НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

«На правах рукопису»

УДК _____

«До захисту допущено»

Завідувач кафедри
_____Олександр КОВАЛЬ
«____» _____ 202__ р.

Магістерська дисертація

За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

Спеціальність 121 Інженерія програмного забезпечення

на тему: Методика забезпечення функціональної стійкості кіберзахисту систем керування базами даних

Виконала студентка 2 курсу, групи ТВ-42мп

Плачинда Маргарита Володимирівна _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник

к.т.н доцент Шуклін Герман Вікторович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент

_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
….. (підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти другий, магістерський
За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
Спеціальності 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
«___» _____ 202__ р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТЦІ

Плачинді Маргариті Володимирівні

1. Тема дисертації: «Методика забезпечення функціональної стійкості кіберзахисту систем керування базами даних»
Науковий керівник: Шуклін Герман Вікторович, к.т.н., доцент
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)
затверджений наказом вищого навчального закладу від "03"11 2025р. № 4779-с

2. Строк подання студентом роботи: «17» грудня 2025 р.

3. Об'єкт дослідження: Процеси забезпечення кіберзахисту інформаційних ресурсів у системах керування базами даних в умовах деструктивних впливів.

3. Предмет дослідження: Методи, моделі та засоби забезпечення функціональної стійкості кіберзахисту систем керування базами даних під час реалізації кіберзагроз, атак та відмов

4. Перелік питань, які потрібно розробити: Провести аналіз сучасних загроз кібербезпеці систем керування базами даних та методів їх реалізації. Дослідити

існуючі підходи та засоби кіберзахисту СКБД і визначити їх обмеження з погляду функціональної стійкості. Сформувати понятійно-категоріальний апарат функціональної стійкості кіберзахисту СКБД. Розробити модель функціонування системи кіберзахисту СКБД в умовах дестабілізуючих впливів. Розробити методику забезпечення функціональної стійкості кіберзахисту систем керування базами даних. Провести експериментальну оцінку ефективності запропонованої методики та порівняти її з існуючими підходами. Надати практичні рекомендації щодо впровадження методики в реальних інформаційних системах.

5. Перелік ілюстративного матеріалу: Архітектура системи, концептуальна модель системи, діаграма прецедентів взаємодії користувача з системою кіберзахисту СКБД, діаграма класів, діаграма компонентів, скріншоти головного екрану, оцінка якості ML-моделі, скріншоти аналітики атак, скріншоти підключення БД, динамічний графік.

6. Дата видачі завдання «01» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів магістерської дисертації	Примітки
1	Отримання завдання	01.10.2024	Виконано
2	Дослідження предметної області	01.10 - 01.11	Виконано
3	Постановка вимог до Створення методики	01.11 - 15.11	Виконано
4	Дослідження існуючих рішень	15.11 - 01.12.2024	Виконано
5	Розробка програмного продукту	05.03.2025 - 07.10	Виконано
6	Тестування	07.10 - 11.11	Виконано
7	Захист програмного продукту	17.10 – 21.10	Виконано
8	Оформлення магістерської дисертації	22.10 – 20.11	Виконано

9	Передзахист	24.11 - 28.11	Виконано
10	Захист	23.12.2025	

Студент _____ Маргарита ПЛАЧИНДА
(підпис) (Ім’я ПРІЗВИЩЕ)

Науковий керівник _____ Герман ШУКЛІН
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи: робота містить 100 сторінок, 10 рисунків, 2 додатки та 15 посилань.

У даній роботі опрацьовані та реалізовані алгоритми забезпечення функціональної стійкості кіберзахисту систем керування базами даних, виконано аналіз загроз і вразливостей, розроблено методи виявлення та протидії кібератакам, а також здійснено тестування та практичне використання запропонованих рішень у програмному прототипі системи кіберзахисту СКБД.

Метою роботи є створення методики забезпечення функціональної стійкості кіберзахисту систем керування базами даних, а саме:

- проаналізувати існуючі системи та методи кіберзахисту СКБД;
- дослідити сучасні загрози та вразливості баз даних і визначити основні фактори, що впливають на їх функціональну стійкість;
- розробити програмний засіб для моніторингу, виявлення та протидії кібератакам на рівні СКБД.

Практичне значення одержаних результатів полягає у створенні програмного прототипу системи кіберзахисту баз даних та методики забезпечення її функціональної стійкості. Розроблені алгоритми дозволяють здійснювати аналіз SQL-запитів, виявлення підозрілої активності, реєстрацію атак у журналі подій та оцінку рівня загроз у системі керування базами даних.

Для досягнення поставленої мети було спроєктовано архітектуру системи, реалізовано модулі моніторингу, аналізу та обробки атак, а також створено веб-застосунок, який надає можливість перевіряти запити на наявність загроз, аналізувати історію атак, здійснювати оцінку стану безпеки СКБД та візуалізувати результати роботи системи.

Ключові слова: кіберзахист, система керування базами даних, СКБД, функціональна стійкість, кібератака, SQL-ін’єкції.

ABSTRACT

Structure and scope of the thesis: the work consists of 100 pages, 10 figures, 2 appendices, and 15 references.

In this thesis, the algorithms for ensuring the functional resilience of cybersecurity of database management systems are developed and implemented. The analysis of threats and vulnerabilities is carried out, methods for detecting and counteracting cyberattacks are developed, and testing and practical application of the proposed solutions in a software prototype of a DBMS cybersecurity system are performed.

The purpose of the work is to develop a methodology for ensuring the functional resilience of cybersecurity of database management systems, namely:

- to analyze existing systems and methods of cybersecurity of DBMS;
- to investigate modern threats and vulnerabilities of databases and identify the main factors affecting their functional resilience;
- to develop a software tool for monitoring, detecting, and counteracting cyberattacks at the DBMS level.

The practical significance of the obtained results lies in the development of a software prototype of a database cybersecurity system and a methodology for ensuring its functional resilience. The developed algorithms enable the analysis of SQL queries, detection of suspicious activity, registration of attacks in the event log, and assessment of the threat level in database management systems.

To achieve the set goal, the system architecture was designed, monitoring, analysis, and attack processing modules were implemented, and a web application was developed that enables checking queries for threats, analyzing the attack history, assessing the security state of the DBMS, and visualizing the system performance results.

Keywords: cybersecurity, database management system, DBMS, functional resilience, cyberattack, SQL injection.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

СКБД — система керування базами даних.

SQL — Structured Query Language, мова запитів до реляційних баз даних.

SQL-ін'єкція — тип кібератаки, що ґрунтується на впровадженні шкідливих SQL-запитів у вразливі інтерфейси СКБД.

DoS-атака (Denial of Service) — атака типу «відмова в обслуговуванні», спрямована на перевантаження системи та порушення її доступності.

ML (Machine Learning) — машинне навчання, напрям штучного інтелекту, що використовується для аналізу та класифікації SQL-запитів.

TF-IDF (Term Frequency – Inverse Document Frequency) — статистичний метод векторизації тексту, застосований для навчання ML-моделі.

DBMS (Database Management System) — англомовний відповідник терміну «система керування базами даних».

API (Application Programming Interface) — програмний інтерфейс для взаємодії між компонентами системи.

HTTP (HyperText Transfer Protocol) — протокол передавання даних у веб-системах.

REST (Representational State Transfer) — програмна архітектура побудови веб-сервісів.

JSON (JavaScript Object Notation) — формат обміну структурованими даними між клієнтською та серверною частинами застосунку.

ORM (Object-Relational Mapping) — технологія об'єктно-реляційного відображення (реалізована у роботі через SQLAlchemy).

ЗМІСТ

ВСТУП

У сучасних умовах глобальної цифровізації та зростання ролі інформаційних технологій у всіх сферах суспільного життя питання кібербезпеки набуває особливої ваги. Одним із ключових елементів інформаційної інфраструктури будь-якої організації є системи керування базами даних (СКБД), які забезпечують зберігання, обробку, структурування та надання доступу до великих обсягів інформації. Вони відіграють критично важливу роль у функціонуванні державних установ, бізнес-структур, фінансових організацій та оборонно-промислового комплексу.

Методика забезпечення функціональної стійкості кіберзахисту СКБД є надзвичайно актуальною в умовах постійного ускладнення кіберзагроз. Стрімкий розвиток інформаційних технологій, розширення використання хмарних сервісів, розподілених систем та мобільних платформ призводять до того, що обсяг і значущість даних, які зберігаються в базах даних, постійно зростають. Це робить СКБД однією з головних цілей для кіберзлочинців, які прагнуть отримати несанкціонований доступ до критично важливої інформації, викрасти, змінити або знищити її. Відсутність ефективних механізмів кіберзахисту може спричинити витоки конфіденційних відомостей, фінансові збитки, репутаційні втрати, порушення безперервності роботи інформаційних систем і навіть дестабілізацію діяльності організацій.

З кожним роком спектр кіберзагроз, спрямованих на бази даних, розширюється. Серед найбільш поширених атак можна виокремити SQL-ін’єкції, несанкціонований доступ до систем, комплексні DoS-атаки (Denial of Service), ескалацію привілеїв (Escalation of Privileges), маніпуляції даними та інші форми кіберзлочинної активності. Їх застосування здатне призвести до масштабних інцидентів, включно з викраденням чутливої інформації, пошкодженням баз даних,

блокуванням доступу до інформаційних ресурсів та порушенням цілісності інформаційної інфраструктури. У зв’язку з цим забезпечення стійкості СКБД до кіберзагроз потребує не фрагментарних, а комплексних та науково обґрунтованих підходів, які враховують сучасний рівень розвитку технологій та різноманіття методів атак.

Особливого значення ця проблема набуває у військовій та оборонній сферах, де інформаційні системи містять стратегічно важливі дані — плани операцій, інформацію про ресурси, військову інфраструктуру, логістичні ланцюги тощо. Будь-яке порушення цілісності чи конфіденційності таких даних може мати серйозні наслідки для національної безпеки. В умовах сучасних геополітичних конфліктів, коли інформаційні війни та кібератаки є невід’ємною складовою бойових дій, питання кіберзахисту військових інформаційних систем набуває стратегічного характеру.

Отже, актуальність дослідження зумовлена необхідністю розроблення та вдосконалення ефективних методик кіберзахисту систем керування базами даних, здатних забезпечити їх функціональну стійкість в умовах швидко змінюваного цифрового середовища та зростаючої складності кіберзагроз. Такі методики повинні враховувати як технічні, так і організаційні аспекти захисту, бути гнучкими, адаптивними та придатними до застосування у критично важливих сферах, зокрема в оборонному секторі.

Разом із тим слід зазначити, що на практиці значна частина наявних засобів захисту СКБД орієнтована переважно на окремі рівні безпеки та не завжди забезпечує комплексний підхід до протидії сучасним кібератакам. Це призводить до зниження ефективності систем захисту в умовах комбінованих та багатоетапних атак. У зв’язку з цим усе більшої актуальності набуває питання не лише захисту даних, а й забезпечення саме функціональної стійкості СКБД як здатності системи зберігати працездатність у процесі та після кібератак.

Функціональна стійкість кіберзахисту систем керування базами даних

передбачає здатність системи своєчасно виявляти загрози, адаптуватися до змін характеру атак, локалізувати їх наслідки та забезпечувати відновлення нормального режиму роботи з мінімальними втратами для інформаційної інфраструктури. Реалізація такої стійкості потребує застосування сучасних методів аналізу, моніторингу та автоматизованого реагування на інциденти інформаційної безпеки.

Важливу роль у забезпеченні функціональної стійкості відіграє використання інтелектуальних методів аналізу даних, зокрема алгоритмів машинного навчання, які дозволяють здійснювати виявлення аномальної активності, прогнозувати можливі загрози та підвищувати ефективність систем кіберзахисту. Поєднання класичних методів безпеки з інтелектуальними технологіями відкриває нові можливості для створення адаптивних і стійких до сучасних кібератак систем захисту баз даних.

Крім того, забезпечення функціональної стійкості кіберзахисту СКБД тісно пов’язане з необхідністю своєчасного виявлення атак ще на ранніх етапах їх реалізації. Це дозволяє мінімізувати потенційні збитки та запобігти розповсюдженню шкідливого впливу на інші компоненти інформаційної системи. Особливе значення при цьому має організація безперервного моніторингу подій безпеки, аналізу журналів доступу та журналів атак, а також використання механізмів кореляції подій.

Сучасні кібератаки дедалі частіше мають складний багаторівневий характер і поєднують у собі декілька технік проникнення, маскування та обходу засобів захисту. Саме тому ефективні системи кіберзахисту повинні бути здатними не лише реагувати на вже відомі загрози, а й виявляти нові, раніше невідомі типи атак. У цьому контексті важливу роль відіграють методи поведінкового аналізу та машинного навчання, які дозволяють формувати профілі нормальної роботи СКБД та на їх основі визначати аномальні відхилення.

Окрему увагу слід приділити проблемі внутрішніх загроз, які виникають

унаслідок неправомірних дій користувачів із легітимним доступом до баз даних. Такі загрози є особливо небезпечними, оскільки традиційні засоби захисту не завжди здатні своєчасно їх ідентифікувати. Саме тому сучасні методики кіберзахисту мають враховувати як зовнішні, так і внутрішні фактори ризику.

З огляду на зазначене, розробка методики забезпечення функціональної стійкості кіберзахисту систем керування базами даних є своєчасним і важливим науково-практичним завданням, розв'язання якого сприятиме підвищенню рівня інформаційної безпеки, забезпеченню стабільної роботи СКБД та зниженню ризиків виникнення кіберінцидентів у державному секторі, бізнес-середовищі та оборонній галузі.

У зв'язку з наведеним вище, **метою даної магістерської роботи** є розробка методики забезпечення функціональної стійкості кіберзахисту систем керування базами даних на основі аналізу сучасних загроз, вразливостей та методів протидії кібератакам**.** Досягнення поставленої мети передбачає комплексне поєднання теоретичних досліджень і практичної реалізації програмних засобів для моніторингу, виявлення та протидії атакам на рівні СКБД.

РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ МЕТИ Й ОСНОВНИХ ЗАВДАНЬ ПРАКТИКИ

1.1. Напрями досліджень у галузі СКБД

Сучасний цифровий простір є надзвичайно уразливим до кіберзагроз, що стають дедалі складнішими і більш винахідливими. У межах досліджень кіберзахисту систем керування базами даних (СКБД) особлива увага приділяється аналізу та протидії цим загрозам, які постійно еволюціонують і охоплюють широкий спектр атак. Однією з найбільш поширених є SQL-ін'єкція, при якій зловмисники вводять шкідливі SQL-команди в інтерфейси СКБД для викрадення або модифікації даних.

Розглянемо декілька прикладів:

У 2014 році було здійснено масштабну SQL-ін'єкцію на сервери Yahoo, внаслідок якої було викрадено понад 500 мільйонів особистих акаунтів користувачів. Ці акаунти включали в себе ім'я користувача, адреси електронної пошти, дати народження, зашифровані паролі та іншу інформацію.

Наслідки:

• Викрадення даних 500 мільйонів користувачів, що включали чутливу інформацію.

• Витік паролів та доступ до конфіденційних акаунтів.

У 2015 році британський телекомунікаційний оператор TalkTalk став жертвою масштабної SQL-ін'єкції. Зловмисники використовували вразливість на веб-сайті компанії для здійснення SQL-ін'єкції, що дозволило їм отримати доступ до баз даних клієнтів.

Наслідки:

• Зловмисники змогли викрасти особисті дані понад 157 000 користувачів, включаючи імена, адреси, дати народження та номери телефонів.

• Було отримано дані 15 000 рахунків клієнтів, що містили банківські реквізити.

У 2017 році компанія Equifax, один із найбільших кредитних бюро в США, постраждала від серйозної кібератаки через SQL-ін'єкцію. Уразливість у їхній веб-системі дозволила зловмисникам отримати доступ до конфіденційної інформації понад 145 мільйонів людей, включаючи номери соціального страхування, дати народження та адреси.

Наслідки:

• Викрадення особистих даних 145 мільйонів користувачів, включаючи чутливу інформацію про кредитні карти.

• Загроза для національної безпеки та економічної безпеки внаслідок викрадення критично важливої інформації.

Іншим важливим напрямом досліджень є аналіз несанкціонованого доступу до бази даних, який часто відбувається через недостатньо надійні механізми аутентифікації або вразливі точки доступу. Такими можуть бути помилки в налаштуваннях паролів або відсутність багаторівневих засобів автентифікації.

Розглянемо декілька прикладів:

У 2010 році зловмисники змогли отримати несанкціонований доступ до системи баз даних SAP, яка використовується багатьма великими компаніями для управління бізнес-процесами. Виявлення слабкості в механізмах автентифікації дозволило хакерам успішно обійти контроль доступу та отримати повний доступ до усіх конфіденційних даних.

Наслідки:

• Зловмисники змогли отримати доступ до чутливої фінансової інформації, що використовувалася для проведення транзакцій.

• Викрадення корпоративної інформації, що призвело до фінансових втрат і репутаційної шкоди.

У 2017 році компанії, які використовували Apache Solr, популярний сервер для пошуку та аналізу великих обсягів даних, стали жертвами атак через несанкціонований доступ до баз даних. Веб-додатки, які використовували Solr для індексації та пошуку даних, мали вразливості, що дозволяли зловмисникам виконувати шкідливі запити до бази даних і обробляти дані без авторизації. Наслідки:

• Атака дозволила отримати доступ до великої кількості і конфіденційної інформації, яка не була належним чином захищена.

• Викрадення особистих даних користувачів або конфіденційної комерційної інформації.

У 2019 році виявлено серйозну уразливість в системах баз даних Oracle, яка дозволила зловмисникам отримати несанкціонований доступ до даних. Атака полягала у використанні CVE-2019-2725, що дозволяло атакуючим віддалено виконувати код і отримувати доступ до критичних даних, навіть якщо вони не мали облікових даних для входу.

Наслідки:

•Витік конфіденційних даних клієнтів і персоналу.

• Можливість виконання зловмисних операцій на базах даних, таких як видалення або модифікація інформації.

Атаки через несанкціонований доступ до баз даних можуть мати серйозні наслідки, включаючи викрадення конфіденційної інформації, фінансові втрати, репутаційні шкоди та порушення законодавства. Для запобігання таким атакам важливо здійснювати постійний моніторинг систем безпеки, застосовувати сильні механізми автентифікації, шифрувати дані та регулярно оновлювати програмне забезпечення для усунення вразливостей.

Метою цієї дисертаційної роботи є розробка та вдосконалення методики

забезпечення функціональної стійкості кіберзахисту систем керування базами даних (СКБД). Вибір цієї теми обумовлений зростаючою необхідністю створення спеціалізованих і адаптованих механізмів захисту, які враховують специфіку функціонування баз даних та здатні ефективно протистояти різноманітним, постійно змінюваним кіберзагрозам. Сучасні методи кіберзахисту повинні бути гнучкими й адаптивними, оскільки технології розвиваються дуже швидко, а разом із ними й еволюціонують методи атак.

З огляду на це, в роботі пропонується такий підхід до проєктування, реалізації та тестування систем безпеки для баз даних, що дозволить врахувати всі особливості цих систем та забезпечити їх безперебійну функціональність навіть в умовах складних та швидко змінюваних кіберзагроз. Основну увагу буде приділено створенню таких інструментів та методик, які гарантують захист баз даних не лише від типових атак, але й від новітніх технік, що використовуються в умовах гібридних та кібервійськових конфліктів.

Розроблені механізми повинні враховувати важливість збереження конфіденційності, цілісності та доступності даних у таких критичних сферах, як оборона, національна безпека та стратегічне управління. Крім того, у рамках роботи буде розглянуто питання про побудову надійних систем моніторингу та реагування на загрози, а також механізмів тестування й оцінки стійкості до кіберзагроз, що дозволить забезпечити безперервність роботи цих систем в умовах реальних атак.
Таким чином, основним завданням цієї дисертації є розробка цілісної методики забезпечення кіберзахисту СКБД, що дозволить суттєво підвищити їх стійкість до атак і забезпечити безпеку інформації в умовах сучасних загроз, зокрема в критично важливих галузях, пов’язаних із національною безпекою та військовими операціями. Отже, методики забезпечення функціональної стійкості кіберзахисту СКБД є актуальним напрямом наукових досліджень, які дозволяють удосконалити технології кібербезпеки та створити нові стратегії для боротьби з новими, більш

складними загрозами. Розробка таких методик є необхідною для забезпечення належного рівня безпеки даних і для підвищення надійності СКБД, що є критично важливим для організацій усіх типів в умовах зростаючих кіберзагроз.

1.2. Постановка завдання

Основною метою дослідження є розробка методики забезпечення функціональної стійкості кіберзахисту систем керування базами даних (СКБД), яка дозволить підвищити їхню надійність і захищеність від сучасних кіберзагроз. Для досягнення цієї мети необхідно вирішити низку завдань, спрямованих на підвищення рівня безпеки СКБД в умовах новітніх загроз, зокрема з фокусом на такі аспекти, як контроль доступу, шифрування, аудит, оновлення, резервне копіювання та автоматизація процесів. Завдання дослідження включають наступні пункти, які ми детально розглянемо.

Постановка завдань:

1. Аналіз існуючих методів кіберзахисту СКБД.
2. Проєктування архітектури веб-сервісу
3. Розробка механізмів парсингу та фільтрації SQL-запитів
4. Інтеграція машинного навчання для оцінки підозрілих запитів
5. Безпечне виконання SQL-запитів
6. Логування, аудит та моніторинг
7. Побудова тестової інфраструктури та сценаріїв тестування
8. Оцінка ефективності запропонованих рішень

Розглянемо кожен пункт більш детально.

Аналіз існуючих методів захисту баз даних

- Проведено аналіз сучасних технологій і методів захисту СУБД, зокрема механізмів контролю доступу, шифрування даних, виявлення SQL-ін’єкцій та запобігання несанкціонованим запитам. Оцінено ефективність цих підходів.

Проєктування архітектури веб-сервісу

- Розроблено багаторівневу архітектуру системи, що забезпечує розділення логіки між модулями: автентифікації, логування, аналітики, обробки запитів та основного API. Передбачено гнучкість структури для можливості масштабування та підключення додаткових компонентів безпеки.

Розробка механізмів парсингу та фільтрації SQL-запитів

- Реалізовано механізм первинного аналізу SQL-запитів використанням правил парсингу, що дозволяє виявляти небезпечні ключові слова, команди та токени. Забезпечено можливість формування пояснення причин блокування запиту для подальшого аудиту.

Інтеграція алгоритмів машинного навчання

- Для підвищення точності виявлення потенційно шкідливих запитів інтегровано модуль машинного навчання. Класифікатор, навчений на основі TF-IDF та логістичної регресії, оцінює рівень ризику запитів і приймає рішення про їхнє виконання або блокування. Передбачено адаптивну порогову систему для визначення рівня загрози.

Безпечне виконання SQL-запитів

- Реалізовано механізм безпечного виконання SQL-запитів за допомогою транзакцій SQLAlchemy у режимі sandbox, що дозволяє аналізувати результати без внесення змін у базу даних. Забезпечено додатковий рівень захисту від зловмисних дій.

Логування, аудит та моніторинг системи

- Організовано систему аудиту, що фіксує всі запити, їхній статус, джерело та рівень ризику. Логування дозволяє проводити подальший аналіз активності користувачів і потенційних атак, а також формувати звіти про безпеку.

Побудова тестового середовища та перевірка працездатності

- Створено набір тестових сценаріїв, що охоплюють безпечні, шкідливі та граничні SQL-запити. Проведено комплексне тестування системи, спрямоване на

перевірку ефективності алгоритмів фільтрації, логування та ML-модуля.

Оцінка ефективності розроблених рішень

- Виконано експериментальну оцінку ефективності роботи системи на основі порівняння з традиційними методами захисту баз даних. Виявлено переваги комплексного підходу, що поєднує правила парсингу, машинне навчання та безпечне виконання запитів.

Висновки до розділу 1

У першому розділі було проаналізовано сучасний стан кіберзагроз у сфері систем керування базами даних та визначено основні напрями досліджень у галузі кіберзахисту СКБД. На основі наведених прикладів реальних інцидентів продемонстровано, що SQL-ін’єкції та несанкціонований доступ до баз даних залишаються одними з найнебезпечніших і найпоширеніших загроз, які здатні призводити до масштабних витоків конфіденційної інформації, значних фінансових втрат і серйозних репутаційних наслідків.

Показано, що традиційні засоби захисту не завжди є достатніми в умовах швидкої еволюції методів атак, що зумовлює необхідність застосування адаптивних та інтелектуальних підходів до кіберзахисту СКБД. Особливу увагу приділено проблемам контролю доступу, захисту від SQL-ін’єкцій, аудиту, моніторингу та своєчасного виявлення аномальної активності.

У розділі сформульовано мету дисертаційної роботи, яка полягає у розробці методики забезпечення функціональної стійкості кіберзахисту СКБД, а також визначено основні завдання дослідження, спрямовані на проєктування архітектури веб-сервісу, розробку механізмів парсингу та фільтрації SQL-запитів, інтеграцію алгоритмів машинного навчання, організацію безпечного виконання запитів, логування, аудиту та побудову тестового середовища.

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ РОЗРОБКИ

2.1. Загальна характеристика об’єкта досліджень

У межах даного дослідження системи керування базами даних розглядаються як складні багаторівневі інформаційні системи, що включають програмні, апаратні та мережеві компоненти. Кожен з цих рівнів може стати потенційним об’єктом кіберзагроз, а отже потребує комплексного захисту. Особливістю сучасних СКБД є їх тісна інтеграція з веб-застосунками, корпоративними інформаційними системами та хмарними сервісами, що значно розширює поверхню атаки та ускладнює забезпечення належного рівня безпеки.

СКБД функціонують у середовищі постійної взаємодії з користувачами, програмними агентами та іншими інформаційними системами. Така взаємодія супроводжується великою кількістю запитів до баз даних, що створює сприятливі умови для реалізації різноманітних атак, зокрема SQL-ін’єкцій, атак на автентифікацію та авторизацію, перехоплення сесій, підміни даних, а також атак типу відмови в обслуговуванні. Виявлення таких загроз ускладнюється тим, що шкідливі дії часто маскуються під легітимну активність користувачів.

Особливу небезпеку становлять цільові та багатоетапні атаки, що здійснюються з використанням складних сценаріїв проникнення. У таких випадках зловмисник послідовно досліджує систему, знаходить слабкі місця в конфігурації, облікових записах або програмному забезпеченні, після чого поступово розширює свої привілеї та отримує доступ до критично важливих даних. Саме тому дослідження функціональної стійкості кіберзахисту СКБД повинно враховувати не лише окремі типи загроз, а й їх комбінації, часову динаміку розвитку атак та наслідки для всієї інформаційної системи.

Важливим елементом об’єкта дослідження є процеси моніторингу подій безпеки в СКБД. Моніторинг включає збір, збереження та аналіз даних про всі спроби доступу до бази даних, виконання SQL-запитів, зміну прав доступу, помилки автентифікації та інші події, що можуть свідчити про небезпечну

активність. Ефективний моніторинг дозволяє своєчасно виявляти аномалії, локалізувати джерела загроз та мінімізувати шкоду від кібератак.

Не менш важливим є дослідження процесів реагування на інциденти інформаційної безпеки. Реагування передбачає не лише виявлення факту атаки, а й оперативне прийняття рішень щодо блокування небезпечних дій, ізоляції скомпрометованих сегментів системи, відновлення нормального режиму роботи та проведення аналізу причин інциденту. У межах даної роботи ці процеси розглядаються як складові забезпечення функціональної стійкості СКБД.

Окрему увагу в дослідженні приділено питанням резервного копіювання та відновлення даних. Умови постійних кіберзагроз вимагають наявності надійних механізмів створення резервних копій баз даних, а також перевірених процедур їх відновлення у разі пошкодження або знищення інформації внаслідок атак. Стійкість СКБД безпосередньо залежить від швидкості та повноти відновлення даних після інцидентів.

Значну роль у забезпеченні кіберзахисту відіграють механізми керування доступом. Дослідження охоплює питання розмежування прав користувачів, автентифікації, авторизації та обліку дій користувачів у СКБД. Неправильна організація доступу або надмірні привілеї облікових записів підвищують ризик внутрішніх загроз та сприяють успішній реалізації атак.

В об’єкті дослідження також розглядається вплив програмних вразливостей СКБД та суміжного програмного забезпечення. До таких вразливостей належать помилки в програмному коді, застарілі версії програмних компонентів, неправильні налаштування серверів баз даних, веб-серверів та мережевого обладнання. Своєчасне виявлення та усунення таких вразливостей є необхідною умовою забезпечення стійкості системи.

Окремим напрямом дослідження є використання інтелектуальних методів аналізу для підвищення ефективності кіберзахисту СКБД. Алгоритми машинного навчання дозволяють здійснювати аналіз великих обсягів журналів подій, формувати профілі нормальної роботи системи та виявляти відхилення, що можуть

свідчити про кібератаки. Застосування таких методів підвищує точність виявлення загроз та зменшує залежність від заздалегідь визначених сигнатур.

Важливим аспектом об’єкта дослідження є також оцінка ризиків інформаційної безпеки для СКБД. Аналіз ризиків дозволяє визначити найбільш вразливі елементи системи, оцінити ймовірність реалізації загроз та можливі збитки, а також обґрунтувати вибір оптимальних заходів захисту. У межах дослідження ризики розглядаються як динамічні величини, що змінюються залежно від розвитку загроз та стану системи.

Дослідження охоплює і питання взаємодії СКБД з іншими елементами системи кібербезпеки організації, зокрема міжмережевими екранами, системами виявлення та попередження вторгнень, антивірусним програмним забезпеченням, засобами управління подіями безпеки. Така інтеграція дозволяє створити єдиний захисний контур та підвищити загальний рівень функціональної стійкості інформаційної інфраструктури.

Слід також зазначити, що об’єкт дослідження охоплює не лише технічні, а й організаційні аспекти забезпечення кіберзахисту СКБД. До них належать підготовка персоналу, розробка політик безпеки, регламентів реагування на інциденти, а також контроль дотримання вимог інформаційної безпеки. Людський фактор залишається одним з основних джерел ризиків, тому його врахування є необхідною умовою комплексного підходу до захисту.

У межах дослідження також аналізується вплив зовнішніх факторів на стійкість СКБД, зокрема збоїв електроживлення, відмов мережевого обладнання, апаратних несправностей серверів, а також природних та техногенних чинників. Функціональна стійкість у такому контексті розглядається як здатність системи зберігати працездатність або швидко відновлювати її не лише після кібератак, а й після різного роду аварійних ситуацій.

Таким чином, об’єкт дослідження в даній роботі охоплює сукупність процесів, пов’язаних із функціонуванням, захистом та відновленням систем керування базами даних в умовах впливу сучасних кіберзагроз. Комплексний розгляд технічних, програмних, організаційних та аналітичних аспектів дозволяє

сформувати цілісне уявлення про об’єкт дослідження та створює наукове підґрунтя для розробки ефективної методики забезпечення функціональної стійкості кіберзахисту СКБД.

2.2. Аналіз існуючих досліджень

За оглядом літератури встановлено, що існуючі методи кіберзахисту, які традиційно використовуються в СКБД, не завжди здатні ефективно протистояти новітнім загрозам, таким як складні багатоетапні атаки чи внутрішні загрози. У роботі «Database Security: Concepts, Approaches, and Challenges» (2016), автори Bertino, Sandhu, Sandhu зазначають важливість застосування базових методів контролю доступу, шифрування та автентифікації користувачів. Однак вони також визнають, що ці підходи не завжди ефективні для захисту від нових загроз, таких як SQL-ін’єкції або атаки на рівні додатків.

Так, у статті «A Survey of Database Security Issues and Techniques» (2014) Shiri і Khelil підкреслюється, що для досягнення належного рівня захисту необхідно впроваджувати новітні технології, такі як аналіз великих даних і машинне навчання для виявлення аномалій. Вони вказують, що традиційні методи не здатні ефективно запобігати новим типам атак, і пропонують використання методів класифікації для виявлення нетипової поведінки в системах СКБД.

Особливу увагу також варто звернути на роботи Dinesh і Kumar в статті «Anomaly Detection for Database Security: A Review» (2018), де описується використання технологій машинного навчання та аналізу патернів для виявлення аномалій, які можуть свідчити про потенційну атаку. Вони зазначають, що використання таких підходів дає можливість не тільки вчасно виявити загрози, а й адаптивно реагувати на змінені умови кіберзагроз, що значно підвищує стійкість СКБД до атак.

У статті «Intrusion Detection and Prevention Systems for Database Security» (2019) Choi і Lee аналізують інтеграцію систем виявлення і запобігання вторгненням (IDS/IPS) до СКБД, що дозволяє автоматично блокувати небезпечні

запити і операції в реальному часі. Вони підкреслюють, що таке поєднання технологій здатне не тільки виявляти, а й усувати загрози, запобігаючи значним порушенням безпеки.

Ці та інші подібні дослідження надають важливі основи для розробки нових методик і стратегій кіберзахисту СКБД, які дозволяють підвищити їх стійкість до кібератак. Так, у роботах B. Gupta та V. Jain у «Database Security: Challenges, Solutions, and Future Directions» (2017) розглядаються нові підходи до моніторингу і виявлення уразливих місць у СКБД, а також пропонуються рекомендації щодо інтеграції багаторівневих методів захисту, включаючи криптографічні технології та засоби аутентифікації.

Технології, що використовують машинне навчання для виявлення аномалій, можуть значно підвищити ефективність виявлення загроз, порівняно з традиційними методами контролю доступу та шифрування. Це дозволяє здійснювати більш точну і швидку ідентифікацію нових типів атак, таких як SQL ін'єкції, атаки типу "відмова в обслуговуванні" (DoS), а також інші складні загрози, що виникають у результаті внутрішніх зловживань або помилок в роботі системи. Машинне навчання дає змогу моделювати нормальні патерни поведінки користувачів та запитів до баз даних, що дозволяє виявляти аномальні або підозрілі дії навіть при мінімальній зміні їх поведінки.

Зокрема, технології, що ґрунтуються на методах глибокого навчання, забезпечують можливість аналізу величезних обсягів даних за короткий час, що робить ці методи надзвичайно ефективними для виявлення складних і багатоступінчастих атак. Вони використовують нейронні мережі для вивчення патернів у поведінці системи та здатні адаптуватися до нових типів атак, значно зменшуючи кількість помилкових спрацьовувань, які притаманні більш традиційним методам.

Відзначимо також дослідження «Security and Privacy Issues in Database Systems» (2020) авторів Li, Zhang і Chen, які акцентують увагу на важливості інтеграції таких технологій, як блокчейн, для забезпечення прозорості і надійності збереження журналів транзакцій. Вони розглядають можливості застосування

блокчейна для забезпечення постійної цілісності баз даних, особливо в контексті використання її для запису всіх змін в системі. Впровадження блокчейн-технологій дозволяє не тільки фіксувати всі операції в незмінному вигляді, а й забезпечує високий рівень прозорості для перевірки можливих порушень.

Інтеграція систем виявлення і запобігання вторгненням (IDS/IPS) продовжує залишатися ключовим елементом для забезпечення безпеки. Новітні підходи до реалізації таких систем з використанням машинного навчання дозволяють вчасно виявляти несанкціоновані спроби доступу та атаки, що здійснюються через уразливості в додатках, веб-сервісах чи мережах. Крім того, удосконалення цих систем дозволяє оперативно здійснювати блокування небезпечних запитів у реальному часі, що мінімізує можливі збитки.

Ці дослідження дозволяють сформулювати цілісну стратегію захисту СКБД, що включає комплексне застосування різних методів для збереження цілісності, доступності та конфіденційності даних. Застосування інтелектуальних технологій, таких як аналітика великих даних та моделювання загроз на основі реального аналізу даних, відкриває нові можливості для адаптивного і про активного захисту від новітніх кіберзагроз. Сучасні методи захисту СКБД повинні включати стратегії, які в максимальному ступені автоматизують виявлення та реагування на загрози. Підвищення ефективності захисту потребує створення інтегрованих архітектур, які можуть функціонувати в реальному часі, одночасно забезпечуючи високу ступінь адаптивності та масштабованості.

2.3. Підхід до проектування

Проектування системи забезпечення функціональної стійкості кіберзахисту для систем керування базами даних (СКБД) вимагає комплексного, всебічного та ретельно продуманого підходу, що охоплює безліч аспектів, які включають як технічні, так і організаційні заходи. Ця система повинна не тільки ефективно захищати бази даних від широкого спектра кіберзагроз, таких як несанкціонований

доступ, атаки на цілісність даних, маніпуляції з інформацією або крадіжку даних, а й гарантувати здатність до оперативного відновлення в разі порушення цілісності чи доступності даних. Важливим є також забезпечення того, щоб навіть у випадку складних або масштабних атак система залишалася стійкою і могла продовжувати виконувати свої функції без серйозних перерв. У разі того, що атаки або інциденти безпеки відбудуться, система повинна мати розвинуті механізми для швидкого відновлення та повернення до нормальної роботи.

Особливо важливо, щоб система не лише захищала дані від прямого несанкціонованого доступу або маніпуляцій з боку внутрішніх чи зовнішніх зловмисників, а й забезпечувала захист чутливих даних на всіх етапах їх обробки — від збору до зберігання та передачі через мережу. Також ключовим аспектом є здатність системи реагувати на нові, часто складні і невідомі кіберзагрози, які з’являються постійно в результаті розвитку технологій і нових методів атак.

Отже, проектування такої системи вимагає інтеграції певної кількості критичних компонентів, кожен з яких сприяє підвищенню загальної безпеки і стійкості СКБД до атак.

Розглянемо список цих компонентів:

- механізми автентифікації та авторизації;
- шифрування даних;
- механізми моніторингу та виявлення вторгнень (IDS/IPS);
- контроль доступу та політики безпеки;
- резервне копіювання та відновлення даних;
- інструменти для виявлення і запобігання SQL-ін'єкціям;
- системи управління патчами та оновленнями;
- інструменти для аудиту і звітності;
- план відновлення після інцидентів (Disaster Recovery Plan);
- інструменти для аналізу вразливостей і тестування безпеки;
- механізми взаємодії з кіберрозвідкою;
- автоматизація механізмів з використанням QI.

Розглянемо механізми автентифікації та авторизації:

Автентифікація – це перевірка особи користувача за допомогою паролів, біометрії, смарт-карт або багатофакторної автентифікації. Авторизація визначає рівень доступу користувача після успішної автентифікації.

Переваги

- захист від несанкціонованого доступу;
- багатофакторна автентифікація знижує ризик злому;
- легка інтеграція з іншими системами безпеки;
- аудит і моніторинг доступу.

Недоліки

- вразливість паролів до атак;
- складність налаштування та управління доступом;
- біометричні дані неможливо змінити у разі компрометації;
- ускладнення входу через багатофакторну автентифікацію;
- додаткові фінансові витрати на впровадження та підтримку;

Розглянемо шифрування даних:

Шифрування – це метод захисту інформації шляхом її перетворення у формат, який неможливо прочитати без спеціального ключа. Воно використовується для захисту конфіденційних даних під час їх збереження та передавання.

Симетричне шифрування (AES, DES) використовує один ключ для шифрування і розшифрування. Воно швидке, але потребує безпечного обміну ключами.

Асиметричне шифрування (RSA, ECC) застосовує два ключі: відкритий для шифрування і закритий для розшифрування. Воно забезпечує безпечний обмін даними, але є ресурсномістким. Шифрування застосовується у VPN, TLS для захисту мережевого трафіку, а також у шифруванні дисків, файлів і баз даних для захисту збережених даних.

Переваги

- захист конфіденційної інформації від несанкціонованого доступу;

- можливість безпечного передавання даних через відкриті канали;
- запобігання втратам у разі фізичного доступу до пристрою.

Недоліки

- високе навантаження на ресурси системи при складних алгоритмах;
- необхідність безпечного зберігання та управління ключами;
- ризик компрометації ключів, що може призвести до розшифрування даних.

Розглянемо Механізми моніторингу та виявлення вторгнень (IDS/IPS):

Системи виявлення та запобігання вторгненням призначені для контролю мережевого трафіку і дій користувачів з метою виявлення підозрілих активностей та атак. IDS (Intrusion Detection System) – це пасивна система, яка аналізує трафік і повідомляє про можливі загрози. Вона буває мережевою (NIDS) або хостовою (HIDS). IPS (Intrusion Prevention System) – це активна система, яка не лише виявляє загрози, а й блокує їх у реальному часі. Вона інтегрується у мережеві пристрої та може автоматично запобігати атакам. Такі механізми використовуються для захисту корпоративних мереж, серверів та систем.

Переваги

- виявлення атак та загроз у реальному часі;
- автоматичне реагування на виявлені загрози;
- захист як мережевих, так і окремих хостів.

Недоліки

- велика кількість помилкових спрацьовувань, що треба перевіряти;
- високі вимоги до ресурсів системи;
- недостатня ефективність проти складних багатовекторних атак.

Розглянемо механізм контролю доступу та політики безпеки:

Контроль доступу – це система, яка визначає, хто може отримати доступ до певних ресурсів, які дії дозволені та за яких умов. ACL (Access Control List) використовує списки дозволених і заборонених дій для конкретних користувачів або груп. RBAC (Role-Based Access Control) надає доступ на основі ролей, що

спрощує управління великими системами. ABAC (Attribute-Based Access Control) враховує додаткові атрибути, такі як час, місцезнаходження чи тип пристрою.

Політики безпеки включають правила використання мережі, пристроїв, облікових записів і збережених даних для запобігання витокам та несанкціонованому доступу. *Переваги*

- запобігання несанкціонованому доступу до критичних систем;
- гнучке налаштування рівнів доступу для різних користувачів;
- підвищення безпеки за рахунок чітко визначених політик.

Недоліки

- складність налаштування та адміністрування великих систем;
- можливість конфліктів між політиками доступу;
- можливе зниження продуктивності через додаткові перевірки доступу.

Розглянемо резервне копіювання та відновлення даних:

Резервне копіювання – це створення копій даних для їхнього відновлення у разі втрати, пошкодження або кібератаки. Використовуються повне, диференційне та інкрементне копіювання. Повне копіювання зберігає всі дані, забезпечуючи максимальну надійність, але потребує багато місця. Диференційне копіювання зберігає лише зміни з моменту останнього повного копіювання, а інкрементне – лише останні зміни, що зменшує використання ресурсів. Відновлення даних передбачає відновлення інформації зі збережених копій. Воно може бути ручним або автоматичним, залежно від налаштованих політик.

Переваги

- захист від втрати даних через збої, віруси або людські помилки;
- можливість швидкого відновлення критичних систем;
- гнучкість у виборі стратегій копіювання.

Недоліки

- велике споживання дискового простору при повному копіюванні;
- часові затримки при відновленні великих обсягів даних;
- ризик компрометації резервних копій при недостатньому захисті.

Інструменти для виявлення і запобігання SQL-ін'єкціям:

SQL-ін'єкції – це атаки, при яких зловмисники вводять шкідливі SQL-запити для отримання несанкціонованого доступу до БД. Захист включає використання параметризованих запитів, підготовлених виразів та ORM (Object Relational Mapping), які унеможливлюють виконання шкідливого коду. Інструменти виявлення SQL-ін'єкцій аналізують вхідні запити та виявляють аномальні дії. Веб-аплікаційні WAF можуть блокувати підозрілі запити ще до їхнього виконання.

Переваги

- запобігання витоку конфіденційної інформації з баз даних;
- автоматичне виявлення та блокування атак;
- легка інтеграція з сучасними веб-додатками.

Недоліки

- можливі помилкові блокування легітимних запитів;
- високе навантаження на сервер при аналізі запитів у реальному часі;
- необхідність регулярного оновлення правил безпеки.

Розглянемо з чого складається системи управління патчами та оновленнями:

Системи управління патчами забезпечують оновлення програмного забезпечення для усунення вразливостей, підвищення продуктивності та додавання нових функцій. Автоматичне оновлення дозволяє швидко закривати вразливості, зменшуючи ризик атак. Централізовані системи (WSUS, SCCM, Ansible) допомагають керувати оновленнями у великих мережах, дозволяючи тестувати патчі перед розгортанням. Оновлення можуть бути безпечними (зосередженими лише на виправленні вразливостей) або функціональними (що додають нові можливості).

Переваги

- підвищення безпеки шляхом усунення вразливостей;
- автоматизація процесу оновлення, що знижує ризик помилок;
- централізоване управління у великих інфраструктурах.

Недоліки

- ризик несумісності оновлень з існуючими системами;
- можливі збої під час встановлення патчів;
- тестування оновлень перед розгортанням у критичних системах.

Розглянемо способи використання інструментів для аудиту і звітності:

Інструменти для аудиту і звітності забезпечують детальний моніторинг дій користувачів та адміністраторів у системах та мережах. Вони реєструють події, що можуть свідчити про порушення безпеки або неналежне використання ресурсів. Завдяки цим інструментам можна генерувати звіти про активність, виявляти підозрілі чи аномальні події, що допомагають у розслідуваннях інцидентів. Такі інструменти включають SIEM-системи (Security Information and Event Management), які аналізують та корелюють дані з різних джерел для оперативного реагування.

Переваги

- покращена здатність виявляти порушення та несанкціоновану діяльність;
- можливість забезпечення відповідності нормативним вимогам , наприклад (GDPR, PCI DSS);
- детальне розслідування інцидентів на основі збережених журналів.

Недоліки

- велика кількість даних для обробки, що потребує потужних систем;
- висока вартість і складність інтеграції в існуючі інфраструктури;
- необхідність регулярного оновлення політик та правил для виявлення нових загроз.

План відновлення після інцидентів (Disaster Recovery Plan):

План відновлення після інцидентів (Disaster Recovery Plan, DRP) описує сукупність організаційних і технічних процесів, що дозволяють відновити критичні бізнес-системи після значних інцидентів або катастроф, зокрема кібератак, відмов обладнання, збоїв електроживлення чи природних лих. До основних заходів плану належать створення та регулярне оновлення резервних копій даних, налаштування

дублюючих систем, визначення відповідальних осіб за відновлення, а також забезпечення безперервності бізнес-процесів і мінімізація часу простою систем.

Важливими показниками в межах DRP є час відновлення (RTO — Recovery Time Objective) та допустимі втрати даних (RPO — Recovery Point Objective), які визначають максимально допустимі межі простою системи та обсягу втрачених даних. Наявність регламентованого плану відновлення дозволяє швидко реагувати на інциденти та зменшувати їх негативний вплив на діяльність організації.

Існують різні стратегії відновлення, що відрізняються рівнем готовності до роботи, швидкістю відновлення та обсягом необхідних ресурсів. Вибір конкретної стратегії залежить від критичності інформаційних систем, фінансових можливостей організації та вимог до безперервності її діяльності.

Переваги

- забезпечення безперервності бізнесу в разі великих інцидентів;
- швидке відновлення критичних систем і даних;
- мінімізація збитків через швидке реагування.

Недоліки

- висока вартість на створення та підтримку резервних майданчиків;
- складність у підтримці та тестуванні актуальності плану;
- ризик недостатнього планування для всіх можливих сценаріїв.

Розглянемо інструменти для аналізу вразливостей і тестування безпеки:

Інструменти для аналізу вразливостей сканують системи, мережі та додатки на наявність відомих уразливостей, таких як незахищені порти, старі версії ПЗ чи неправильні налаштування. Вони використовуються для проактивного виявлення слабких місць, які можуть бути використані зловмисниками.

Тестування безпеки включає як автоматизовані сканери, так і ручне тестування (пентестинг). Найпоширеніші інструменти включають Nessus, OpenVAS, Burp Suite та інші.

Переваги

- виявлення вразливостей до того, як вони можуть бути використані;
- полегшення процесу виправлення помилок і підвищення безпеки;

- покращення відповідності стандартам безпеки.

Недоліки

- не завжди виявляються нові або невідомі вразливості;
- можливі помилки в результатах сканування, що вимагає перевірки;
- залежність від актуальності бази даних уразливостей.

Розглянемо механізми взаємодії з кіберрозвідкою:

Механізми взаємодії з кіберрозвідкою включають отримання інформації про нові загрози, вектори атак і зловмисників з відкритих джерел та спеціалізованих кіберрозвідувальних агентств. Інтеграція з кіберрозвідкою допомагає організаціям оперативно реагувати на нові атаки та запобігати загрозам. Це включає обмін даними і аналіз кіберінцидентів на глобальному рівні.

Переваги

- оперативне виявлення нових типів загроз;
- поліпшення прогнозування атак та їхніх векторів;
- підвищення рівня захисту завдяки обміну знаннями.

Недоліки

- потрібна постійна актуальність і точність даних розвідки;
- висока вартість інтеграції з кіберрозвідувальними платформами;
- залежність від зовнішніх джерел інформації.

Автоматизація механізмів з використанням QI:

Автоматизація з використанням квантових інтерфейсів (QI) полягає в застосуванні технологій квантових обчислень. Це включає автоматизацію процесів аналізу загроз, виявлення аномалій і прогнозування атак з використанням квантових алгоритмів. Це новий напрямок, для підвищення ефективності безпеки в інфраструктурах з високим рівнем складності.

Переваги

- значне підвищення швидкості обробки великих обсягів даних;
- можливість прогнозування загроз завдяки;
- зниження ризику людських помилок завдяки автоматизації процесів.

Недоліки

- висока вартість впровадження квантових технологій;
- обмежена доступність квантових ресурсів на ринку;
- необхідність в фахівцях для роботи з квантовими алгоритмами.

Технологічний розвиток супроводжується зростанням кіберзагроз, що робить комплексний підхід до захисту даних критично важливим. Автентифікація, шифрування, виявлення загроз, резервне копіювання, аудит та моніторинг утворюють взаємопов'язану систему безпеки, де кожен елемент підтримує інші. Відсутність хоча б одного з цих компонентів створює вразливості, які можуть призвести до витоку даних, фінансових втрат і репутаційних збитків.

Компанії по всьому світу активно впроваджують такі методи. **Google** застосувала двофакторну автентифікацію, що суттєво знизило ризик несанкціонованого доступу. **Microsoft** шифрує дані на всіх етапах роботи сервісів Azure, забезпечуючи відповідність міжнародним стандартам. **Amazon** використовує системи моніторингу AWS для швидкого виявлення аномалій, **Dropbox** — стратегії

резервного копіювання, а **Apple** впровадила аудит додатків у App Store для захисту користувачів.

Ці приклади підтверджують: лише комплексна інтеграція сучасних методів кіберзахисту гарантує стабільність інформаційних систем та збереження довіри користувачів.

Висновки до розділу 2

У другому розділі здійснено комплексний аналіз методів розробки та забезпечення кіберзахисту систем керування базами даних як багаторівневих інформаційних систем, що функціонують в умовах постійного впливу зовнішніх і внутрішніх загроз. Встановлено, що сучасні СКБД характеризуються високим рівнем інтеграції з веб-застосунками, корпоративними системами та хмарними

сервісами, що суттєво підвищує ризик реалізації складних кібератак і ускладнює процеси захисту.

Проаналізовано основні групи загроз для СКБД, зокрема SQL-ін’єкції, несанкціонований доступ, атаки на автентифікацію та авторизацію, перехоплення даних, DoS-атаки, а також багатоетапні цільові атаки. Показано, що ефективний кіберзахист неможливий без системного підходу, який поєднує технічні, програмні та організаційні заходи безпеки.

На основі аналізу існуючих досліджень встановлено, що традиційні методи захисту СКБД (контроль доступу, шифрування, сигнатурні механізми виявлення загроз) є недостатніми для протидії сучасним адаптивним і багаторівневим атакам. Обґрунтовано доцільність використання інтелектуальних методів аналізу, зокрема алгоритмів машинного навчання та глибокого навчання, для підвищення точності виявлення аномалій і зниження кількості помилкових спрацьовувань.

У розділі детально розглянуто основні компоненти системи кіберзахисту СКБД, включно з механізмами автентифікації та авторизації, шифруванням даних, системами IDS/IPS, контролем доступу, резервним копіюванням і відновленням, захистом від SQL-ін’єкцій, управлінням оновленнями, аудитом і звітністю, планом відновлення після інцидентів, засобами аналізу вразливостей, інтеграцією з кіберрозвідкою та автоматизацією процесів. Для кожного з цих компонентів визначено їхні переваги та обмеження.

Показано, що максимальний рівень функціональної стійкості СКБД досягається лише за умов комплексної інтеграції всіх перелічених механізмів у єдиний захисний контур. Відсутність або недостатня ефективність окремих елементів системи створює критичні вразливості, які можуть бути використані зловмисниками.

Таким чином, результати аналізу, отримані у другому розділі, підтверджують необхідність розробки інтегрованої, адаптивної та інтелектуальної системи кіберзахисту СКБД, здатної забезпечувати їх функціональну стійкість в умовах сучасних кіберзагроз. Отримані висновки слугують теоретичним підґрунтям для

подальшого проєктування, реалізації та експериментального дослідження запропонованої системи у наступних розділах роботи.

РОЗДІЛ 3. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Використані технології

Розроблення системи забезпечення функціональної стійкості кіберзахисту баз даних здійснювалося із застосуванням сучасних технологій, фреймворків та інструментів, що забезпечують ефективність, масштабованість і безпеку веб-додатку.

Під час створення системи враховувалися вимоги до **надійності, гнучкості, продуктивності та стійкості до кібератак**. Архітектура рішення побудована за принципами модульності, що дозволяє легко розширювати функціонал і адаптувати систему під різні сценарії використання.

Особливу увагу приділено захисту даних на всіх рівнях — від перевірки користувацьких запитів до контролю доступу та моніторингу активності бази даних. Використання сучасних засобів машинного навчання та аналітики підвищує рівень автоматизації виявлення загроз і дозволяє оперативно реагувати на підозрілі дії.

Використані технології:

Мова програмування Python

Основна мова розробки проєкту. Використовувалася для створення серверної логіки, реалізації обробки SQL-запитів, модулів машинного навчання та аналітики. Python забезпечує високу продуктивність розробки та широкий набір бібліотек для роботи з базами даних і безпекою.

Фреймворк Flask

Легковаговий веб-фреймворк, який забезпечує створення REST-API для обробки HTTP-запитів. Flask дозволив реалізувати модульну структуру сервісу, інтегрувати механізми автентифікації, логування та перевірки SQL-запитів.

Бібліотека SQLAlchemy

Використана як ORM-шар для взаємодії з базою даних PostgreSQL. Забезпечує безпечне формування SQL-запитів, транзакційність операцій та ізоляцію рівня даних, що знижує ризики SQL-ін’єкцій.

Система керування базами даних PostgreSQL

Основна СКБД проєкту, що забезпечує зберігання користувацьких даних, логів та результатів виконання запитів. PostgreSQL підтримує розширені можливості безпеки, такі як контроль доступу, SSL-шифрування та тригери аудиту.

Модуль машинного навчання (scikit-learn)

Використано для створення моделі класифікації SQL-запитів. Застосовувалися
методи TF-IDF векторизації та логістичної регресії для оцінки ймовірності шкідливості запитів.

Мова запитів SQL

Основний засіб взаємодії з базою даних. У проєкті реалізовано механізми аналізу, фільтрації та безпечного виконання SQL-запитів з метою запобігання ін’єкціям і порушенням цілісності даних.

JSON (JavaScript Object Notation)

Формат обміну даними між клієнтською частиною та сервером. Забезпечує зручну структуру для передачі SQL-запитів і результатів їхнього виконання.

Технології безпеки та кіберзахисту

Реалізовано комбінацію статичних і динамічних методів аналізу SQL-запитів, фільтрацію на основі правил і алгоритмів машинного навчання, а також sandbox-режим для безпечного виконання підозрілих запитів.

3.2. Використані інструменти

PyCharm

Основне середовище розробки (IDE), яке забезпечує зручну роботу з Python-кодом, інтеграцію з віртуальним середовищем, налагодження (debugging) і запуск локального Flask-сервера.

PostgreSQL Tools (pgAdmin)

Використовувалися для адміністрування бази даних, створення таблиць, перегляду даних, виконання SQL-запитів і моніторингу активності.

Віртуальне середовище Python (.venv)

Застосовувалося для ізоляції залежностей проєкту та уникнення конфліктів між бібліотеками.

Git і GitHub

Система контролю версій, яка використовувалася для збереження, відстеження змін у коді та командної роботи над проєктом.

Postman / cURL

Інструменти для тестування API-запитів. Використовувалися для перевірки коректності взаємодії між клієнтом і сервером, а також для тестування захищеності ендпойнтів.

Requests (Python-бібліотека)

Застосовувалася для внутрішнього тестування API-викликів безпосередньо з Python-середовища.

Jupyter Notebook / Google Colab

Використовувалися на етапі попереднього навчання і тестування моделей машинного навчання.

Бібліотеки допоміжного призначення:

- **pandas** — обробка табличних даних;
- **joblib** — збереження моделей ML;
- **traceback / logging** — діагностика та відстеження помилок.

У процесі розроблення системи було використано сучасні технології та інструменти, які забезпечили ефективність, безпеку й масштабованість рішення. Завдяки застосуванню **Flask** реалізовано гнучку архітектуру веб-додатку, **SQLAlchemy** забезпечило безпечну роботу з базою даних, а інтеграція засобів **машинного навчання** підвищила рівень інтелектуального аналізу SQL-запитів.

Використання **PostgreSQL**, **PyCharm** та системи контролю версій **Git** сприяло стабільній роботі, зручному налагодженню та розширюваності системи.

Висновки до розділу 3

У третьому розділі проаналізовано програмні технології та інструменти, використані для розроблення системи забезпечення функціональної стійкості кіберзахисту баз даних. Обґрунтовано вибір сучасного технологічного стеку, який забезпечує надійність, масштабованість, безпеку та гнучкість створюваного веб-застосунку. Встановлено, що використання мови програмування Python дозволило ефективно реалізувати серверну частину системи, модулі аналізу SQL-запитів та механізми машинного навчання завдяки широкому набору спеціалізованих бібліотек. Застосування фреймворку Flask забезпечило побудову модульної REST-архітектури, що спростило інтеграцію компонентів безпеки, логування та обробки запитів.

Використання ORM-бібліотеки SQLAlchemy дозволило реалізувати безпечну взаємодію з базою даних PostgreSQL, мінімізувавши ризики SQL-ін’єкцій та забезпечивши транзакційність операцій. PostgreSQL, як надійна та захищена СКБД, забезпечила стабільне зберігання даних, журналів подій та результатів аналізу. Інтеграція модуля машинного навчання на основі бібліотеки scikit-learn підвищила рівень інтелектуального аналізу SQL-запитів і дозволила реалізувати механізми автоматичної класифікації запитів за рівнем небезпеки. Використання форматів JSON забезпечило ефективний та стандартизований обмін даними між клієнтською і серверною частинами системи. Застосування сучасних інструментів розробки, зокрема PyCharm, pgAdmin, Git, Postman та Jupyter Notebook, забезпечило зручність тестування, налагодження і супроводу програмного коду, а також сприяло підвищенню якості програмної реалізації. Використання віртуальних середовищ Python дозволило ізолювати залежності проєкту та уникнути конфліктів бібліотек.

Таким чином, обрані технології та інструменти повністю відповідають вимогам до розроблення веб-застосунків та дозволяють реалізувати СКБД з високим рівнем функціональної стійкості, безпеки з можливістю масштабування.

РОЗДІЛ 4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура застосунку

Архітектура розробленої системи забезпечення функціональної стійкості кіберзахисту систем керування базами даних побудована за модульним принципом і має багаторівневу структуру, що дозволяє розділити функції збору даних, їх аналізу, обробки та візуалізації результатів. Такий підхід забезпечує масштабованість системи, підвищену надійність та можливість роботи в режимі реального часу.

Клієнтський рівень представлений веб-браузером користувача, через який здійснюється доступ до інтерфейсу системи. Саме на цьому рівні відбувається візуалізація результатів аналізу SQL-запитів, статистики атак, журналів подій та аналітичних показників у вигляді таблиць і графіків. Клієнтська частина забезпечує зручний контроль стану безпеки СКБД та підтримку прийняття рішень адміністратором.

Серверний рівень архітектури виконує основну функціональну обробку даних та реалізує бізнес-логіку системи. Він забезпечує прийом HTTP-запитів, їх аналіз, перевірку на наявність загроз, застосування механізмів автентифікації та авторизації, а також взаємодію з базами даних. Серверна частина є центральним елементом системи, який об’єднує всі модулі кіберзахисту в єдиний захисний контур.

Для підтримки обробки подій у режимі реального часу в архітектурі передбачено використання потокових сервісів, які забезпечують передачу даних про активність у СКБД без затримок. Це дозволяє оперативно реагувати на підозрілі дії та відображати актуальний стан безпеки системи.

Окреме місце в архітектурі займає інтелектуальний рівень, який реалізується

за допомогою модуля машинного навчання. Він здійснює аналіз поведінки користувачів і SQL-запитів, виявляє аномальну активність та формує оцінку рівня ризику. Результати цього аналізу використовуються для прийняття рішень щодо блокування або обмеження доступу.

У системі використовується декілька спеціалізованих баз даних: база користувацьких даних, база журналів атак та робоча база для виконання SQL-запитів. Такий розподіл дозволяє ізолювати критично важливу інформацію, підвищити безпеку зберігання даних та оптимізувати процеси аналізу.

Для взаємодії з зовнішніми сервісами та системами кібербезпеки в архітектурі застосовується API-шлюз, який забезпечує контрольований доступ до зовнішніх API та інтеграцію з додатковими засобами захисту й моніторингу.

Запропонована архітектура поєднує класичні механізми кіберзахисту з інтелектуальними методами аналізу та характеризується гнучкістю, модульністю і високою адаптивністю до сучасних кіберзагроз. Саме така побудова системи створює основу для реалізації серверної логіки, детально описаної в підрозділі 4.1.

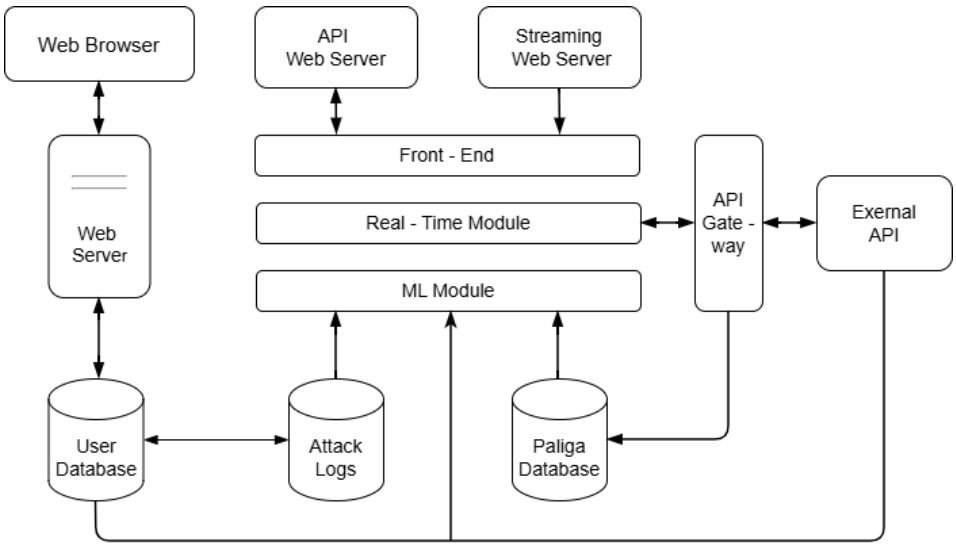


Рисунок 4.1 – Архітектура системи

На рисунку 4.1 представлено архітектуру системи кіберзахисту систем керування базами даних.

Система побудована за клієнт-серверним принципом і включає користувацький, серверний та аналітичний рівні. Клієнтський рівень реалізований у вигляді веб-інтерфейсу, через який користувач взаємодіє з системою. Серверний рівень забезпечує прийом і обробку запитів, а також взаємодію між усіма внутрішніми модулями. Обробка подій безпеки здійснюється в режимі реального часу за допомогою спеціалізованого модуля. Для інтелектуального аналізу використовується модуль машинного навчання. У системі передбачено декілька баз даних для зберігання користувацької інформації, журналів атак та робочих даних. Архітектура підтримує потокову обробку інформації для оперативного реагування на загрози. Також реалізована інтеграція із зовнішніми сервісами через API-шлюз. Запропонована структура забезпечує модульність, масштабованість та підвищену функціональну стійкість системи.

4.2. Концептуальна модель системи

Концептуальна модель системи відображає логічну структуру розробленого програмного забезпечення та визначає основні функціональні модулі, що забезпечують кіберзахист системи керування базами даних. На відповідній діаграмі система подана як сукупність взаємопов’язаних компонентів, кожен з яких виконує окремий набір завдань, але разом вони формують єдиний захисний контур.

На верхньому рівні модель представлена узагальненим елементом «Система», що охоплює всі підсистеми моніторингу, аналізу та обробки SQL-запитів. Нижче виділено три ключові функціональні модулі: модуль реального часу, модуль ML-аналізу та модуль журналів атак. Таке розділення дозволяє чітко окреслити обов’язки кожної підсистеми та спростити подальшу програмну реалізацію.

Модуль реального часу відповідає за первинну обробку SQL-запитів, що

надходять до системи. У концептуальній моделі він розділений на два підфункціональні блоки: перехоплення SQL та аналіз правил. Підсистема перехоплення забезпечує фіксацію усіх запитів до бази даних до їх фактичного виконання, що дає змогу застосовувати механізми попереджувального контролю. Підсистема аналізу правил виконує перевірку запитів на відповідність наперед визначеним політикам безпеки, виявляє небезпечні конструкції та готує дані для подальшого інтелектуального аналізу.

Модуль ML-аналізу реалізує інтелектуальну частину системи та працює з даними, отриманими з модуля реального часу та журналів атак. У моделі він поділений на дві основні функції: класифікація запитів та навчання моделі. Класифікація запитів полягає у визначенні рівня ризику конкретного SQL-запиту (безпечний, підозрілий, потенційно шкідливий) з використанням попередньо навченої моделі машинного навчання. Підсистема навчання моделі відповідає за періодичне оновлення параметрів моделі на основі нових даних, що дозволяє системі адаптуватися до змін у характері атак та підвищувати точність виявлення загроз.

Модуль журналів атак забезпечує накопичення та структуризацію інформації про всі підозрілі та заблоковані запити. На діаграмі він деталізований як дві функції: збір і збереження логів та формування датасету. Підсистема збору логів реєструє час, джерело запиту, тип виявленої загрози, рішення системи та інші атрибути, необхідні для аудиту та подальшого аналізу. Підсистема формування датасету агрегує ці дані у зручному для машинного навчання форматі, що використовується модулем ML-аналізу для навчання та валідації моделі.

Розглянемо концептуальну діаграму системи, яка наочно відображає взаємозв’язки між усіма функціональними модулями. Вона демонструє послідовність проходження SQL-запиту від етапу первинної обробки до журналювання результатів і подальшого навчання ML-моделі.

Рисунок 4.2 – Концептуальна модель системи

На рисунку 4.2 представлено концептуальну модель системи кіберзахисту СКБД у вигляді ієрархічної структури, де центральним елементом є система, що об’єднує модуль реального часу, модуль ML-аналізу та модуль журналів атак. Модуль реального часу містить підсистеми перехоплення SQL-запитів і аналізу правил. Модуль ML-аналізу включає блоки класифікації запитів і навчання моделі. Модуль журналів атак відображає процеси збору логів та формування датасету. Стрілки на діаграмі показують напрямки обміну даними між основними компонентами.

Таким чином, концептуальна модель системи демонструє, як модуль реального часу, модуль ML-аналізу та модуль журналів атак спільно забезпечують виявлення, класифікацію та документування кібератак на рівні SQL-запитів. Така структура дозволяє чітко розмежувати відповідальність між компонентами, спростити їх програмну реалізацію та подальше масштабування системи. У наступних підрозділах ці концептуальні елементи деталізуються на рівні архітектури застосунку та опису серверної й клієнтської частин.

Для деталізації взаємодії користувача із функціональними модулями системи концептуальна модель доповнена діаграмою прецедентів. На ній відображено основні сценарії використання системи кіберзахисту СКБД.

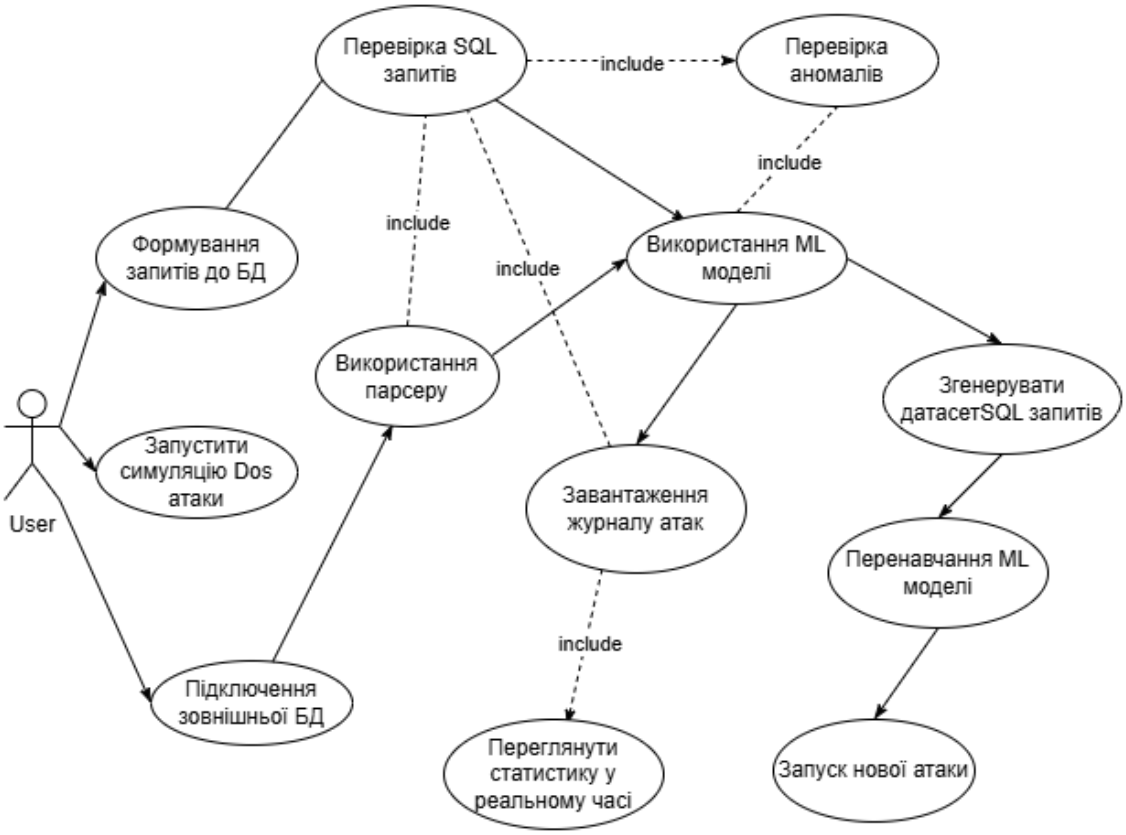


Рисунок 4.3 - Діаграма прецедентів взаємодії користувача з системою кіберзахисту СКБД

Зокрема ми можемо побачити формування та перевірку SQL-запитів, використання парсеру, застосування ML-моделі для виявлення аномалій, перегляд статистики в реальному часі, завантаження журналу атак, генерацію навчального датасету та перенавчання моделі. Окремими прецедентами подано сценарії підключення до зовнішньої бази даних, запуску нових атак та моделювання DoS-атак з метою тестування стійкості системи. Використання зв’язків типу «include» на діаграмі відображає логічну послідовність процесів обробки запиту: від його формування та синтаксичної перевірки до інтелектуального аналізу із застосуванням ML-моделі та реєстрації події в журналі атак. Таким чином, діаграма прецедентів логічно доповнює концептуальну модель системи та наочно демонструє практичну реалізацію взаємодії користувача з основними функціональними компонентами системи кіберзахисту СКБД.

4.3. Логічна структура програмної системи

Логічна структура програмної системи відображає основні програмні сутності серверної частини, їх функціональне призначення та взаємозв'язки між собою. Вона формується на основі модульного підходу, що забезпечує чітке розмежування відповідальності між компонентами, підвищує керованість коду, надійність роботи та спрощує процес масштабування системи кіберзахисту СКБД.

Центральним елементом логічної структури є клас-координатор, який здійснює узгоджену взаємодію між модулями синтаксичного аналізу SQL-запитів, машинного навчання та доступу до бази даних. Саме через цей клас відбувається інтеграція всіх основних механізмів перевірки запитів та прийняття рішень щодо їх виконання або блокування.

Клас **SQLRuleParser** відповідає за реалізацію правил сигнатурного аналізу SQL-запитів. Він містить набір правил та список небезпечних операцій, на основі яких виконується перевірка тексту запиту. Метод перевірки повертає структурований результат у вигляді словника, що містить інформацію про наявність заборонених конструкцій, тип виявленої загрози та підставу для блокування. Даний клас забезпечує первинний рівень захисту на основі формальних правил.

Клас **MLChecker** реалізує інтелектуальний рівень аналізу запитів і відповідає за оцінювання ризику з використанням методів машинного навчання. У його складі збережено векторизатор, навчальну модель, а також порогові значення ризику. Метод прогнозування дозволяє визначити числову оцінку небезпеки SQL-запиту, а додатковий метод класифікації приймає рішення про віднесення запиту до безпечних, підозрілих або шкідливих. Таким чином, MLChecker виконує функцію адаптивного аналізу загроз.

Клас **DBConnectionManager** забезпечує централізоване керування підключенням до бази даних. Він зберігає параметри підключення, формує об'єкт з'єднання та надає його іншим модулям системи. Це дозволяє ізолювати логіку

доступу до бази даних від бізнес-логіки перевірки запитів, підвищує безпеку та спрощує адміністрування.

Клас **SchemaExplorer** використовується для отримання інформації про структуру бази даних. Він звертається до менеджера з'єднань, виконує запити для отримання переліку таблиць та їх полів, що необхідно для аналізу структури БД, формування навчальних вибірок та коректної генерації SQL-запитів. Цей клас відіграє допоміжну роль у процесі підготовки даних для машинного навчання.

Клас **DatasetBuilder** відповідає за формування навчальних наборів даних. Він реалізує механізми автоматичної генерації безпечних та шкідливих SQL-запитів, обробку структури бази даних, а також створення датасетів для подальшого навчання моделі машинного навчання. Даний клас забезпечує зв'язок між реальними даними СКБД та інтелектуальним модулем аналізу.

Взаємодія між класами побудована за принципом використання допоміжних сервісів. Клас-координатор викликає модулі SQLRuleParser та MLChecker для перевірки запиту, MLChecker використовує підключення до бази через DBConnectionManager, а формування датасету відбувається із залученням SchemaExplorer та DatasetBuilder. Така структура дозволяє гнучко змінювати або розширювати окремі компоненти без порушення функціональності системи.

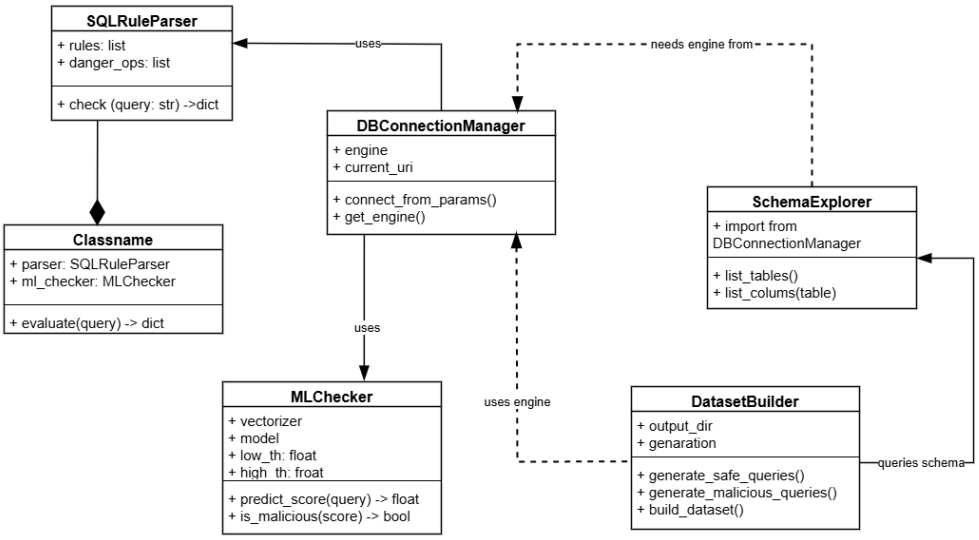


Рисунок 4.4– Діаграма класів

На рисунку 4.4 зображена логічна структура програмної системи кіберзахисту СКБД (UML-діаграма класів). Таким чином, логічна структура програмної системи забезпечує чітку модульність, розмежування відповідальності між компонентами та узгоджену роботу механізмів перевірки, аналізу та навчання. Це створює основу для стабільної, масштабованої та функціонально стійкої роботи системи кіберзахисту СКБД.

4.4. Серверна частина застосунку

Серверна частина веб-застосунку відповідає за прийом та обробку HTTP-запитів, безпечне виконання SQL-запитів, контроль доступу користувачів, аудит дій та моніторинг безпеки. Вона побудована на основі **Python** і фреймворку **Flask**, що дозволяє реалізувати REST-інтерфейси та інтегрувати різні модулі системи.

Flask-сервер (ядро застосунку).

Flask-сервер забезпечує координацію роботи всіх модулів системи. Він приймає HTTP-запити від клієнтів, забезпечує маршрутизацію їх до відповідних функцій та обробляє відповіді. Сервер відповідає за основну логіку веб-застосунку і забезпечує безпечний обмін даними між користувачем і базою даних. А саме:

- прийом і маршрутизація HTTP-запитів;
- реалізація REST-інтерфейсу для взаємодії клієнт-сервер;
- координація роботи всіх модулів системи.

Модуль перевірки SQL-запитів (SQLRuleParser)

Цей модуль виконує синтаксичний аналіз SQL-запитів, виявляє шкідливі команди і токени, забезпечує блокування небезпечних запитів. У разі виявлення порушення формує причину блокування, здійснюється аудит та контроль. А саме:

- синтаксичний аналіз SQL-запитів;
- виявлення небезпечних команд, таких як DROP, DELETE, ALTER;
- формування причини блокування (reason) для аудиту.

Модуль машинного навчання (MLChecker)

MLChecker оцінює ризик SQL-запиту на основі попередньо навченої моделі (TF-IDF + Logistic Regression). Модуль приймає рішення про дозвіл, блокування або виконання у sandbox, що дозволяє оцінити підозрілі запити без зміни даних у базі.

А саме:

- оцінка ризику SQL-запитів із використанням попередньо навченої моделі;
- прийняття рішень: дозволити, блокувати або виконати у sandbox;
- Порівняння результатів з пороговими значеннями ML_LOW_THRESHOLD і ML_HIGH_THRESHOLD.

Модуль взаємодії з базою даних (SQLAlchemy ORM)

Цей модуль забезпечує безпечне виконання SQL-запитів і транзакцій. Він підтримує sandbox-режим для підозрілих запитів, що дозволяє перевіряти їх результати без внесення змін у базу даних. ORM-підхід оптимізує доступ до даних та полегшує інтеграцію з іншими модулями.

А саме:

- безпечне виконання SQL-запитів і транзакцій;
- sandbox-режим для ізоляції підозрілих запитів;
- оптимізація доступу до даних через ORM-підхід.

Модуль автентифікації та авторизації (auth_module)

Цей модуль контролює доступ користувачів до системи, забезпечуючи безпеку ресурсів через токени або сесійні механізми. Він гарантує, що лише авторизовані користувачі можуть виконувати запити та отримувати доступ до даних. А саме:

- контроль доступу користувачів до системи;
- використання токенів або сесійних механізмів безпеки.

Логування та аудит (log_module)

Модуль логування фіксує всі виконані, заблоковані та підозрілі запити. Він зберігає інформацію про час, джерело запиту, рівень ризику та результати аналізу, що дозволяє моніторити кіберзагрози та формувати звіти для аудиту. А саме:

- фіксація виконаних, заблокованих та підозрілих запитів;
- збереження інформації про час, джерело та результати аналізу;
- підготовка звітів для моніторингу кіберзагроз.

Аналітичний модуль (analytics_module)

Аналітичний модуль проводить статистичний аналіз заблокованих та підозрілих запитів. Він визначає найпоширеніші типи атак та аномальну активність, що допомагає у побудові системи сповіщень та підтримці прийняття рішень. А саме:

- статистичний аналіз заблокованих запитів;
- виявлення типових атак та аномальної активності;
- підготовка даних для системи сповіщень.

Обробка помилок і моніторинг

Система має централізоване логування помилок і механізми винятків (try-except) для оперативного виявлення проблем. Це забезпечує стабільність роботи серверної частини та інформування про критичні збої під час виконання запитів.

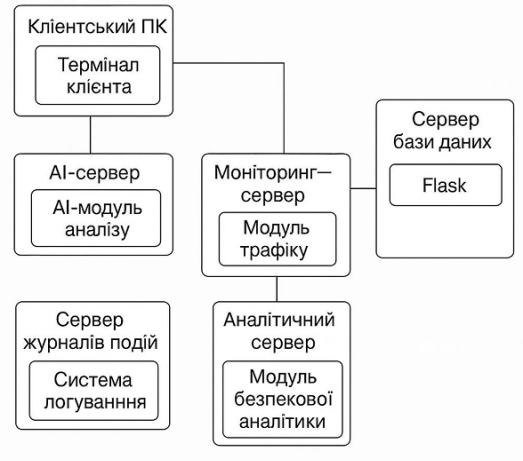


Рисунок 4.5 - Діаграма компонентів

Розглянемо діаграму компонентів, що зображена на рисунку 4.4 та яка, демонструє спосіб логічного розподілення систми на модулі: фільтрація SQL-запитів, AI-модуль, система логування, аналітика загроз. Це відображає гнучку, модульну архітектуру, яка дозволяє масштабувати систему, оперативно замінювати або оновлювати окремі частини без ризику втрати функціональної цілісності. Така структура — ключ до забезпечення стійкості в умовах загроз. Крім того, така побудова системи спрощує інтеграцію нових сервісів та модулів без необхідності суттєвих змін базової архітектури, що підвищує адаптивність системи.

Висновки до розділу 4

У четвертому розділі було здійснено детальний опис програмної реалізації системи забезпечення функціональної стійкості кіберзахисту СКБД, що дозволило послідовно показати шлях від загальної архітектури до конкретних програмних модулів. Запропонована архітектура застосунку побудована за клієнт-серверною моделлю з виокремленням аналітичного та інтелектуального рівнів, що забезпечує модульність, масштабованість і можливість роботи в режимі реального часу. Розмежування клієнтського, серверного та аналітичного рівнів створює чітку структуру відповідальності, спрощує супровід системи та дає змогу гнучко розширювати її функціонал.

Концептуальна модель системи описує взаємодію трьох ключових модулів – модуля реального часу, модуля ML-аналізу та модуля журналів атак. Така побудова дає змогу розділити первинну фільтрацію SQL-запитів, інтелектуальну класифікацію загроз і довгострокове зберігання аналітичних даних, що в сукупності формує єдиний захисний контур. Додаткова діаграма прецедентів конкретизує сценарії роботи користувача з системою – від формування та перевірки запитів до моделювання атак і перенавчання ML-моделі – і демонструє практичну реалізацію механізмів кіберзахисту в реальних умовах експлуатації.

Логічна структура програмної системи, реалізована у вигляді UML-діаграми класів, показує чітко організовану взаємодію між класами SQLRuleParser, MLChecker, DBConnectionManager, SchemaExplorer, DatasetBuilder та координаційним компонентом. Такий підхід забезпечує ізоляцію відповідальності, повторне використання коду та можливість незалежної модифікації окремих модулів без порушення загальної працездатності системи. Особливе значення має поєднання сигнатурного аналізу SQL-запитів із машинним навчанням, що підвищує точність і адаптивність виявлення загроз.

Серверна частина застосунку, побудована на базі Python/Flask та SQLAlchemy, інтегрує модулі перевірки SQL-запитів, ML-аналізу, безпечного виконання запитів, автентифікації та авторизації, логування й аналітики. Діаграма компонентів відображає логічний поділ системи на окремі функціональні блоки – фільтрацію SQL-запитів, AI-модуль, систему логування та модуль аналітики загроз. Така модульна організація полегшує супровід і розширення системи, забезпечує можливість заміни чи оновлення окремих компонентів та сприяє інтеграції із зовнішніми сервісами й інструментами кібербезпеки.

У підсумку, розділ 4 демонструє, що розроблена програмна реалізація відповідає поставленій меті – створити гнучку, інтелектуальну та функціонально стійку систему кіберзахисту СКБД. Поєднання продуманої архітектури, чіткої логічної структури, модульності серверної частини й інтеграції механізмів машинного навчання забезпечує здатність системи ефективно виявляти, класифікувати та документувати кібератаки, а також адаптуватися до змінних умов загроз.

РОЗДІЛ 5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1. Клієнська частина застосунку

Клієнтська частина відповідає за взаємодію користувача з системою через веб-інтерфейс. Вона забезпечує зручний та безпечний спосіб відправки SQL-запитів на сервер, отримання результатів їх виконання та відображення аналітики.

Клієнтська частина побудована на базі сучасних веб-технологій (HTML, CSS, JavaScript) і використовує AJAX-запити для асинхронної взаємодії з сервером.

Інтерфейс користувача (Frontend UI)

Інтерфейс надає користувачу доступ до основних функцій системи, включаючи відправку SQL-запитів, перегляд результатів, сповіщення про підозрілі або заблоковані запити, а також аналітичні звіти. Дизайн орієнтований на простоту та зручність, щоб користувач міг швидко оцінювати безпеку своїх запитів. А саме:

- відправка SQL-запитів на сервер через REST API;
- відображення результатів виконання запитів у табличному форматі;
- показ повідомлень про заблоковані або підозрілі запити;
- відображення статистики та аналітичних звітів.

Взаємодія з сервером (AJAX/Fetch)

Клієнтська частина використовує асинхронні запити (AJAX, Fetch API) для обміну даними з сервером без перезавантаження сторінки. Це дозволяє оперативно отримувати результати виконання SQL-запитів та оновлювати інформацію про безпеку в реальному часі. А саме:

- асинхронне відправлення SQL-запитів на сервер;
- отримання JSON-відповідей і відображення їх на сторінці;
- обробка помилок і повідомлень про блокування запитів.

Валідація введення

На клієнтській стороні реалізована базова перевірка даних перед відправкою на сервер, щоб зменшити кількість некоректних або потенційно шкідливих запитів. Це забезпечує додатковий рівень безпеки та покращує зручність роботи користувача.

А саме:

- Перевірка порожніх або некоректних запитів.
- Фільтрація очевидно небезпечних символів (напр., коментарі, спецсимволи).
- Підказки користувачу щодо формату запитів.

Візуалізація та аналітика

Клієнтська частина забезпечує відображення результатів виконання SQL-запитів та аналітичної інформації у вигляді таблиць і графіків. Це дозволяє користувачу швидко оцінювати ризики та ефективність системи кіберзахисту, контролювати поточний стан безпеки, аналізувати статистику атак і виявляти підозрілу активність у режимі реального часу. Крім того, інтерфейс клієнтської частини забезпечує зручну взаємодію з користувачем, надає доступ до журналів подій, результатів перевірки запитів та звітів про виявлені загрози, що підвищує оперативність прийняття рішень щодо реагування на інциденти інформаційної безпеки. А саме:

- табличне відображення результатів запитів;
- графічні діаграми для аналізу підозрілих та заблокованих запитів;
- відображення ML-score та причин блокування запитів.

Адаптивний дизайн та кросбраузерність

Інтерфейс розроблено з урахуванням адаптивності для різних пристроїв і кросбраузерної сумісності. Це дозволяє користувачу працювати з системою як на десктопах, так і на мобільних пристроях без втрати функціональності.

А саме:

- адаптація макету під різні розміри екранів;
- сумісність з основними браузерами (Chrome, Firefox, Edge, Safari);
- використання сучасних CSS-фреймворків для покращення UX/UI.

Таким чином, клієнтська частина застосунку виконує важливу роль у забезпеченні ефективної взаємодії користувача із системою кіберзахисту СКБД. Вона забезпечує зручне введення SQL-запитів, оперативне отримання результатів їх перевірки та виконання, а також відображення аналітичної інформації щодо рівня загроз. Реалізація асинхронної взаємодії з сервером, механізмів валідації введених даних та аналітичної візуалізації.

Робота користувача з програмною системою організована таким чином, щоб забезпечити інтуїтивну взаємодію та високий рівень безпеки при виконанні SQL-запитів. Користувач розпочинає роботу через веб-інтерфейс, проходить автентифікацію та отримує доступ до основного робочого простору. У спеціальному полі він може вводити SQL-запити, які система автоматично перевіряє на безпечність за допомогою вбудованих правил і алгоритмів машинного навчання. Після аналізу запит або виконується, або блокується. Результати виконання відображаються у зручному табличному форматі. Усі дії користувача, зокрема надсилання, перевірка й виконання запитів, фіксуються у системі логування, що дає змогу проводити аналіз активності та оцінювати потенційні кіберзагрози.

Основні етапи роботи:

- авторизація користувача;
- введення SQL-запитів;
- перевірка безпечності;
- отримання результатів;
- логування дій.

5.2. Графічний інтерфейс користувача програмної системи

1. ГОЛОВНА СТОРІНКА ВЕБ-ЗАСТОСУНКУ «DB CYBERSHIELD LAB»

Головна сторінка веб-застосунку «DB CyberShield Lab» призначена для ознайомлення користувача з функціональними можливостями системи та забезпечує початкову точку доступу до основних режимів роботи лабораторії кіберзахисту баз даних. Вона виконує інформаційну, навігаційну та презентаційну функції, формуючи загальне уявлення про призначення системи.

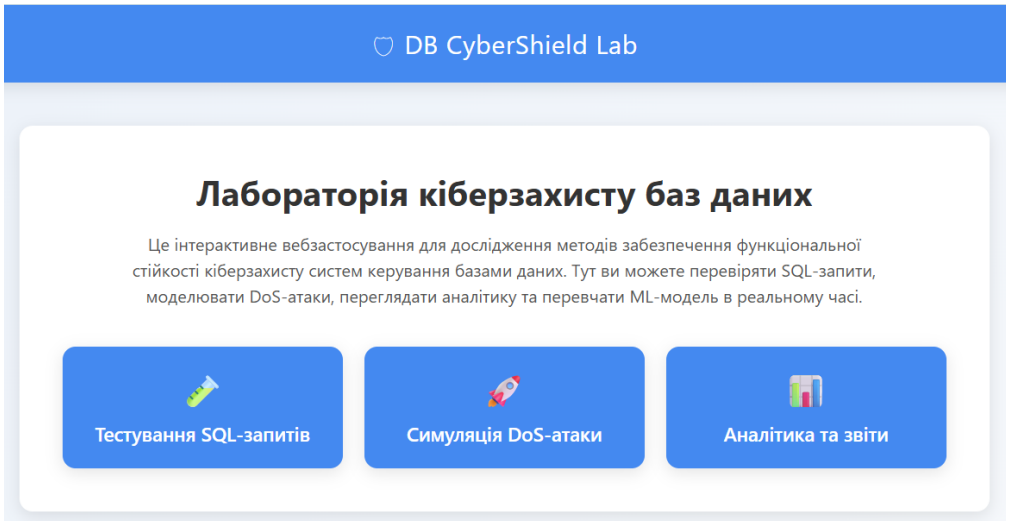


Рисунок 5.1 – Головний екран

У центральній частині сторінки подано заголовок, що відображає назву лабораторії кіберзахисту баз даних, а також короткий опис її призначення. З цього тексту користувач отримує уявлення про те, що система є інтерактивним веб-застосунком для дослідження методів забезпечення функціональної стійкості кіберзахисту СКБД. Також акцент зроблено на ключових можливостях: перевірці SQL-запитів, моделюванні DoS-атак, перегляді аналітики та перенавчанні ML-моделі в реальному часі. Таким чином, уже на початковому етапі користувач розуміє, що система поєднує як засоби тестування безпеки, так і інструменти аналітики та машинного навчання.

Нижче розміщено три функціональні кнопки, які забезпечують перехід до основних підсистем застосунку. Перша кнопка відповідає за тестування SQL-запитів і слугує для переходу до режиму перевірки запитів на наявність ознак SQL-

ін’єкцій та інших загроз. Друга кнопка використовується для запуску режиму симуляції DoS-атак, що дозволяє досліджувати поведінку системи під навантаженням. Третя кнопка відкриває підсистему аналітики та звітів, де користувач може переглядати статистику атак, журнали подій та результати роботи ML-модуля.

Композиція сторінки побудована за принципами мінімалізму та зрозумілої ієрархії інформації. Відсутність перевантаження деталями сприяє швидкій орієнтації користувача та зменшує когнітивне навантаження. Головна сторінка, таким чином, виконує роль навігаційного центру всієї системи та забезпечує зручний доступ до її ключових функціональних модулів.

2. ЕКРАН ОЦІНЮВАННЯ ЯКОСТІ ML-МОДЕЛІ (ROC AUC, PRECISION, RECALL)

ROC AUC: 0.9855

	precision	recall	f1-score	support
0	0.0000	0.0000	0.0000	1
1	0.9942	0.9942	0.9942	172
accuracy			0.9884	173
macro avg	0.4971	0.4971	0.4971	173
weighted avg	0.9884	0.9884	0.9884	173

Рисунок 5.2 – Оцінка якості ML-моделі

Представлений екран відображає результати оцінювання ефективності роботи моделі машинного навчання, яка використовується в системі для класифікації SQL-запитів за рівнем небезпеки. На цьому етапі система демонструє показники, що характеризують якість побудованого класифікатора.

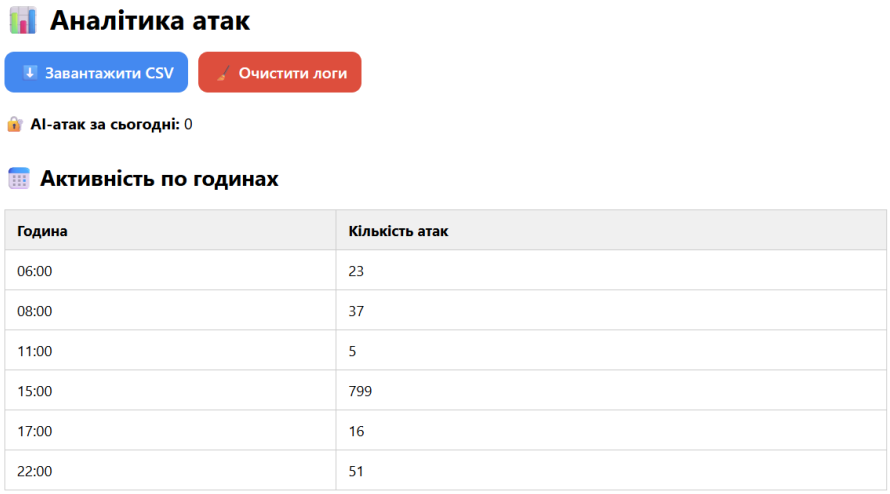
Основним інтегральним показником є ROC AUC, який дозволяє оцінити здатність моделі відокремлювати шкідливі запити від безпечних. Високе значення цього коефіцієнта свідчить про добру узагальнювальну здатність моделі та її ефективність у задачах бінарної класифікації. Окрім цього, на екрані подано

стандартний звіт класифікації з показниками precision, recall та f1-score для кожного класу.

Показник precision характеризує точність передбачення шкідливих запитів, тобто частку коректно визначених загроз серед усіх виявлених моделлю. Recall відображає повноту виявлення атак і показує, яка частина реальних шкідливих запитів була правильно знайдена системою. F1-score є гармонійним середнім між precision та recall і дозволяє комплексно оцінити баланс між точністю і повнотою. Також відображається загальна точність (accuracy), яка демонструє частку правильних класифікацій серед усіх перевірених запитів. Додатково подано макро- та зважені середні значення, що дозволяє оцінити роботу моделі з урахуванням дисбалансу класів.

Таким чином, даний екран забезпечує користувача повною аналітичною інформацією про якість навченої ML-моделі та дозволяє робити висновки щодо доцільності її подальшого використання або необхідності перенавчання.

3. ЕКРАН «АНАЛІТИКА АТАК» ТА СТАТИСТИКА АКТИВНОСТІ



Година	Кількість атак
06:00	23
08:00	37
11:00	5
15:00	799
17:00	16
22:00	51

Рисунок 5.2 – Аналітика атак

Даний екран призначений для перегляду результатів роботи системи кіберзахисту в частині фіксації та аналізу атак. Він реалізує функцію моніторингу стану безпеки в режимі реального часу.

У верхній частині розміщені керуючі елементи для роботи з журналами подій. Користувач може завантажувати лог-файли у форматі CSV для подальшого аналізу або очищувати базу журналів. Це забезпечує зручність у роботі з великими обсягами накопичених даних та підтримує можливість інтеграції з іншими аналітичними системами.

Окремо відображається кількість AI-атак, виявлених за поточну добу. Цей показник є важливим індикатором поточного рівня загроз і дозволяє оперативно оцінювати активність атакуючих. Нижче наведено таблицю активності по годинах, у якій показано розподіл кількості атак у часовому розрізі. Такий підхід дає змогу виявляти пікові періоди навантаження, аналізувати закономірності атак та прогнозувати ймовірні часові інтервали підвищеного ризику.

Таким чином, модуль аналітики реалізує функції оперативного та стратегічного аналізу кіберінцидентів, що є необхідним елементом систем забезпечення функціональної стійкості кіберзахисту.

4. ЕКРАН ПІДКЛЮЧЕННЯ ДО БАЗИ ДАНИХ ТА ГЕНЕРАЦІЇ ДАТАСЕТУ

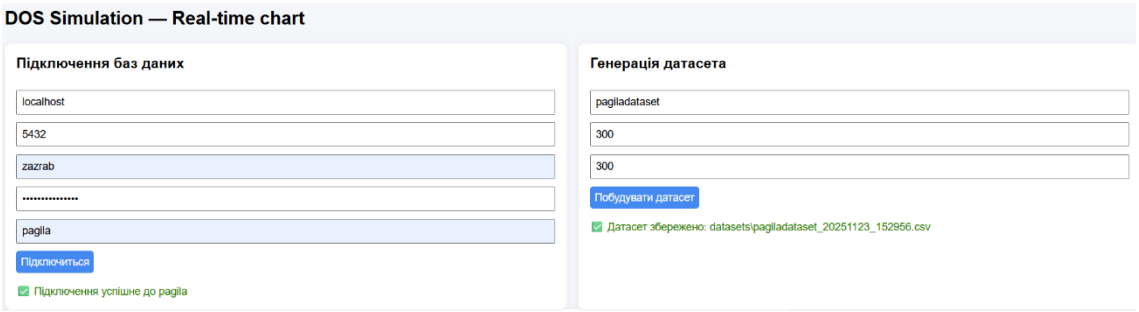


Рисунок 5.3 – Підключення БД

Цей екран забезпечує функціональні можливості інтеграції системи з реальною базою даних та формування навчальних вибірок для машинного навчання.

У лівій частині реалізовано форму підключення до СКБД, де користувач вводить параметри з’єднання: адресу сервера, порт, ім’я користувача, пароль та назву бази даних. Наявність повідомлення про успішне підключення підтверджує коректність налаштування з’єднання та готовність системи до роботи з реальними даними.

У правій частині інтерфейсу представлено модуль генерації датасету. Користувач має змогу задати ім’я датасету та кількість записів для формування навчальної вибірки. Після запуску процесу система автоматично створює файл набору даних і повідомляє про успішне збереження.

Даний функціональний блок відіграє ключову роль у підготовці навчальних даних для ML-модуля та забезпечує можливість оперативного оновлення вибірок відповідно до поточних умов експлуатації системи.

5. ЕКРАН СИМУЛЯЦІЇ DOS-АТАК У РЕЖИМІ РЕАЛЬНОГО ЧАСУ

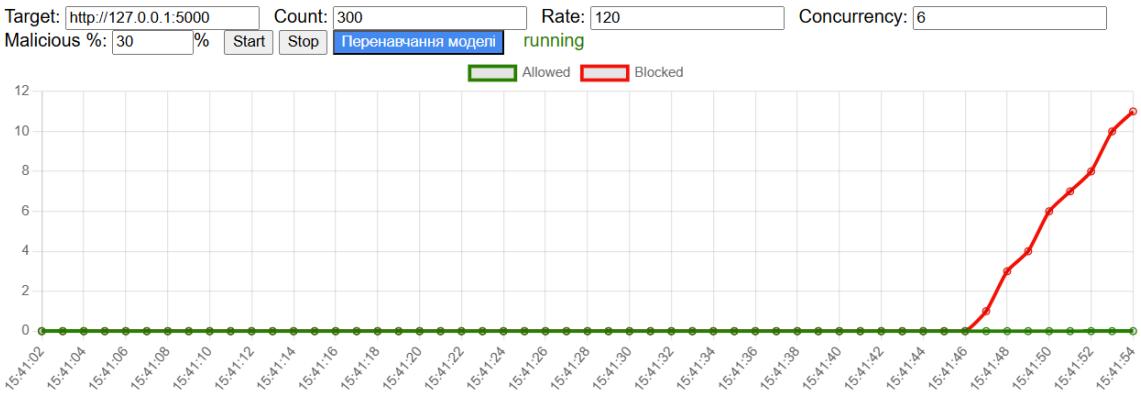


Рисунок 5.4 – Динамічний графік

Екран симуляції DoS-атак реалізує експериментальний режим роботи системи, призначений для перевірки її стійкості до масових запитів та навантаження. Користувач задає параметри імітації атаки, зокрема адресу цільового сервера, кількість запитів, швидкість їх надсилання, рівень конкурентності та відсоток шкідливих запитів. Також передбачена можливість запуску та зупинки процесу симуляції у ручному режимі.

У центральній частині відображається графік у режимі реального часу, який демонструє кількість дозволених та заблокованих запитів. Зелена крива відображає успішно дозволені запити, тоді як червона — заблоковані як потенційно шкідливі. Такий спосіб візуалізації дозволяє миттєво оцінювати ефективність системи фільтрації та швидкість реагування на атаку.

Цей модуль є важливим інструментом експериментального дослідження функціональної стійкості системи кіберзахисту та підтверджує її здатність працювати в умовах інтенсивного навантаження.

Аналіз динаміки графіків дозволяє оцінити пропускну здатність системи в умовах інтенсивного навантаження. За характером зміни кривих визначається стабільність роботи системи під час проведення DoS-атаки. Співвідношення дозволених та заблокованих запитів демонструє ефективність механізмів фільтрації і блокування шкідливого трафіку в реальному часі.

Висновки до розділу 5

У п’ятому розділі було розглянуто практичні аспекти роботи користувача з програмною системою кіберзахисту СКБД, зокрема організацію клієнтської частини веб-застосунку та особливості взаємодії з графічним інтерфейсом. Показано, що клієнтська частина забезпечує не лише зручне введення та надсилання SQL-запитів, але й їх прозору перевірку, відображення результатів аналізу та доступ до аналітичної інформації щодо поточного стану безпеки. Використання асинхронної взаємодії з сервером (AJAX/Fetch), валідації введення та візуалізації результатів у вигляді таблиць і графіків дозволяє забезпечити інтуїтивний та водночас безпечний режим роботи для користувача.

Окремо було проаналізовано низку ключових екранів системи: головну сторінку, модуль оцінювання якості ML-моделі, інтерфейс аналітики атак, екран підключення до бази даних і генерації датасету, а також модуль симуляції DoS-атак. Разом вони формують цілісне середовище для експериментальної перевірки функціональної стійкості системи, дозволяючи користувачу не лише працювати із запитами, а й спостерігати наслідки атак, якість класифікації, динаміку загроз та якість навчання моделі. Важливим є те, що інтерфейс орієнтований одночасно на прикладне використання (перевірка запитів) і на дослідницькі задачі (аналіз логів, генерація датасетів, оцінка ROC AUC, precision, recall тощо).

Таким чином, у розділі 5 продемонстровано, що розроблена клієнтська частина не є лише «оболонкою» для серверної логіки, а виступає повноцінним інструментом дослідження і керування кіберзахистом СКБД. Вона забезпечує користувачу можливість у єдиному інтерфейсі формувати SQL-запити, контролювати їхню безпечність, аналізувати статистику атак, перевіряти якість ML-моделі та моделювати навантаження у вигляді DoS-атак. Це підтверджує відповідність програмної системи поставленій меті роботи: поєднати науково обґрунтовані підходи до кіберзахисту з практичною.

ВИСНОВКИ

У магістерській роботі розв’язано актуальне науково-практичне завдання з розробки методики забезпечення функціональної стійкості кіберзахисту систем керування базами даних (СКБД) в умовах сучасних кіберзагроз. Актуальність дослідження зумовлена зростанням кількості кібератак на бази даних, ускладненням їхніх сценаріїв, а також необхідністю забезпечення безперервної та безпечної роботи інформаційних систем у критично важливих сферах.

Метою даної магістерської роботи було створення методики забезпечення функціональної стійкості кіберзахисту систем керування базами даних на основі аналізу сучасних загроз, вразливостей та методів протидії кібератакам, а також розробка програмного засобу для моніторингу, виявлення та протидії атакам на рівні СКБД.

Для досягнення поставленої мети в роботі було послідовно розв’язано комплекс науково-практичних завдань.

У ході дослідження:

- проаналізовано сучасні загрози та вразливості систем керування базами даних, зокрема SQL-ін’єкції, несанкціонований доступ, DoS-атаки, внутрішні загрози та багатоетапні кібератаки;
- виконано огляд та аналіз існуючих методів і засобів кіберзахисту СКБД, визначено їхні переваги та обмеження;
- розроблено багаторівневу архітектуру системи кіберзахисту, яка поєднує модулі моніторингу, аналізу, логування, машинного навчання та безпечного виконання SQL-запитів;
- реалізовано механізм парсингу та сигнатурної фільтрації SQL-запитів для виявлення небезпечних конструкцій на основі правил безпеки;
- інтегровано модуль машинного навчання (TF-IDF + Logistic Regression) для інтелектуальної класифікації SQL-запитів та оцінки рівня їх небезпечності;

- впроваджено sandbox-режим виконання SQL-запитів, який забезпечує безпечну перевірку підозрілих запитів без ризику пошкодження даних;
- створено систему логування, аудиту та аналітики, що дозволяє відстежувати події безпеки, накопичувати статистику атак та проводити їх подальший аналіз;
- розроблено клієнтську частину веб-застосунку для взаємодії користувача із системою, візуалізації результатів перевірки запитів та аналітичних показників;
- реалізовано модуль симуляції DoS-атак для експериментальної перевірки функціональної стійкості системи в умовах інтенсивного навантаження;
- проведено тестування працездатності розробленої системи на різних типах SQL-запитів та сценаріях атак.

У результаті виконання цієї магістерської роботи **поставлену мету досягнуто повністю**. Розроблено та реалізовано програмний прототип системи кіберзахисту СКБД, який забезпечує:

- виявлення потенційно небезпечних SQL-запитів;
- адаптивну оцінку рівня загроз на основі машинного навчання;
- безпечне виконання запитів у контрольованому середовищі;
- журналювання та аналітику подій безпеки;
- експериментальну перевірку стійкості системи до DoS-атак.

Практичне значення отриманих результатів полягає у можливості використання розробленої методики та програмного засобу для підвищення рівня кіберзахисту реальних систем керування базами даних, зокрема в інформаційних системах критичної інфраструктури. Запропонований підхід може бути використаний як основа для створення промислових рішень у сфері кібербезпеки, а також для навчальних і дослідницьких цілей.

Таким чином, розроблена методика та програмний комплекс підтверджують доцільність поєднання класичних методів захисту з інтелектуальними технологіями машинного навчання для забезпечення функціональної стійкості кіберзахисту СКБД та відповідають сучасним вимогам інформаційної безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лукова-Чайко Н. В. Метолологічні основи забезпечення функціональної стійкості розподілених інформаційних систем кібернетичних загроз: дис. …д-ра тех. наук : 05.13.06. Київ, 2018. 323 с.

2. Березовська Ю. В. Методика забезпечення функціональної стійкості інформаційних систем на основі програмно-конфігуративних мереж: дис. …д-ра тех. Наук : 621.396.4. Київ, 2021. 140 с.

3. Мохор В. В, Цуркан В. В, Дорогий Я. Ю, та Штифурак Ю. М, “Архітектуризування системи управління інформаційною безпекою”, на VI міжнародній науково-практичній конференції Актуальні питання забезпечення кібербезпеки та захисту інформації. Київ, 2020. С. 82-84.

4. Бабійчук Д, та Турти М, “Дослідження системи управління інформаційною безпекою судна на мережі Петрі”, на 7-й Міжнародній науково-технічної конференції Інформаційні системи та технології, Харків, 2018. 395 с.

5. Howard E. Python for cybersecurity. Using Python for a cyber offense and defense. Poston 3. New Jersey : John Wiley & Sons, 2022. 243 p.

6. Seitz J., Arnold T. Black hat Python. Python programming for hackers and pentesters. San Francisko: No starch press, 2021. 276 p.

7. Valentine T. Dtabase-Driven Web Development. Second edition. Selkirk, Canada: MB, 2023. 214 p.

8. Sherman Shen X. Blockchain-Based Data Security in Heterogeneous Communications Networks. Waterloo, Canada: Springer, 2024. 236 p.

9. Roeckl A. Cybersecurity in the Age of Artificial Intelligence — Safeguarding AI Systems.Meduum.URL: https://medium.com/@RocketMeUpCybersecurity/cybersecurity-in-the-age-of artificial-intelligence-safeguarding-ai-systems-f19fbb9ce9e5

10. Andrios R. 10 Real-World Examples of Database Automation Done Right. URL: https://hoop.dev/blog/10-real-world-examples-of-database-automation-done-right/

11. Raza M. Database Automation Explained: Concepts & Best Practices. URL: https://www.bmc.com/blogs/database-automation/

12. Red E. How AI Transforms Cybersecurity. Medium. URL: https://medium.com/@earlred/how-ai-transforms-cybersecurity-a6edf9b7099a

13. Katz E. Top 10 Security Automation Tools. URL: https://spectralops.io/blog/top 10-security-automation-tools/

14. Marbilia Possagnolo Sergio; Talita de Souza Costa; Marcelo S. de Paula Pessoa; Paulo S. M. Pedro. A Semantic Approach to Support the Analysis of Abstracts in a Bibliographical Review. 2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). 15.08.2019. URL: https://ieeexplore.ieee.org/document/8795382

15. Коломицев М., Носок С., и Тоцький Р., "Порівнювальний аналіз моделей оцінки зрілості інформаційної безпеки", Захист інформації, т. 21, no. 4, с. 224- 232, URL: http://dx.doi.org/10.18372/2410-7840. 21.14337.

ДОДАТОК А Фрагмент листінгу програми

#app.py#

```
import traceback
from flask import Flask, render_template
from sqlalchemy import create_engine
from analytics_routes import analytics_bp
from auth_module import auth_bp
from config_db import DATABASE_URI, SECRET_KEY
from dataset_routes import dataset_bp
from db_routes import db_bp
from log_module import log_bp
from ml_routes import ml_bp
from tester_routes import tester_bp
try:
    from realtime_routes import realtime_bp

    _HAS_REALTIME = True
    except Exception as exc:
    realtime_bp = None
    _HAS_REALTIME = False
    print("⚠ realtime_routes не импортирован:", exc)
    traceback.print_exc()
app = Flask(__name__)
app.secret_key = SECRET_KEY
engine = create_engine(DATABASE_URI)
if _HAS_REALTIME and realtime_bp is not None:
    try:
        app.register_blueprint(realtime_bp)
        print("realtime_bp registered")
    except Exception as exc:
        print("Не вдалось зареєструватися realtime_bp:", exc)
        traceback.print_exc()
else:
    print("ℹ Пропущена регистрация realtime_bp")
app.register_blueprint(db_bp)
app.register_blueprint(auth_bp)
app.register_blueprint(ml_bp)
app.register_blueprint(tester_bp)
app.register_blueprint(analytics_bp)
app.register_blueprint(log_bp)
app.register_blueprint(dataset_bp)
@app.route("/")
def index():
    return render_template("index.html")
@app.route("/dos_sim")
def dos_sim():
    return render_template("dos_sim.html")
if __name__ == "__main__":
```

```
    app.run(debug=True)

#auth_module.py#
from flask import Blueprint, redirect, render_template, request, session, url_for
from decorators import login_required
auth_bp = Blueprint("auth", __name__)
USERS = {
    "devops": {"password": "1234", "role": "admin"},
    "analyst": {"password": "5678", "role": "viewer"},
    "guest": {"password": "0000", "role": "guest"},
}
@auth_bp.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form.get("username")
        password = request.form.get("password")
        user = USERS.get(username)
        if user and user["password"] == password:
            session["logged_in"] = True
            session["username"] = username
            session["role"] = user["role"]
            next_url = session.pop("next_url", None)
            if next_url:
                return redirect(next_url)
            return redirect(url_for("log.view_logs"))
        return render_template(
            "login.html",
            error="Невірний логін або пароль",
        )
    return render_template("login.html")
@auth_bp.route("/logout")
@login_required
def logout():
    session.clear()
    return redirect(url_for("auth.login"))

#analytics_routes.py#
from collections import Counter
from datetime import datetime
from flask import Blueprint, render_template
from sqlalchemy.orm import joinedload
from analytics_module import AnalyticsSession, AttackLog
from decorators import login_required
analytics_bp = Blueprint("analytics", __name__)

@analytics_bp.route("/analytics")
@login_required
def analytics():
    with AnalyticsSession() as db:
        logs = (
```

```
    db.query(AttackLog)
    .options(joinedload(AttackLog.attack_type))
    .order_by(AttackLog.timestamp.desc())
    .all()
)

ai_count_today = 0
hourly_counter = Counter()
dangerous_queries = []

for log in logs:
    try:
        ts = datetime.strptime(log.timestamp, "%Y-%m-%d %H:%M:%S")

        hour = ts.strftime("%H:00")
        hourly_counter[hour] += 1

        if (
            log.reason.upper().startswith("AI")
            and ts.date() == datetime.today().date()
        ):
            ai_count_today += 1
            dangerous_queries.append(
                {
                    "timestamp": ts,
                    "reason": log.reason,
                    "query": log.query,
                    "attack_type": getattr(
                        log.attack_type,
                        "code",
                        "-",
                    ),
                    "score": log.score,
                }
            )
    except Exception:
        continue

top_queries = sorted(
    dangerous_queries,
    key=lambda x: x["timestamp"],
    reverse=True,
)[:5]

for query in top_queries:
    query["timestamp"] = query["timestamp"].strftime(
        "%Y-%m-%d %H:%M:%S"
    )
    query["score"] = (
        f"{query['score']:.2f}"
```

```
            if query["score"] is not None
            else "-"
        )

    return render_template(
        "analytics.html",
        ai_today=ai_count_today,
        hourly_data=sorted(hourly_counter.items()),
        top_queries=top_queries,
    )

#dataset_builder.py#
import csv
import os
import random
from datetime import datetime

from .schema_explorer import get_table_schema, list_tables


SAFE_SELECT_TEMPLATES = [
    "SELECT * FROM {table} LIMIT {limit};",
    "SELECT {column} FROM {table} WHERE {column} IS NOT NULL LIMIT {limit};",
    "SELECT COUNT(*) FROM {table};",
    "SELECT DISTINCT {column} FROM {table};",
    (
        "SELECT {column}, COUNT(*) FROM {table} "
        "GROUP BY {column} ORDER BY COUNT(*) DESC LIMIT {limit};"
    ),
    "SELECT * FROM {table} WHERE {column} LIKE 'A%';",
    "SELECT * FROM {table} ORDER BY {column} DESC LIMIT {limit};",
    "SELECT * FROM {table} WHERE {column} BETWEEN 10 AND 100;",
    (
        "SELECT * FROM {table} WHERE {column} IN "
        "(SELECT {column} FROM {table} LIMIT 5);"
    ),
    (
        "SELECT {column}, AVG({column}) FROM {table} "
        "GROUP BY {column} HAVING AVG({column}) > 10;"
    ),
    (
        "SELECT * FROM {table} WHERE {column} = "
        "(SELECT MAX({column}) FROM {table});"
    ),
    "SELECT * FROM {table} WHERE {column} IS NULL;",
    (
        "SELECT * FROM {table} WHERE {column} > 0 "
        "ORDER BY {column} ASC LIMIT {limit};"
    ),
    "SELECT {column} FROM {table} WHERE {column} LIKE '%test%';",
```

```
]

SAFE_JOIN_TEMPLATES = [
    (
        "SELECT {t1}.{c1}, {t2}.{c2} FROM {t1} "
        "JOIN {t2} ON {t1}.{c1} = {t2}.{c2} LIMIT {limit};"
    ),
    (
        "SELECT COUNT(*) FROM {t1} LEFT JOIN {t2} "
        "ON {t1}.{c1} = {t2}.{c2};"
    ),
    (
        "SELECT {t1}.{c1}, {t2}.{c2} FROM {t1} "
        "INNER JOIN {t2} ON {t1}.{c1} = {t2}.{c2} "
        "WHERE {t1}.{c1} IS NOT NULL LIMIT {limit};"
    ),
    (
        "SELECT * FROM {t1} JOIN {t2} "
        "ON {t1}.{c1} = {t2}.{c2} "
        "WHERE {t1}.{c1} > 0 LIMIT {limit};"
    ),
]

SAFE_TEMPLATES = SAFE_SELECT_TEMPLATES + SAFE_JOIN_TEMPLATES

MALICIOUS_SYNTAX_TEMPLATES = [
    "SELEC FROM {table};",
    "SELECT * FORM {table};",
    "SELECT * FROM WHERE id = 1;",
    "SELECT * FROM {table} WHERE ;",
]

MALICIOUS_DDL_TEMPLATES = [
    "DROP TABLE {table};",
    "TRUNCATE TABLE {table};",
    "DELETE FROM {table};",
    "UPDATE {table} SET {column} = 'HACKED';",
    "INSERT INTO {table} VALUES (9999, 'Evil', 'Hacker');",
]

MALICIOUS_INJECTION_TEMPLATES = [
    "SELECT * FROM {table} WHERE {column} = '' OR '1'='1';",
    (
        "SELECT * FROM {table} WHERE {column} = '' "
        "UNION SELECT version(), current_user, now();--';"
    ),
    "SELECT * FROM {table}; DROP TABLE {table};--",
    (
        "SELECT * FROM {table} WHERE {column} = '' "
        "exec xp_cmdshell('dir'); --"
```

```
    ),
    (
        "SELECT * FROM {table} UNION "
        "SELECT username, password FROM users; --"
    ),
]

MALICIOUS_RUNTIME_TEMPLATES = [
    "SELECT 1/0;",
    "SELECT * FROM non_existing_table;",
    "SELECT fake_column FROM {table};",
]

MALICIOUS_TEMPLATES = (
    MALICIOUS_SYNTAX_TEMPLATES
    + MALICIOUS_DDL_TEMPLATES
    + MALICIOUS_INJECTION_TEMPLATES
    + MALICIOUS_RUNTIME_TEMPLATES
)

def _get_random_column(table):
    columns = get_table_schema(table)
    if not columns:
        return "id"
    return random.choice(columns)["name"]

def _get_two_distinct_tables():
    tables = list_tables()
    if len(tables) < 2:
        return tables[0], tables[0]
    return random.sample(tables, 2)

def generate_safe_queries(num_queries=300):
    queries = []
    tables = list_tables()
    if not tables:
        raise ValueError("У базі даних немає таблиць")

    for _ in range(num_queries):
        template = random.choice(SAFE_TEMPLATES)
        table = random.choice(tables)
        column = _get_random_column(table)
        limit = random.randint(1, 50)

        if "{t1}" in template:
            t1, t2 = _get_two_distinct_tables()
            c1 = _get_random_column(t1)
            c2 = _get_random_column(t2)
            query = template.format(
                t1=t1,
```

```
                t2=t2,
                c1=c1,
                c2=c2,
                limit=limit,
            )
        else:
            query = template.format(
                table=table,
                column=column,
                limit=limit,
            )

        queries.append((query, 0))

    return queries

def generate_malicious_queries(num_queries=300):
    queries = []
    tables = list_tables()
    if not tables:
        raise ValueError("У базі даних немає таблиць")

    for _ in range(num_queries):
        template = random.choice(MALICIOUS_TEMPLATES)
        table = random.choice(tables)
        column = _get_random_column(table)
        query = template.format(
            table=table,
            column=column,
        )
        queries.append((query, 1))

    return queries

def build_dataset(
    filename: str,
    num_safe=300,
    num_malicious=300,
    output_dir="datasets",
):
    os.makedirs(output_dir, exist_ok=True)
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    full_path = os.path.join(
        output_dir,
        f"{filename}_{timestamp}.csv",
    )

    safe_queries = generate_safe_queries(num_safe)
    malicious_queries = generate_malicious_queries(num_malicious)
```

```
    with open(
        full_path,
        mode="w",
        newline="",
        encoding="utf-8",
    ) as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["query", "label"])

        for query, label in safe_queries + malicious_queries:
            writer.writerow([query, label])

    print(f"Датасет збережено: {full_path}")
    print(
        f"{len(safe_queries)} безпечних + "
        f"{len(malicious_queries)} шкідливих"
    )

    return full_path

if __name__ == "__main__":
    build_dataset(
        filename="universal_dataset",
        num_safe=200,
        num_malicious=200,
    )
```

```
#db_connection.py#
import urllib.parse

from sqlalchemy import create_engine, text

from sqlalchemy.exc import SQLAlchemyError

_current_engine = None

_current_uri = None

def build_postgres_uri(

    host: str,

    port: int,

    user: str,

    password: str,

    dbname: str,

) -> str:

    user_enc = urllib.parse.quote_plus(user)

    password_enc = urllib.parse.quote_plus(password)

    return (
```

```
        f"postgresql+psycopg2://"
        f"{user_enc}:{password_enc}@{host}:{port}/{dbname}"
    )

def build_mysql_uri(
    host: str,
    port: int,
    user: str,
    password: str,
    dbname: str,
) -> str:
    user_enc = urllib.parse.quote_plus(user)
    password_enc = urllib.parse.quote_plus(password)
    return (
        f"mysql+pymysql://"
        f"{user_enc}:{password_enc}@{host}:{port}/{dbname}"
    )

def build_sqlite_uri(file_path: str) -> str:
    return f"sqlite:///{file_path}"

def connect_from_params(
    db_type: str,
    host: str | None = None,
    port: int | None = None,
    dbname: str | None = None,
    user: str | None = None,
    password: str | None = None,
    file_path: str | None = None,
):
    global _current_engine, _current_uri

    try:
        if db_type == "postgresql":
```

```
            uri = build_postgres_uri(host, port, user, password, dbname)
        elif db_type == "mysql":
            uri = build_mysql_uri(host, port, user, password, dbname)
        elif db_type == "sqlite":
            if not file_path:
                return {
                    "success": False,
                    "message": "Не вказано шлях до SQLite файлу",
                }
            uri = build_sqlite_uri(file_path)
        else:
            return {
                "success": False,
                "message": f"Невідомий тип БД: {db_type}",
            }

        engine = create_engine(uri, echo=False)

        with engine.connect() as conn:
            conn.execute(text("SELECT 1"))

        _current_engine = engine
        _current_uri = uri
        print(f"Підключено до БД ({db_type}) → {uri}")

        return {
            "success": True,
            "message": f"Підключено до {db_type} БД",
        }
    except SQLAlchemyError as exc:
        print(f"Помилка підключення: {exc}")
        _current_engine = None
        _current_uri = None
        return {
```

```
            "success": False,
            "message": str(exc),
        }

def connect_to_database(
    host: str,
    port: int,
    user: str,
    password: str,
    dbname: str,
):
    result = connect_from_params(
        db_type="postgresql",
        host=host,
        port=port,
        dbname=dbname,
        user=user,
        password=password,
    )
    return result["success"]

def get_engine():
    return _current_engine

def get_current_uri():
    return _current_uri

def disconnect():
    global _current_engine, _current_uri
    _current_engine = None
    _current_uri = None
    print("З'єднання з БД розірвано")
```

```
#schema_explorer.py#
from sqlalchemy import inspect
from .db_connection import get_engine


def list_tables():
    engine = get_engine()
    if engine is None:
        raise ConnectionError("Немає активного з'єднання з базою даних")

    inspector = inspect(engine)
    return inspector.get_table_names()


def get_table_schema(table_name: str):
    engine = get_engine()
    if engine is None:
        raise ConnectionError("Немає активного з'єднання з базою даних")

    inspector = inspect(engine)
    columns_info = inspector.get_columns(table_name)

    return [
        {
            "name": column["name"],
            "type": str(column["type"]),
        }
        for column in columns_info
    ]


def get_full_schema():
    tables = list_tables()
    full_schema = {}

    for table in tables:
        full_schema[table] = get_table_schema(table)

    return full_schema


if __name__ == "__main__":
    try:
        tables = list_tables()
        print("Таблиці:", tables)

        if tables:
            schema = get_table_schema(tables[0])
            print(
                f"Структура таблиці {tables[0]}:",
```

```
            schema,
        )
    except Exception as exc:
        print("Помилка:", exc)


#decorators.py#
from functools import wraps
from flask import abort, redirect, request, session, url_for


def login_required(view_func):
    @wraps(view_func)
    def wrapped_view(*args, **kwargs):
        if not session.get("logged_in"):
            session["next_url"] = request.url
            return redirect(url_for("auth.login"))

        return view_func(*args, **kwargs)

    return wrapped_view


def role_required(*roles):
    def decorator(view_func):
        @wraps(view_func)
        def wrapped_view(*args, **kwargs):
            if not session.get("logged_in"):
                return redirect(url_for("auth.login"))

            user_role = session.get("role")
            if user_role not in roles:
                return abort(403)

            return view_func(*args, **kwargs)

        return wrapped_view

    return decorator

#log_module.py#
from io import StringIO
from flask import Blueprint, Response, redirect, render_template, url_for
from sqlalchemy.orm import joinedload
from analytics_module import AnalyticsSession, AttackLog
from decorators import login_required


log_bp = Blueprint("log", __name__)
```

```
@log_bp.route("/logs")
@login_required
def view_logs():
    with AnalyticsSession() as db:
        logs = (
            db.query(AttackLog)
            .options(joinedload(AttackLog.attack_type))
            .order_by(AttackLog.timestamp.desc())
            .limit(100)
            .all()
        )

    return render_template("logs.html", logs=logs)


@log_bp.route("/logs/download")
@login_required
def download_logs():
    with AnalyticsSession() as db:
        logs = (
            db.query(AttackLog)
            .options(joinedload(AttackLog.attack_type))
            .order_by(AttackLog.timestamp.desc())
            .all()
        )

    output = StringIO()
    output.write(
        "timestamp,reason,attack_type,query,score,source_ip,status\n"
    )

    for log in logs:
        attack_type = log.attack_type.code if log.attack_type else "-"
        score_str = (
            f"{log.score:.2f}"
            if log.score is not None
            else "-"
        )
        output.write(
            f"{log.timestamp},"
            f"{log.reason},"
            f"{attack_type},"
            f"{log.query},"
            f"{score_str},"
            f"{log.source_ip},"
            f"{log.status}\n"
        )

    return Response(
```

```
        output.getvalue(),
        mimetype="text/csv",
        headers={
            "Content-Disposition": "attachment; filename=attack_logs.csv",
        },
    )


@log_bp.route("/logs/clear")
@login_required
def clear_logs():
    with AnalyticsSession() as db:
        db.query(AttackLog).delete()
        db.commit()

    return redirect(url_for("log.view_logs"))

#ml_checker.py#
import os
import pickle

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sqlalchemy import create_engine, text

from config_db import DATABASE_URI


ML_LOW_THRESHOLD = 0.5
ML_HIGH_THRESHOLD = 0.8


class MLChecker:
    MODEL_FILE = "ml_model.pkl"

    def __init__(self):
        self.vectorizer = TfidfVectorizer()
        self.model = LogisticRegression()
        self.engine = create_engine(DATABASE_URI)

        if os.path.exists(self.MODEL_FILE):
            with open(self.MODEL_FILE, "rb") as file:
                self.vectorizer, self.model = pickle.load(file)
        else:
            self.vectorizer = TfidfVectorizer()
            self.model = LogisticRegression()

    def predict(self, sql: str):
        try:
            x_vect = self.vectorizer.transform([sql])
```

```
            prob = self.model.predict_proba(x_vect)[0][1]
            is_suspicious = prob >= ML_LOW_THRESHOLD
            return is_suspicious, prob
        except Exception:
            return False, 0.0

    def sandbox_execute(self, sql: str):
        try:
            with self.engine.begin() as conn:
                result = conn.execute(text(sql))
                rows = [dict(row) for row in result]
            return rows
        except Exception as exc:
            return {"error": str(exc)}

    def train(self, x_train, y_train):
        x_vect = self.vectorizer.fit_transform(x_train)
        self.model.fit(x_vect, y_train)

        with open(self.MODEL_FILE, "wb") as file:
            pickle.dump((self.vectorizer, self.model), file)

#sql_parser.py#
import re

import sqlparse
from sqlparse.tokens import Comment, Keyword

from rule_checker import BLOCKED_COMMANDS, BLOCKED_TOKENS


def clean_token(value: str) -> str:
    return re.sub(r"[;(),]", "", value).strip().upper()


class SQLRuleParser:
    def __init__(self):
        self.blocked_commands = set(BLOCKED_COMMANDS)
        self.blocked_tokens = set(BLOCKED_TOKENS)

    def is_safe(self, sql: str) -> bool:
        result, _ = self.check_query(sql)
        return result

    def check_query(self, sql: str):
        try:
            parsed = sqlparse.parse(sql)
            if not parsed:
                return False, "Empty or invalid SQL"
```

```
            stmt = parsed[0]

            first_token = stmt.token_first(skip_cm=True)
            if first_token is None:
                return False, "No first token"

            first_value = clean_token(first_token.value)
            if first_value in self.blocked_commands:
                return False, f"Blocked command: {first_value}"

            for token in stmt.flatten():
                if token.ttype in (Keyword, Comment):
                    value = clean_token(token.value)
                    if value in self.blocked_tokens:
                        return False, f"Blocked token: {value}"

            return True, "Safe"

        except Exception as exc:
            return False, f"Error parsing SQL: {exc}"

#generate_sql_dataset.py#
import csv
import random


OUT = "ml_sql_dataset.csv"
N_SAFE = 600
N_MAL = 400

random.seed(42)


SAFE_TEMPLATES = [
    "SELECT {cols} FROM {table} WHERE {pk} = {val};",
    "SELECT {cols} FROM {table} LIMIT {n};",
    "SELECT {cols} FROM {table} ORDER BY {col} DESC LIMIT {n};",
    "SELECT {cols} FROM {table} WHERE {col} LIKE '%{term}%';",
    "SELECT COUNT(*) FROM {table};",
    (
        "SELECT a.{c1}, b.{c2} FROM {t1} a "
        "JOIN {t2} b ON a.{fk}=b.{pk} "
        "WHERE a.{col} > {num};"
    ),
]


TABLES = {
    "actor": ["actor_id", "first_name", "last_name", "last_update"],
    "customer": ["customer_id", "first_name", "last_name", "email"],
```

```
    "film": ["film_id", "title", "description", "release_year"],
    "payment": ["payment_id", "customer_id", "amount", "payment_date"],
    "users": ["id", "username", "email", "created_at"],
}


MAL_TEMPLATES = [
    "'; DROP TABLE {table}; --",
    "' OR '1'='1'; --",
    "' UNION SELECT {cols} FROM {table} --",
    "'; UPDATE {table} SET {col} = {val} WHERE {pk} = {val}; --",
    "'; INSERT INTO {table} ({cols}) VALUES ({vals}); --",
    "'; SELECT pg_sleep(10); --",
    "'; EXEC xp_cmdshell('dir'); --",
    "' OR 1=1; --",
    "'; DELETE FROM {table} WHERE {pk} = {val}; --",
]


def rnd_col(table):
    return random.choice(TABLES[table])


def rnd_cols(table, k=2):
    columns = random.sample(TABLES[table], min(k, len(TABLES[table])))
    return ", ".join(columns)


def gen_safe(count):
    output = []

    for _ in range(count):
        table = random.choice(list(TABLES.keys()))
        template = random.choice(SAFE_TEMPLATES)

        cols = rnd_cols(table, k=random.randint(1, 2))
        pk = TABLES[table][0]
        val = str(random.randint(1, 200))
        col = rnd_col(table)
        limit = random.randint(1, 50)
        term = random.choice(
            ["John", "Action", "2020", "Smith", "com"]
        )

        table_2 = random.choice(list(TABLES.keys()))
        c1 = rnd_col(table)
        c2 = rnd_col(table_2)
        fk = rnd_col(table)

        output.append(
```

```
            template.format(
                cols=cols,
                table=table,
                pk=pk,
                val=val,
                col=col,
                n=limit,
                term=term,
                t1=table,
                t2=table_2,
                c1=c1,
                c2=c2,
                fk=fk,
                num=limit,
            )
        )

    return output


def gen_mal(count):
    output = []

    for _ in range(count):
        table = random.choice(list(TABLES.keys()))
        template = random.choice(MAL_TEMPLATES)

        cols = rnd_cols(table, k=random.randint(1, 2))
        pk = TABLES[table][0]
        val = str(random.randint(1, 200))
        col = rnd_col(table)
        vals = ", ".join(
            f"'{random.choice(['x', 'admin', 'test'])}'"
            for _ in range(2)
        )

        output.append(
            template.format(
                table=table,
                cols=cols,
                pk=pk,
                val=val,
                col=col,
                vals=vals,
            )
        )

    return output
```

```
if __name__ == "__main__":
    rows = []
    rows.extend((query, 0) for query in gen_safe(N_SAFE))
    rows.extend((query, 1) for query in gen_mal(N_MAL))

    random.shuffle(rows)

    with open(
        OUT,
        "w",
        newline="",
        encoding="utf-8",
    ) as file:
        writer = csv.writer(file)
        writer.writerow(["sql", "label"])

        for sql, label in rows:
            writer.writerow([sql, label])

    print(f"Generated {len(rows)} rows -> {OUT}")
```

ДОДАТОК Б Презентація

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

Методика забезпечення функціональної стійкості кіберзахисту систем керування базами данних

Виконала:
Студентка групи ТВ-42мп
Плачинда Маргарита Володимирівна

Керівник:
доцент
Шуклін Герман Вікторович

м. Київ - 2025

Постановка задач

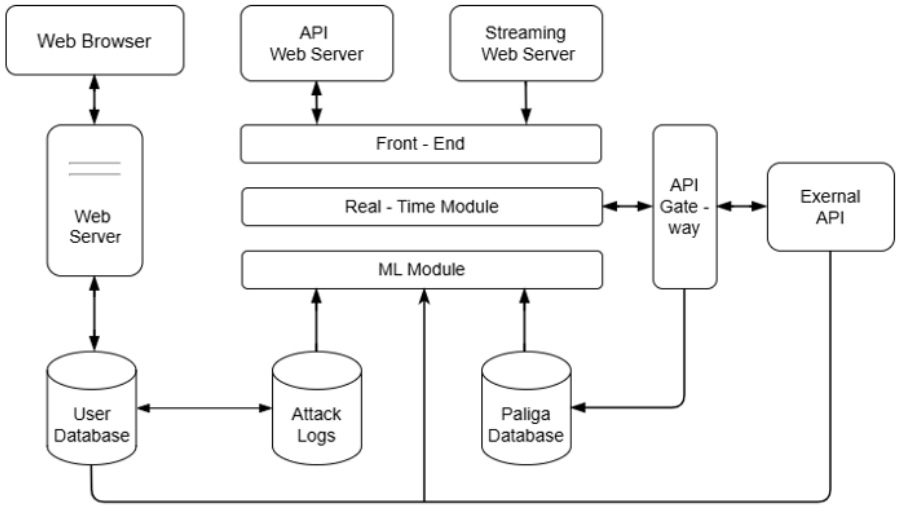
Мета роботи - розробка веб-застосунку для забезпечення функціональної стійкоскі кіберзахисту системи керування базами даних.

1) Проаналізувати існуючі методи кіберзахисту системи керування БД.
2) Розробити архітектури веб-застосунку.
3) Реалізувати модуль перевірки SQL-запитів на основі правил і ML-моделі.
4) Створити модуль симуляції атак та моніторингу в реальному часі.
5) Провести дослідження ефективності розробленої системи.

Опис предметної області

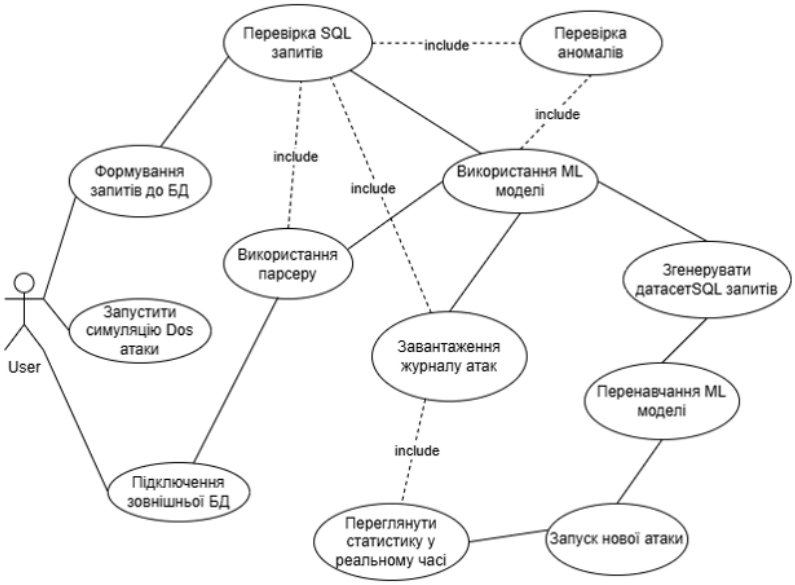
Предметна область охоплює процеси взаємодії користувача з системами керування базами даних у контексті аналізу й оцінювання SQL-запитів та поведінки бази даних під час навантаження. Центральним елементом є можливість підключення до зовнішніх реальних баз, отримання їхньої структури та автоматичне формування на основі цих даних різноманітних SQL-запитів, що створюють репрезентативні датасети. У межах цієї області розглядається комбінований механізм перевірки запитів, який включає синтаксичний аналіз за правилами й машинне навчання для класифікації нормальних та підозрілих операцій. Також важливим аспектом є реєстрація всіх виконаних запитів, їх подальший аналіз, а також відображення ключових показників роботи системи в режимі реального часу. Окремим процесом предметної області є адаптивне оновлення моделі класифікації на основі нових логів і згенерованих датасетів, що забезпечує здатність системи самонавчатися й підтримувати свою ефективність у змінних умовах.

Архітектура системи

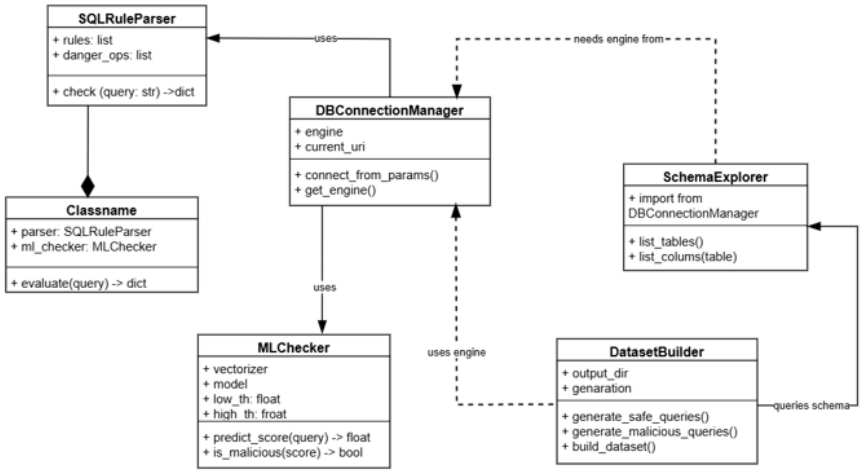


Концептуальна модель ситеми

Діаграма прецедентів



Uml діаграма



Засоби розробки

PyCharm

Flask

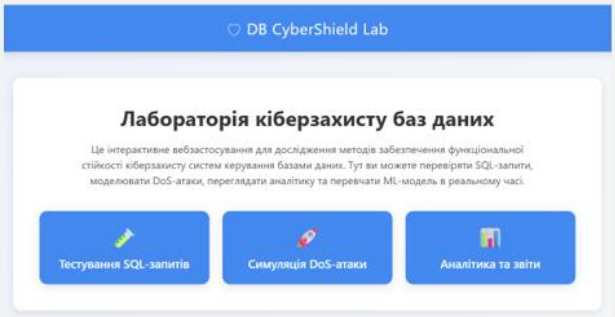
PostgreSQL

pandas

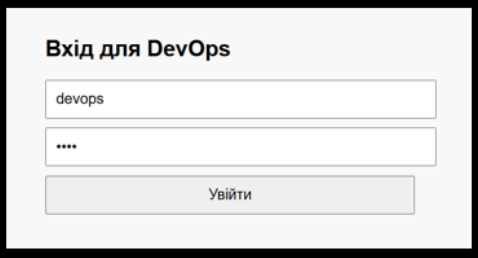
python™

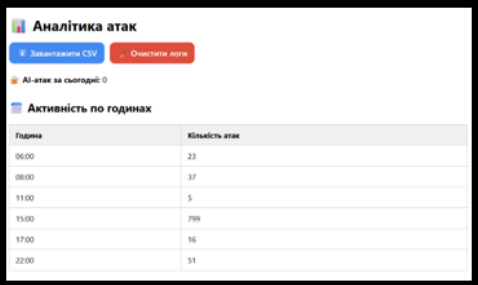
scikit learn

Інтерфейс застосунку



- Основні модулі:
  - Тестування SQL-запитів
  - Симуляція DoS-атак у реальному часі
  - Аналітика аномалій

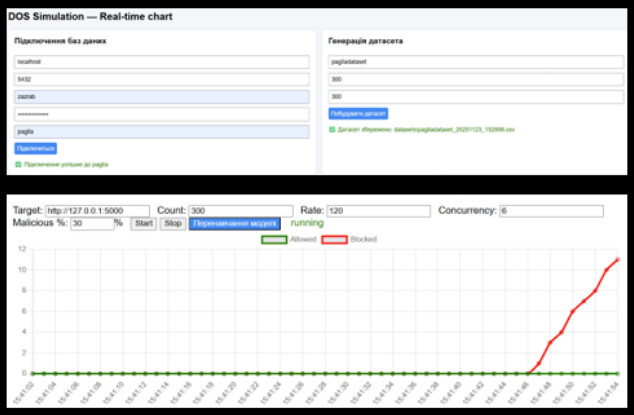




Інтерфейс застосунку

- Система вимагає перевірку доступу перед роботою з даними
- Система відображає журнал запитів та виявлених аномалій

Інтерфейс застосунку



- Підключення БД
- Генерація датасету на основі реальних SQL-запитів
- Відображення графіка обробки атак
- Можливість перенавчання ML-моделі

```
ROC AUC: 0.9855

              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000         1
           1     0.9942    0.9942    0.9942       172

    accuracy                         0.9884       173
   macro avg     0.4971    0.4971    0.4971       173
weighted avg     0.9884    0.9884    0.9884       173
```

Висновки

1. Проаналізовано сучасні методи кіберзахисту систем керування базами даних.
Визначено підходи до виявлення SQL-ін'єкцій, способи моніторингу активності та інструменти машинного навчання для підвищення стійкості систем.

2. Спроєктовано архітектуру веб-застосунку.
Сформовано модульну структуру системи з окремими компонентами: перевірка SQL-запитів, симуляція атак, аналітика, робота з ML-моделлю та зовнішніми БД.

3. Реалізовано модуль перевірки SQL-запитів.
Розроблено механізм подвійної перевірки — на основі правил SQL-парсера та ML-моделі, що забезпечує виявлення небезпечних запитів.

4. Створено модуль симуляції атак і моніторингу в реальному часі.
Забезпечено відтворення потоків запитів, збір статистики та побудову графіків Allowed/Blocked у реальному часі.

5. Проведено оцінку ефективності системи.
Отримано високі показники якості моделі (ROC AUC, precision, recall), перевірено стабільність роботи та підтверджено функціональну стійкість розробленого застосунку.