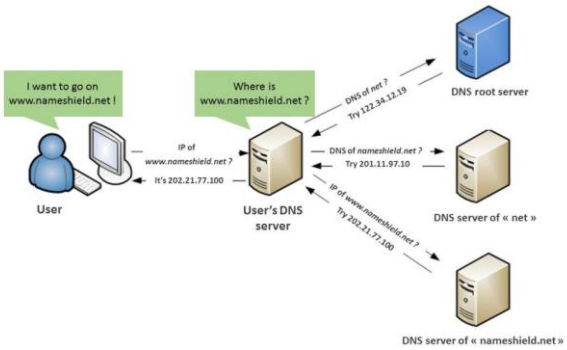
Постійність та еволюція атак DNS Cache Poisoning підкреслюють нагальну потребу у вдосконалених стратегіях виявлення та усунення наслідків. Традиційні заходи безпеки, хоча і є необхідними, виявилися недостатніми проти деяких складних атак, що призвело до нагальної потреби в інноваційних рішеннях. Науковою новизною роботи є розробка та впровадження комплексного рішення на мові програмування PHP для захисту DNS серверів від атак "отруєння кешу". Розроблена система допомагає підвищити безпеку і надійність DNS серверів, запобігаючи атакам і забезпечуючи автоматичне повідомлення адміністратору про події на сервері.

## Мета:

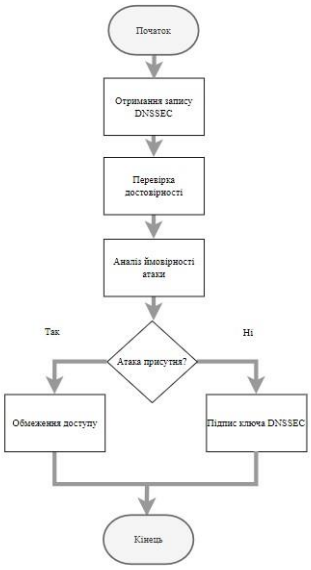
Метою роботи є розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" з використанням машинного навчання для виявлення аномальних патернів в DNS-трафіку. Метою цього дослідження є не тільки усунення безпосередніх загроз, пов'язаних з отруєнням кешу DNS, але й внесок у більш широку сферу кібербезпеки шляхом підвищення стійкості інфраструктури DNS за допомогою інтелектуальних, адаптивних засобів захисту.

# Схема роботи DNS

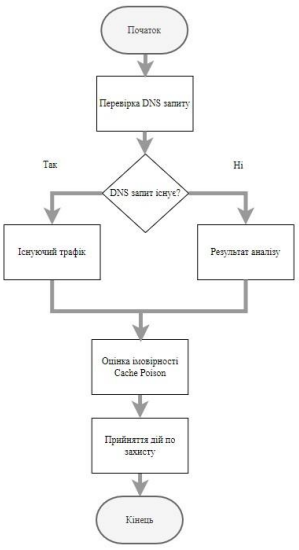
Система доменних імен (DNS) має фундаментальне значення для роботи Інтернету, оскільки вона перетворює доменні імена, що читаються людиною, на машинні IP-адреси. Отруєння кешу DNS, також відоме як підміна DNS, - це процес введення неправдивої інформації в кеш DNS-розпізнавача, щоб він повертав неправильні IP-адреси у відповідь на запити користувачів. Це може перенаправляти користувачів на шкідливі веб-сайти та сприяти фішинговим атакам, розповсюдженню шкідливого програмного забезпечення і витоку даних. Складність цих атак зумовлена вразливістю протоколу DNS та використанням моделі довіри, яка працює в DNS, що робить їх особливо складними для виявлення та усунення.




# Проєктування архітектури системи

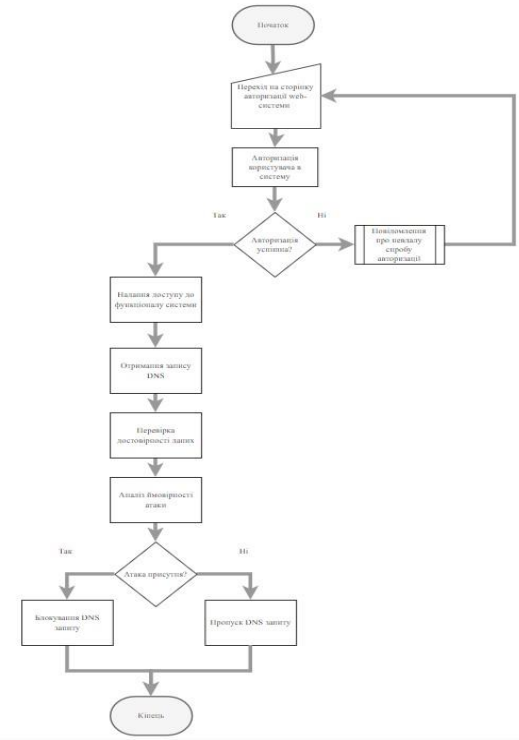



Алгоритм роботи з DNSSEC розробленої системи

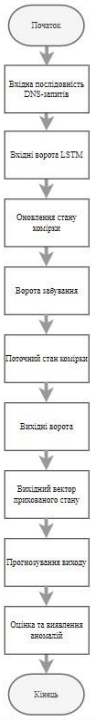



Алгоритм роботи з DNS розробленої системи

# Схема взаємодії користувача з розробленою системою




Блок-схема LSTM-архітектури з роботою воріт розробленої системи




Блок-схема процесу навчання LSTM нейронної мережі розробленої системи


Блок-схема LSTM-архітектури розробленої системи

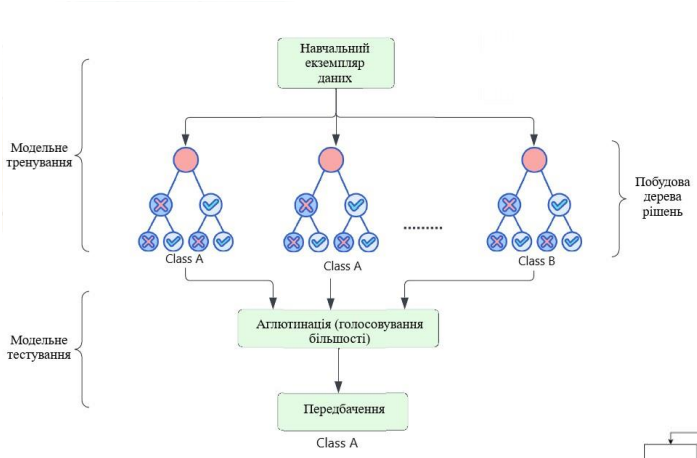
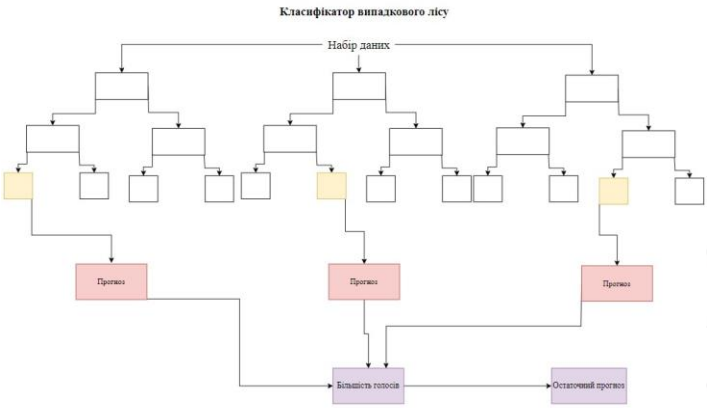
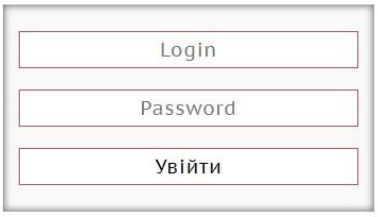



Схема алгоритму випадкового лісу

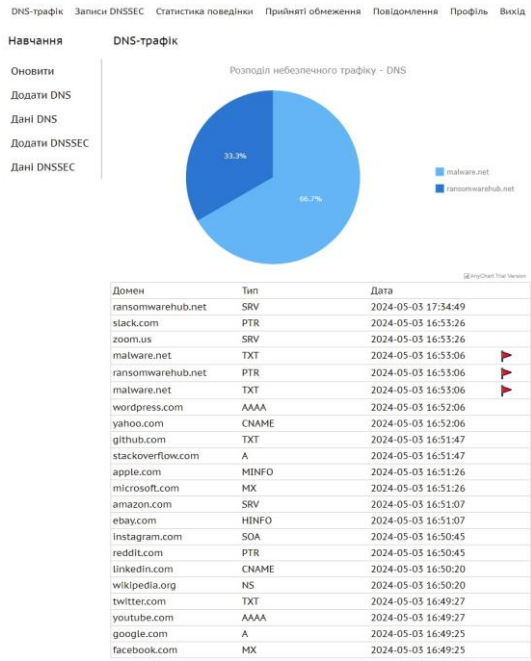
Діаграма класифікації випадкового лісу




# Структура інтерфейсу




Сторінка авторизації адміністратора

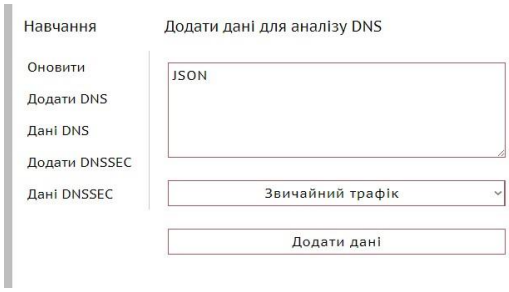



Головне вікно системи DNS-трафіку

# Завантаження даних для машинного навчання

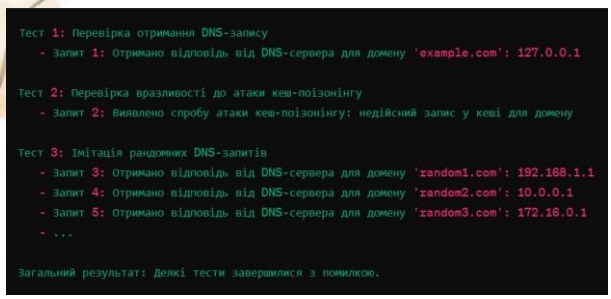

Вигляд сторінки машинного навчання


Вигляд сторінки даних для машинного навчання DNS

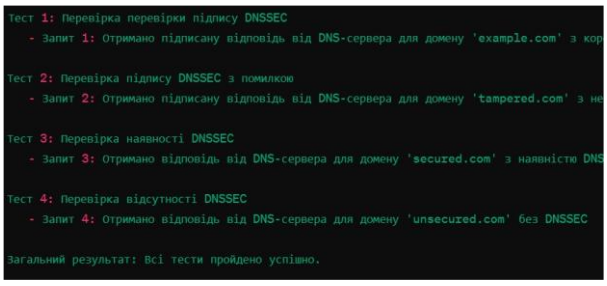



Вигляд сторінки додавання даних аналізу машинного навчання

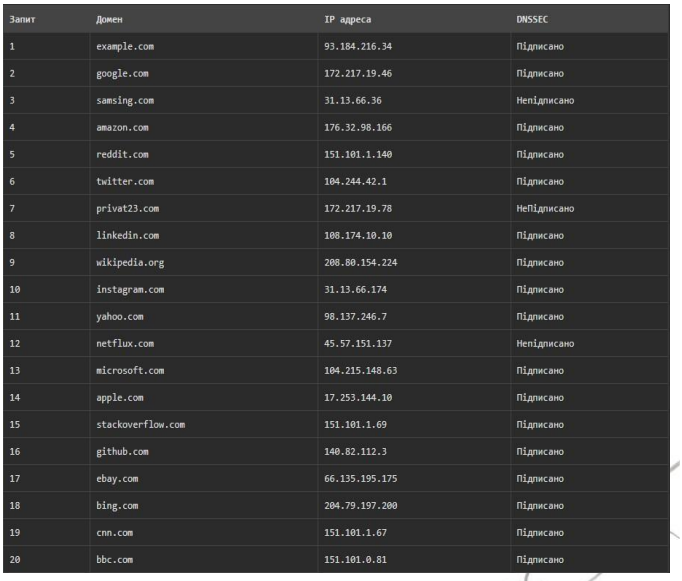
# Сценарій тестування та результати




DNS тест




DNSSEC тест



| Запит | Домен | IP адреса | DNSSEC |
|---|---|---|---|
| 1 | example.com | 93.184.216.34 | Підписано |
| 2 | google.com | 172.217.19.46 | Підписано |
| 3 | samsing.com | 31.13.66.36 | Непідписано |
| 4 | amazon.com | 176.32.98.166 | Підписано |
| 5 | reddit.com | 151.101.1.140 | Підписано |
| 6 | twitter.com | 104.244.42.1 | Підписано |
| 7 | privat23.com | 172.217.19.78 | НеПідписано |
| 8 | linkedin.com | 108.174.10.10 | Підписано |
| 9 | wikipedia.org | 208.80.154.224 | Підписано |
| 10 | instagram.com | 31.13.66.174 | Підписано |
| 11 | yahoo.com | 98.137.246.7 | Підписано |
| 12 | netflux.com | 45.57.151.137 | Непідписано |
| 13 | microsoft.com | 104.215.148.63 | Підписано |
| 14 | apple.com | 17.253.144.10 | Підписано |
| 15 | stackoverflow.com | 151.101.1.69 | Підписано |
| 16 | github.com | 140.82.112.3 | Підписано |
| 17 | ebay.com | 66.135.195.175 | Підписано |
| 18 | bing.com | 204.79.197.200 | Підписано |
| 19 | cnn.com | 151.101.1.67 | Підписано |
| 20 | bbc.com | 151.101.0.81 | Підписано |

Діалогове вікно інформації про запити

# Висновки

В даній дипломній роботі вдалося виконати усі поставлені задачі, а саме аналіз проблеми атак "отруєння кешу DNS" та існуючих методів їх виявлення і протидії; вивчення підходів машинного навчання для виявлення аномалій в мережевому трафіку; розробка навчальної моделі для класифікації DNS-запитів та ідентифікації атак "отруєння кешу"; створення системи моніторингу та аналізу DNS-трафіку з використанням навченої моделі; тестування та оцінка ефективності розробленої інтелектуальної системи. Таким чином, була здійснена розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів.

Розроблений інтерфейс системи аналізу DNS-трафіку є інтуїтивно зрозумілим та досить простим у використанні для користувача.

Отже, аналізуючи отримані результати роботи, можна вважати, що розроблена система представляє собою потужний інструмент для виявлення та запобігання атакам на DNS-системи, який може бути успішно впроваджений у різні організації та мережеві інфраструктури, сприяючи підвищенню загального рівня кібербезпеки. Адже дана інтелектуальна система, яка забезпечує ефективний захист від атак "отруєння кешу DNS" за допомогою машинного навчання, готова до використання в реальних умовах та сприяє підвищенню безпеки мережевої інфраструктури.

# Дякую за увагу

## Додаток Е. Протокол перевірки на антиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка інтелектуальної системи для виявлення та реагування на атаки "отруєння кешу DNS" (DNS Cache Poisoning) з використанням машинного навчання для виявлення аномальних патернів

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

**Показники звіту подібності Unicheck**

Оригінальність 98 % Схожість 2 %

Аналіз звіту подібності (відмітити потрібне):
**1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Слюсар Д.Ю.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Грицак А.В.
(підпис) (прізвище, ініціали