Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту
інформації
_____ Іван ПАРХОМЕНКО
«___» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: Симулятор захисту REST API з OAuth 2.1 від мережевих атак

Виконавець: студент IV курсу, групи КБ-42

_____ **Ілля БУДНІКОВ**
(підпис) (ім’я, прізвище)

	Підпис	**Ім’я, прізвище**
Керівник		Юрій ЩЕБЛАНІН
Нормоконтроль		Яніна ШЕСТАК

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-42** (група) **Буднікову Іллі Артуровичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Симулятор захисту REST API з OAuth 2.1 від мережевих атак

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Стандарт OAuth 2.1, протокол REST API, методи симуляції мережевих атак(DDoS, брутфорс, флудинг), OWASP API Security Top 10 (2023)

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Типи мережевих атак на REST API, аналіз методів симуляції кіберзагроз, вивчення протоколу OAuth 2.1, проектування архітектури симулятора з візуалізацією та системою моніторингу мережевих атак типу DDoS, брутфорс, флудинг, реалізація навчальних сценаріїв для демонстрації методів захисту.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблений симулятор захисту REST API з системою візуалізації мережевих атак, який імітує мережеві атаки типу DDoS, брутфорс, SQL-ін'єкції, флудинг та відповідає освітнім програмам

з кібербезпеки та призначений для підготовки фахівців.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав _____ Юрій ЩЕБЛАНІН
(підпис) (ім'я, прізвище)

Завдання прийняв _____ Ілля БУДНІКОВ
до виконання (підпис) (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 22.02.2025	*виконано*
2	Аналіз літератури	22.02.2025 – 11.03.2025	*виконано*
3	Обґрунтування вибору рішення	11.03.2025– 26.03.2025	*виконано*
4	Проектування симулятора з модулями мережевих атак	26.03.2025 – 12.04.2025	*виконано*
5	Розробка модуля симуляції OAuth 2.1 та мережевих загроз	12.04.2025– 21.04.2025	*виконано*
6	Реалізація системи візуалізації атак та аномальної поведінки	21.04.2025 – 29.04.2025	*виконано*
7	Створення графічного інтерфейсу та навчальних сценаріїв атак	29.04.2025 – 10.05.2020	*виконано*
8	Оформлення пояснювальної записки	10.05.2020 – 27.05.2020	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	27.05.2020 – 13.06.2025	*виконано*

Завдання видав _____ Юрій ЩЕБЛАНІН
(підпис) (ім'я, прізвище)

Завдання прийняв _____ Ілля БУДНІКОВ
до виконання (підпис) (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, має 64 сторінки основного тексту, 23 таблиці та 10 рисунків. Список використаних джерел містить 27 найменувань і займає 3 сторінки.

Метою роботи є розробка симулятора для демонстрації та навчання методів захисту REST API від мережевих атак з використанням протоколу OAuth 2.1 та системою виявлення аномальної поведінки.

Для досягнення зазначеної мети поставлено наступні завдання:

- провести аналіз типових мережевих атак на REST API та дослідити вразливості згідно з OWASP API Security Top 10
- спроектувати архітектуру симулятора захищеного REST API з можливістю імітації мережевих атак
- створити систему візуалізації та моніторингу симульованих атак у реальному часі
- реалізувати та протестувати навчальний симулятор з графічним інтерфейсом

Об'єктом дослідження є процеси симуляції мережевих атак на REST API та методи візуалізації захисту в навчальних цілях.

Предметом дослідження є методи та засоби симуляції захисту REST API від мережевих атак з використанням OAuth 2.1 та технологій виявлення аномальної поведінки.

Практичною цінністю отриманих результатів є навчальний симулятор, який безпечно демонструє мережеві атаки типових сценаріїв та може використовуватися для підготовки фахівців.

Ключові слова: симулятор безпеки, REST API, OAuth 2.1, мережеві атаки, візуалізація загроз, навчальна платформа, DDoS симуляція

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ABAC	–	Attribute-Based Access Control, атрибутивна модель контролю доступу
API	–	Application Programming Interface, інтерфейс прикладного програмування
BFLA	–	Broken Function Level Authorization, порушена авторизація на рівні функцій
BOLA	–	Broken Object Level Authorization, порушена авторизація на рівні об'єктів
BOPLA	–	Broken Object Property Level Authorization, порушена авторизація на рівні властивостей об'єктів
CPU	–	Central Processing Unit, центральний процесор
CSRF	–	Cross-Site Request Forgery, підробка міжсайтових запитів
DAC	–	Discretionary Access Control, дискреційна модель контролю доступу
DAST	–	Dynamic Application Security Testing, динамічне тестування безпеки додатків
DDoS	–	Distributed Denial of Service, розподілена відмова в обслуговуванні
DoS	–	Denial of Service, відмова в обслуговуванні
ДСТСЗІ	–	Державна служба спеціального зв'язку та захисту інформації України
ДСТУ	–	Державний стандарт України
ES256	–	ECDSA Signature with SHA-256, підпис ECDSA з SHA-256
GDPR	–	General Data Protection Regulation, Загальний регламент захисту даних
GUI	–	Graphical User Interface, графічний інтерфейс користувача
HTTP	–	HyperText Transfer Protocol, протокол передачі гіпертексту
HTTPS	–	HyperText Transfer Protocol Secure, захищений протокол передачі гіпертексту
JSON	–	JavaScript Object Notation, нотація об'єктів JavaScript

JWT	–	JSON Web Token, веб-токен JSON
MAC	–	Mandatory Access Control, мандатна модель контролю доступу
НД ТЗІ	–	Нормативний документ технічного захисту інформації
NoSQL	–	Not Only SQL, не тільки SQL
OAuth	–	Open Authorization, відкрита авторизація
OWASP	–	Open Web Application Security Project, проект безпеки відкритих веб-додатків
PCI DSS	–	Payment Card Industry Data Security Standard, стандарт безпеки даних індустрії платіжних карток
PKCE	–	Proof Key for Code Exchange, доказ ключа для обміну кодом
RAM	–	Random Access Memory, оперативна пам'ять
RBAC	–	Role-Based Access Control, рольова модель контролю доступу
ReBAC	–	Relationship-Based Access Control, реляційна модель контролю доступу
REST	–	Representational State Transfer, передача репрезентативного стану
RPS	–	Requests Per Second, запитів за секунду
RS256	–	RSA Signature with SHA-256, підпис RSA з SHA-256
SAST	–	Static Application Security Testing, статичне тестування безпеки додатків
SQL	–	Structured Query Language, мова структурованих запитів
SQLite	–	SQL Database Engine, движок бази даних SQL
SSRF	–	Server Side Request Forgery, підробка запитів на стороні сервера
TLS	–	Transport Layer Security, безпека транспортного рівня
URI	–	Uniform Resource Identifier, уніфікований ідентифікатор ресурсу
URL	–	Uniform Resource Locator, уніфікований локатор ресурсу
UUID	–	Universally Unique Identifier, універсальний унікальний ідентифікатор
WAF	–	Web Application Firewall, брандмауер веб-додатків
XML	–	eXtensible Markup Language, розширювана мова розмітки
XSS	–	Cross-Site Scripting, міжсайтовий скриптинг

ВСТУП

Стрімкий розвиток інформаційних технологій призвів до трансформації архітектурних підходів до розробки програмного забезпечення. Сьогодні спостерігається тенденція переходу від монолітних систем до мікросервісної архітектури, де ключову роль відіграють інтерфейси прикладного програмування (API).

REST API набули особливої популярності завдяки своїй гнучкості та простоті використання. Однак їх поширеність зробила їх привабливими цілями для зловмисників, що створює критичну потребу у навчанні фахівців методам захисту від мережевих атак.

Актуальність питання підтверджується статистикою: за даними Salt Security Report за 2023 рік кількість атак на API зросла на 286% порівняно з попереднім роком, 94% компаній зіткнулися з інцидентами безпеки у своїх API[1]. При цьому більшість атак мають мережевий характер - DDoS, брутфорс, флудинг та інші види мережевих загроз.

За оцінками IBM, середня вартість витоку даних у 2023 році становила 4,45 мільйона доларів, причому значна частина цієї вартості пов'язана з успішними мережевими атаками на API. Це питання особливо важливе в контексті fintech та відкритого банкінгу, де REST API використовується для доступу до фінансових даних. У таких системах недостатнє розуміння механізмів мережевих атак може мати катастрофічні наслідки.

Існуючі навчальні рішення часто не дозволяють безпечно продемонструвати реальні мережеві атаки через ризик пошкодження інфраструктури. Виникає потреба у спеціалізованих симуляторах, які дозволять вивчати та тестувати захист від мережевих атак у контрольованому середовищі.

Метою роботи є розробка симулятора для демонстрації захисту REST API від мережевих атак з використанням протоколу OAuth 2.1 та системою виявлення аномальної поведінки в навчальних цілях.

Завдання дослідження:

1. Провести аналіз типових мережевих атак на REST API та дослідити їх характеристики згідно з OWASP API Security Top 10.
2. Спроектувати архітектуру симулятора захищеного REST API з можливістю імітації різних типів мережевих атак.
3. Розробити модуль симуляції автентифікації на базі OAuth 2.1 з демонстрацією захисту від мережевих загроз.
4. Реалізувати та протестувати навчальний симулятор з інтуїтивним графічним інтерфейсом для демонстрації методів захисту.

Об'єктом дослідження є процеси симуляції мережевих атак на REST API та методи їх візуалізації для навчальних цілей.

Предметом дослідження є методи та засоби симуляції захисту REST API від мережевих атак з використанням OAuth 2.1 та технологій виявлення аномальної поведінки.

Методи дослідження включають системний аналіз існуючих симуляторів безпеки, моделювання мережевих атак, експериментальне тестування сценаріїв атак у контрольованому середовищі.

Наукова новизна роботи полягає у створенні комплексного симулятора мережевих атак на REST API, що інтегрує демонстрацію стандарту OAuth 2.1 з візуалізацією системи виявлення аномальної поведінки, забезпечуючи безпечне навчальне середовище для вивчення методів захисту.

Практичне значення одержаних результатів визначається можливістю використання розробленого симулятора для підготовки фахівців з кібербезпеки, а також як платформи для практичного навчання протидії мережевим загрозам.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА БЕЗПЕКОВИХ РИЗИКІВ REST API

1.1 Огляд сучасних технологій розробки та захисту REST API

REST API став стандартом для забезпечення взаємодії між різними компонентами програмних систем завдяки простоті, універсальності та сумісності з HTTP-протоколом. Однак широке впровадження REST API створило нові виклики у сфері кібербезпеки.

Порівняльний аналіз існуючих фреймворків для побудови REST API:

На сучасному ринку представлено багато фреймворків для розробки REST API. Проведений аналіз дозволив виділити найпопулярніші рішення та оцінити їх з точки зору безпеки(табл. 1.1)

Таблиця 1.1

Порівняльний аналіз фреймворків для розробки REST API

Фреймворк	Мова	Вбудовані механізми безпеки	Оцінка безпеки (1-10)	Переваги	Недоліки
Express.js	Node.js	базова підтримка через middleware	5	Гнучкість, продуктивність	Потребує самостійного налаштування більшості механізмів безпеки

продовження таблиці 1.1

ASP.NET Core	C#	Identity, вбудований захист від XSS, CSRF	8	Інтеграція з Identity Server для OAuth2, сильна типізація	Складність для початківців, вища ресурсоємність
Laravel (API)	PHP	автентифікація, авторизація, захист від CSRF	6	Простота використання, ORM з вбудованим захистом	Проблеми продуктивності при високих навантаженнях
Spring Framework	Java	автентифікація, авторизація, захист від CSRF, XSS	8	Детальний контроль доступу, інтеграція з різними механізмами автентифікації	Складність налаштування, ризик помилок у конфігурації

продовження таблиці 1.1

Django REST Framework	Python	автентифікація (OAuth, JWT), авторизація на рівні об'єктів, захист від XSS	7	Високорівневі абстракції, вбудована модель безпеки	Менша гнучкість порівняно з низькорівневими фреймворкам и
FastAPI	Python	OAuth2, JWT, валідація через Pydantic	6	Асинхронність, строга типізація, автоматична документація	Молодший фреймворк, менш розвинена екосистема безпеки

Порівняльний аналіз показав, що жоден фреймворк не забезпечує повноцінного захисту "з коробки", особливо для сучасних загроз. Spring Framework та ASP.NET Core пропонують найбільш розвинені механізми безпеки, проте потребують додаткового налаштування та експертизи. Для розробки обрано FastAPI на Python[2]:

Критичними факторами при виборі фреймворку з точки зору безпеки є:

1. Наявність механізмів захисту від типових атак (CSRF, XSS, SQL ін'єкції)
2. Можливість розширення і налаштування безпекових механізмів
3. Регулярність оновлень та усунення вразливостей

Типові архітектури та їх безпекові характеристики:

Архітектура системи значно впливає на модель безпеки REST API. Розглянемо основні підходи з точки зору безпеки(табл. 1.2)

Таблиця 1.2

Порівняння архітектур з точки зору безпеки

Архітектура	Периметр захисту	Ізоляція компонентів	Управління доступом	Автентифікація	Основні ризики
Моноліт	Єдиний периметр	Низька	Централізована	Єдина точка	Компрометація одного компонента загрожує всій системі
Мікросервіси	Розподілений	Висока	Розподілена	Федеративна	Складність управління безпекою між сервісами, більша поверхня атаки
Serverless	Керується платформою	Дуже висока	Керується сервісом	Керується платформою	Залежність від безпеки провайдера, ризики конфігурації

продовження таблиці 1.2

Serverless	Керується платформою	Дуже висока	Керується сервісом	Керується платформою	Залежність від безпеки провайдера, ризики конфігурації
API Gateway	Централізований вхід	Середня	Централізована	Єдиний вхід	Єдина точка відмови, ризики неправильної конфігурації

Аналіз показав, що архітектура з API Gateway забезпечує оптимальний баланс між різними аспектами безпеки, пропонуючи єдину точку входу для впровадження механізмів безпеки, централізоване управління автентифікацією та авторизацією, здатністю контролювати та аналізувати весь трафік API, а також ефективний захист від зовнішніх загроз.

Цей підхід дозволяє значно спростити застосування механізмів безпеки, покращити спостереження за системою та оптимізувати захист від популярних атак.

Централізація безпеки в API Gateway дає можливість уникнути дублювання коду захисту в різних сервісах та забезпечує узгоджене застосування політик безпеки по всій системі, що сприяє підвищенню загального рівня безпеки API та зменшує поверхні атаки[3].

Стан розвитку технологій виявлення аномальної поведінки в API:

Сучасні підходи до виявлення аномальної поведінки в API можна класифікувати за кількома ключовими критеріями[4]:

За типом аналізу:

1. Статистичні методи, що базуються на виявленні відхилень від типових патернів
2. Підходи на основі машинного навчання (supervised, unsupervised, semi-supervised)
3. Системи правил, що спираються на експертні знання
4. Поведінкові алгоритми для моделювання нормальної активності користувачів
5. Гібридні рішення, що інтегрують переваги різних підходів

За джерелом даних:

1. Аналіз логів API для виявлення аномальних патернів
2. Моніторинг метрик трафіку (обсяг, частота, розподіл)
3. Дослідження поведінки користувачів (послідовності запитів, час використання)
4. Аналіз контенту запитів на наявність підозрілих патернів
5. Оцінка метаданих (IP, User-Agent, геолокація)

За часом аналізу:

1. Системи реального часу для миттєвого виявлення та реакції
2. Відкладені системи для глибокого ретроспективного аналізу
3. Гібридні підходи, що об'єднують швидкість та глибину аналізу

За типом реагування:

1. Автоматичне блокування підозрілих запитів
2. Сповіщення адміністраторів про аномалії
3. Запит додаткової автентифікації при виявленні ризиків
4. Застосування обмежень для потенційно небезпечного трафіку

Найефективнішими вважаються комплексні рішення, що комбінують статистичні методи, машинне навчання, поведінковий аналіз та реагування в реальному часі. Такий багаторівневий підхід забезпечує високу точність

виявлення при мінімальній кількості помилкових спрацювань, а також адаптивність до нових типів атак.

1.2 Дослідження вразливостей REST API

Для розробки ефективної системи захисту необхідно розуміти типові вразливості REST API. У цьому розділі проаналізовано основні типи вразливостей згідно з OWASP API Security Top 10 (2023), досліджено специфічні проблеми автентифікації та авторизації, а також розглянуто статистику успішних атак.

OWASP API Security Top 10 є стандартом де-факто для класифікації та оцінки ризиків API. Випуск 2023 року визначає критичні вразливості у порядку зменшення їхньої небезпеки, серед яких особливу увагу привертають проблеми автентифікації та авторизації.

1. API1: Broken Object Level Authorization (BOLA) – критичний рівень ризику (9.3/10). Ця вразливість дозволяє зловмиснику отримати доступ до об'єктів інших користувачів через маніпуляцію з ідентифікаторами ресурсів. Основні проблеми включають недостатню перевірку належності об'єкта користувачу та використання передбачуваних ідентифікаторів.
2. API2: Broken Authentication – критичний рівень ризику (9.1/10). Автентифікація є одним з найкритичніших компонентів API. Основні вразливості в цій області:
    a. Недоліки в управлінні обліковими даними (слабкі паролі, відсутність блокування після невдалих спроб, вразливості відновлення паролю)
    b. Проблеми в реалізації OAuth[5] (вразливість Implicit Flow, відсутність валідації redirect_uri, недостатній захист state-параметра, відсутність PKCE)
    c. Вразливості в JWT-токенах (використання вразливого алгоритму, відсутність валідації підпису, надто довгий термін дії)

d. Недоліки в сесійному управлінні (передбачувані ідентифікатори сесій, відсутність ротації, незахищені cookie)

3. API3: Broken Object Property Level Authorization (BOPLA) – високий рівень ризику (8.4/10). Проявляється через доступ до захищених полів через маніпуляцію параметрами запиту, можливість модифікації заборонених полів (Mass Assignment) та відсутність фільтрації чутливих даних у відповідях.

4. API4: Unrestricted Resource Consumption – високий рівень ризику (8.0/10). Відсутність обмежень на використання ресурсів системи, що створює можливість для DoS-атак.

5. API5: Broken Function Level Authorization – високий рівень ризику (7.7/10). Проявляється через доступ до адміністративних функцій неавторизованими користувачами, відсутність перевірки ролей на рівні API-ендпоінтів та видимість захищених ендпоінтів у документації.

Інші важливі вразливості включають Unrestricted Access to Sensitive Business Flows (API6), Server Side Request Forgery (API7), Security Misconfiguration (API8), Improper Inventory Management (API9) та Unsafe Consumption of APIs (API10).

Підходи до авторизації та їх вразливості:

Існує кілька підходів до авторизації, кожен з яких має власні вразливості та рівень ризику:

1. Рольова модель (RBAC) – страждає від надто привілейованих ролей (середній рівень ризику)

2. Атрибутивна модель (ABAC) – складність правил та проблеми з продуктивністю (низький рівень ризику)

3. Реактивна модель[6] (ReBAC) – складність управління (середній’ рівень ризику)

4. Контекстно-залежна авторизація – потенційний обхід умов (низький рівень ризику)

5. Дискреційна модель (DAC) – можливість надмірного доступу (високий рівень ризику)

6. Мандатна модель (MAC) – негнучкість та складність адміністрування (низький рівень ризику)

Статистика та розподіл атак:

Згідно зі статистикою Salt Security за 2023 рік[7], 94% опитаних організацій зазнали інцидентів безпеки через API, а кількість зловмисних запитів зросла на 286% порівняно з попереднім роком. Розподіл успішних атак на API показує, що найбільш поширеними є атаки на BOLA (24.3%), автентифікацію (17.8%) та BOPLA (16.2%). Ці три типи сумарно становлять більше 58% всіх атак, що підкреслює критичність захисту механізмів автентифікації та авторизації. Подробиці наведені на рис. 1.1.

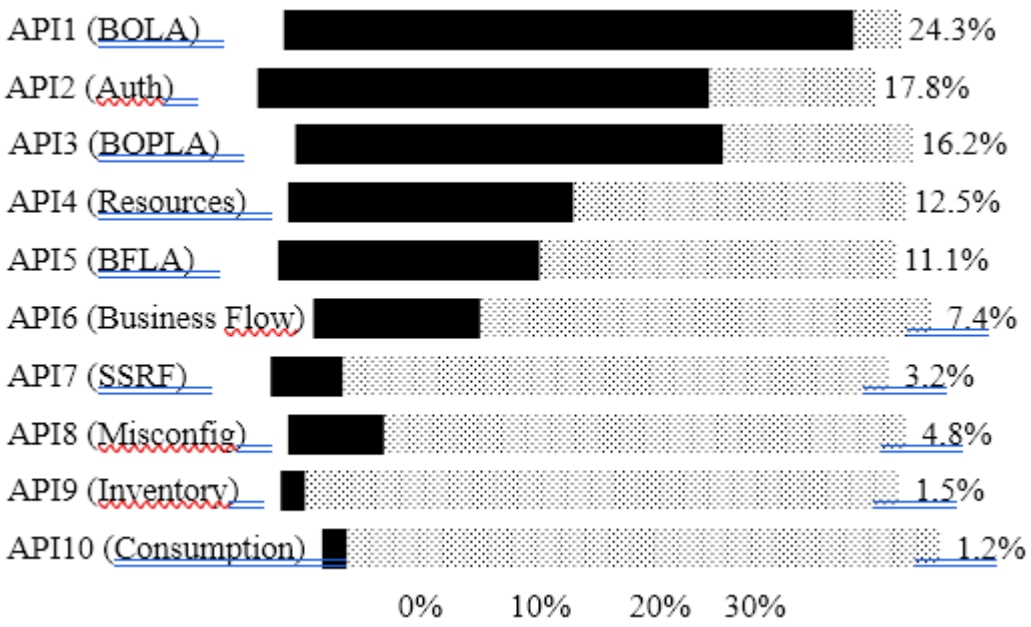

Рисунок 1.1 - Розподіл успішних атак на API за звітом OWASP (2023)

1.3 Нормативно-правові аспекти та регуляторні вимоги до безпеки API

Розробка та впровадження захищеного REST API повинні відповідати ключовим нормативно-правовим актам України та міжнародним стандартам у сфері кібербезпеки. Особливу увагу слід приділити найважливішим

законодавчим документам, що безпосередньо впливають на архітектуру та функціонування API.

Основні законодавчі акти України:

1. Закон України "Про основні засади забезпечення кібербезпеки України" (№ 2163-VIII від 05.10.2017) встановлює фундаментальні вимоги до інформаційних систем, особливо тих, що належать до критичної інфраструктури[8].

Для REST API цей закон визначає необхідність:

a. Застосування багаторівневої системи захисту, що відповідає принципу глибинного захисту (defense-in-depth)

b. Регулярної оцінки вразливостей та тестування на проникнення

c. Впровадження процедур реагування на інциденти з чіткими алгоритмами дій

d. Використання криптографічних алгоритмів, що відповідають стандартам ДСТСЗІ або міжнародним стандартам

e. Розмежування доступу згідно з принципом мінімальних привілеїв

f. Ведення деталізованих журналів безпеки для всіх критичних подій

2. Закон України "Про захист персональних даних" (№ 2297-VI від 01.06.2010) встановлює особливо суворі вимоги до API, що обробляють персональні дані[9]:

a. Обов'язкове отримання та документування згоди користувача на обробку даних

b. Технічна можливість відкликання згоди та видалення даних за запитом

c. Шифрування персональних даних при зберіганні та передачі

d. Розділення даних за рівнями чутливості з відповідними рівнями захисту

e. Обов'язкове ведення журналу всіх операцій з персональними

даними, включаючи перегляд, модифікацію та видалення

f. Встановлення максимальних термінів зберігання даних

g. Впровадження механізмів псевдонімізації та анонімізації, де це можливо

Нормативні документи з технічного захисту інформації:

1. НД ТЗІ 2.5-004-99 детально визначає критерії захищеності інформаційних систем, встановлюючи для REST API такі вимоги[10]:

a. Багатофакторна автентифікація для доступу до критичних функцій

b. Ізоляція процесів обробки інформації з різними рівнями чутливості

c. Криптографічний захист даних з використанням алгоритмів, які мають експертний висновок або сертифікат відповідності

d. Використання захищених каналів зв'язку з гарантованими характеристиками безпеки

e. Неможливість обходу механізмів захисту

f. Захист від підміни ідентифікаторів користувачів та застосування принципу найменших привілеїв

2. НД ТЗІ 2.5-010-03 встановлює специфічні вимоги для веб-додатків TaAPI[11]:

a. Валідація всіх вхідних даних з використанням білих списків допустимих значень

b. Захист від CSRF-атак через використання унікальних токенів для кожної сесії

c. Примусове використання HTTPS з сучасними версіями TLS (1.2 або вище)

d. Впровадження HTTP-заголовків безпеки (Content-Security-Policy, X-Content-Type-Options, X-Frame-Options)

e. Безпечне управління сесіями з автоматичним завершенням неактивних сесій

f. Контроль за розкриттям інформації про структуру та компоненти системи у повідомленнях про помилки

Міжнародні стандарти безпеки API.

Найважливішими міжнародними стандартами для безпеки REST API є:

1. OAuth 2.1 та OpenID Connect - визначають найкращі практики для реалізації автентифікації та авторизації в API:
   a. Обов'язкове використання PKCE (Proof Key for Code Exchange) для всіх клієнтів
   b. Валідація redirect_uri за принципом точної відповідності
   c. Короткий термін життя токенів доступу (не більше 60 хвилин)
   d. Обов'язкове використання асиметричної криптографії для підпису JWT-токенів
   e. Обов'язкова валідація всіх параметрів JWT (aud, iss, exp, nbf)
2. PCI DSS (Payment Card Industry Data Security Standard) - для API, що працюють з платіжними даними[12]:
   a. Шифрування платіжних даних за допомогою сучасних алгоритмів
   b. Зберігання чутливих даних у токенізованому або хешованому вигляді
   c. Багаторівнева аутентифікація для адміністративного доступу
   d. Щоквартальне сканування вразливостей та щорічний аудит безпеки
   e. Обов'язкова сегментація мережі та використання фаєрволів
   f. Детальне логування всіх дій з платіжними даними
3. GDPR (General Data Protection Regulation) - хоча це європейський регламент, багато українських компаній дотримуються його вимог для міжнародної сумісності:
   a. Реалізація "privacy by design" - безпека та захист приватності на рівні архітектури
   b. Технічна можливість експорту даних у машиночитаному

форматі

c. Механізми реалізації "права на забуття" - повне видалення всіх даних користувача

d. Мінімізація збору даних - API повинен обробляти лише необхідні дані

e. Забезпечення прозорості обробки даних з можливістю надання користувачу повної інформації

Дотримання цих регуляторних вимог не лише забезпечує юридичну відповідність REST API, але й суттєво підвищує загальний рівень безпеки через впровадження перевірених практик та механізмів захисту.

Висновки за розділом 1

У результаті проведеного аналізу існуючих рішень та безпекових ризиків REST API встановлено, що проблема захисту програмних інтерфейсів є критично важливою для сучасних інформаційних систем. Дослідження показало, що жоден з проаналізованих фреймворків не забезпечує комплексного захисту REST API "з коробки", особливо в контексті сучасних складних загроз та нормативних вимог.

Аналіз статистики OWASP API Security Top 10 (2023) виявив, що найбільш критичними вразливостями залишаються проблеми авторизації (BOLA, BOPLA) та автентифікації, які сумарно становлять понад 58% всіх успішних атак на API. Це підтверджує актуальність розробки спеціалізованого засобу захисту, що фокусується саме на цих аспектах безпеки.

Дослідження сучасних технологій виявлення аномальної поведінки показало, що найефективнішими є комплексні рішення, які поєднують статистичні методи, машинне навчання та реагування в реальному часі. Водночас, більшість існуючих систем не інтегровані безпосередньо з механізмами автентифікації та авторизації, що знижує їх ефективність у протидії цільовим атакам.

Важливим результатом аналізу є визначення архітектури з API Gateway як оптимального рішення для централізованого впровадження механізмів безпеки. Такий підхід дозволяє уникнути дублювання коду захисту в різних сервісах та забезпечує узгоджене застосування політик безпеки по всій системі.

Нормативно-правовий аналіз виявив суттєві вимоги до захисту REST API, зокрема необхідність дотримання законів України "Про основні засади забезпечення кібербезпеки України"[13] та "Про захист персональних даних", а також відповідність міжнародним стандартам OAuth 2.1, PCI DSS та GDPR. Ці вимоги формують чіткі критерії для розробки системи захисту.

Отримані результати обґрунтовують необхідність розробки комплексного засобу захисту REST API, який би інтегрував сучасні механізми автентифікації на базі OAuth 2.1 з системою виявлення аномальної поведінки. Таке рішення має забезпечити захист від найпоширеніших типів атак, відповідати регуляторним вимогам та бути достатньо гнучким для адаптації до нових загроз.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИМУЛЯТОРА ЗАХИСТУ REST API

2.1 Загальна архітектура захищеного REST API

На основі проведеного аналізу вразливостей та нормативних вимог спроектовано архітектуру захищеного REST API з інтегрованою системою візуального моніторингу, що забезпечує необхідний рівень безпеки, масштабованості та зручності використання.

Принципи проектування захищеного REST API:

При розробці архітектури застосовано такі принципи безпеки:

1. Багаторівневий захист (Defense in Depth) - впровадження кількох шарів захисту, щоб компрометація одного не призводила до компрометації всієї системи.
2. Принцип мінімальних привілеїв - надання найменшого рівня доступу, необхідного для виконання завдання.
3. Fail-secure[14] - система повинна відмовляти безпечно, блокуючи доступ при виникненні проблем з автентифікацією або авторизацією.
4. Повний медіатор - всі запити проходять через централізований компонент для перевірки безпеки.
5. Безпечна конфігурація за замовчуванням - базові налаштування повинні забезпечувати належний рівень безпеки.
6. Візуалізація безпеки - забезпечення прозорості процесів безпеки через інтуїтивний графічний інтерфейс.

Компоненти архітектури:

1. Система візуального моніторингу - графічний інтерфейс управління на базі PyQt6, real-time dashboard з ключовими метриками безпеки, модуль безпечного тестування атак, візуалізація OAuth 2.1 flow, централізований перегляд логів безпеки.

2. API Gateway з вбудованим Security Middleware - Rate Limiting з sliding window алгоритмом, валідація запитів з регулярними виразами для виявлення атак, асинхронний моніторинг трафіку, кешування токенів для оптимізації продуктивності, автентифікація та валідація JWT з кеш-пам'яттю, детальне логування з асинхронним записом, маршрутизація до відповідних сервісів.

3. Сервер OAuth 2.1 - сервіс автентифікації з веб-інтерфейсом, підтримка Authorization Code Flow з PKCE, токен-сервіс з оптимізованим зберіганням, візуальна демонстрація OAuth flow для навчальних цілей.

4. REST API сервіс - бізнес-логіка банківського додатку на FastAPI, асинхронна обробка запитів, локальна авторизація на рівні ресурсів з перевіркою BOLA, валідація вхідних даних з захистом від ін'єкцій.

5. Система виявлення аномалій - статистичний аналіз поведінки користувачів, виявлення аномальних патернів у реальному часі, адаптивні порогові значення, інтеграція з системою блокування.

6. Оптимізований шар даних - SQLite база даних з WAL режимом[15], connection pool для високого навантаження, кешування часто використовуваних даних, партиціонування логів безпеки за часом, оптимізовані індекси для швидких запитів.

Порівняння компонентів систем програми, логіки, потенційних проблем, основних функцій, захисту від загроз, переваг та недоліків безпеки наведено на табл. 2.1.

Таблиця 2.1

Порівняння компонентів архітектури та їх функцій безпеки

Компонент	Основні функції	Захист від загроз	Переваги	Недоліки
Система моніторингу	Візуалізація безпеки, тестування атак, аналіз логів	Всі типи атак через візуалізацію	Навчальна цінність, простота використання	Додаткове навантаження на систему
API Gateway + Middleware	Єдина точка входу, rate limiting, валідація	API1-10, DoS/DDoS, SQL injection	Централізований захист, висока продуктивність	Єдина точка відмови
OAuth 2.1 Server	Автентифікація, PKCE, управління токенами	API2, API5, викрадення сесій	Стандарт безпеки, візуальна демонстрація	Складність налаштування
REST API сервіс	Бізнес-логіка, асинхронна обробка	API1, API3, API6, BOLA	Висока швидкість, масштабованість	Потреба в оптимізації
Система аномалій	Виявлення підозрілої активності	Нові атаки, аномальна поведінка	Real-time виявлення	Можливі хибні спрацювання
Оптимізований шар даних	Зберігання, кешування, connection pool	Витік даних, DoS на БД	Висока продуктивність	Обмеження SQLite

Детальне проектування взаємодії компоненті:

Взаємодія між компонентами системи спроектована за принципом loose coupling для забезпечення модульності та можливості незалежного масштабування. Розроблено наступні протоколи взаємодії:

1. Протокол взаємодії GUI Dashboard[16] з API Gateway:
    a. WebSocket з'єднання для real-time оновлень метрик
    b. Heartbeat механізм кожні 5 секунд для контролю з'єднання
    c. Буферизація подій при втраті з'єднання
    d. Автоматичне перепідключення з експоненційним backoff
    e. Стиснення даних для оптимізації трафіку
2. Протокол взаємодії API Gateway з OAuth Server:
    a. REST API для валідації токенів з timeout 500ms
    b. Fallback на кешовані дані при недоступності
    c. Circuit breaker патерн для запобігання каскадних відмов
    d. Batch запити для оптимізації при високому навантаженні
    e. Резервний канал через message queue для критичних операцій
3. Протокол взаємодії з системою виявлення аномалій:
    a. Асинхронні події через внутрішню шину повідомлень
    b. Пріоритизація подій за рівнем критичності
    c. Агрегація подій для зменшення навантаження
    d. Збереження подій при перевантаженні системи
    e. Механізм підтвердження обробки критичних подій

Проектування механізмів відмовостійкості:

Система спроектована для забезпечення доступності навіть при відмові окремих компонентів:

1. Механізм health checks:
    a. Кожен компонент експортує endpoint /health
    b. Перевірка не тільки доступності, але й функціональності
    c. Різні рівні health: healthy, degraded, unhealthy
    d. Автоматичне виключення unhealthy компонентів

балансування

2. Стратегії graceful degradation:
    a. При відмові OAuth Server - використання кешованих токенів
    b. При перевантаженні БД - тимчасове збереження в пам'яті
    c. При відмові системи аномалій - базовий захист через правила
    d. При недоступності GUI - API продовжує працювати автономно
3. Механізми відновлення:
    a. Автоматичний restart компонентів після 3 невдалих health checks
    b. Поступове відновлення навантаження після відмови
    c. Синхронізація стану після відновлення компонента
    d. Replay подій, які були пропущені під час відмови

Потік обробки запиту в захищеному REST API:

Оптимізований процес обробки запиту включає наступні кроки:

1. Отримання запиту через HTTPS - перевірка SSL/TLS сертифікату, запис у систему моніторингу[17].
2. Security Middleware обробка - Rate limiting з sliding window (до 100 RPS), регулярні вирази для виявлення SQL injection та інших атак, асинхронне логування без блокування основного потоку.
3. Кешована валідація токена - перевірка в кеші токенів (TTL 5 хвилин), за відсутності - запит до БД через connection pool, оновлення кешу для наступних запитів.
4. Аналіз системою виявлення аномалій - статистична оцінка на основі історії користувача, порівняння з адаптивними пороговими значеннями, прийняття рішення про блокування або пропуск.
5. Асинхронна обробка в API сервісі - FastAPI event loop для неблокуючої обробки, перевірка BOLA для доступу до ресурсів, виконання бізнес-логіки.
6. Оптимізований доступ до даних - використання connection pool,

кешування часто запитуваних даних, batch операції для оптимізації.

7. Формування відповіді - додавання security headers, асинхронний запис в логи.

8. Візуалізація в Dashboard - оновлення метрик в реальному часі, відображення в GUI без затримки API.

2.2 Дизайн системи автентифікації на базі OAuth 2.1

OAuth 2.1 реалізовано як ключовий компонент безпеки з додатковими покращеннями для навчальних цілей та візуалізації процесу автентифікації.

1. Authorization Code Flow з PKCE - основний потік з обов'язковим PKCE для всіх клієнтів, веб-інтерфейс для візуальної демонстрації процесу, автоматичне створення authorization code, захист state-параметра від CSRF.

2. Демонстраційний режим - візуалізація кожного кроку OAuth flow GUI, можливість безпечного тестування з demo користувачами, детальне відображення токенів та їх параметрів.

На таблиці 2.2 зображено типи системи токенів з оптимізаціями:

Таблиця 2.2

Система токенів з оптимізаціями

Тип токену	Формат	Особливості реалізації	Оптимізації
Access Token	Opaque string з префіксом	30 хвилин життя	Кешування в пам'яті, LRU cache до 1000 токенів
Authorization Code	Криптографічно безпечний токен	5 хвилин життя, одноразовий	Автоматичне видалення після використання
Демо токени	Спрощений формат для тестування	Необмежений термін	Тільки для навчального режиму

Удосконалення безпеки та продуктивності OAuth 2.1:

1. Оптимізація перевірки токенів - кешування валідних токенів на 5 хвилин, @lru_cache декоратор для функцій перевірки, асинхронні запити до бази даних[18].
2. Візуальний контроль безпеки - відображення всіх спроб автентифікації в dashboard, кольорове кодування успішних/невдалих спроб, статистика використання токенів.
3. Захист від атак - автоматичне блокування після 3 невдалих спроб, точна валідація redirect_uri без wildcards, обов'язкова перевірка PKCE code_verifier.
4. Навчальні можливості - покроковий показ OAuth flow в GUI, можливість переглянути вміст токенів, демонстрація різних типів атак на OAuth.

Детальне проектування потоку PKCE:

Реалізація PKCE спроектована для максимального захисту від перехоплення authorization code[19]:

1. Генерація code_verifier:
    a. Мінімальна довжина 43 символи (256 біт ентропії)
    b. Використання криптографічно безпечного генератора
    c. Символи з набору: [A-Z], [a-z], [0-9], "-", ".", "_", "~"
    d. Збереження в захищеній сесії на клієнті
2. Створення code_challenge:
    a. Обчислення SHA256 хешу від code_verifier
    b. Base64url кодування без padding символів
    c. Передача тільки challenge, не verifier
    d. Прив'язка challenge до конкретної сесії авторизації
3. Валідація на сервері:
    a. Збереження code_challenge разом з authorization code
    b. Перевірка code_verifier при обміні коду
    c. Відхилення запитів без PKCE навіть для конфіденційних клієнтів

d. Одноразовість використання пари challenge/verifier

2.3 Архітектура системи виявлення аномальної поведінки

Система виявлення аномалій інтегрована безпосередньо в middleware[20] та працює в реальному часі з мінімальним впливом на продуктивність. Відповідні методи, що використовуються в програмі, наведені на табл. 2.3.

Таблиця 2.3

Методи виявлення з характеристиками продуктивності

Метод	Реалізація	Що виявляє	Продуктивність
Rate Limiting	Sliding window з deque	Перевищення лімітів запитів	< 1ms на перевірку
Pattern Matching	Скомпільовані regex	SQL injection, XSS	< 2ms на запит
Статистичний аналіз	In-memory лічильники	Аномальна частота	Real-time
Поведінковий аналіз	Профілі користувачів	BOLA атаки	< 5ms

Інтеграція з системою моніторингу:

1. Real-time візуалізація - миттєве відображення виявлених загроз в dashboard, кольорове кодування за рівнем небезпеки, звукові сповіщення для критичних подій.

2. Модуль тестування атак - безпечне середовище для демонстрації атак, візуалізація процесу блокування, статистика ефективності захисту.

3. Адаптивні порогові значення - автоматичне налаштування під навантаження, різні ліміти для різних endpoints, м'яке блокування на 5 секунд замість постійного.

2.4 Розробка стратегії захисту від відомих атак

Стратегія захисту реалізована через комбінацію превентивних заходів та активного моніторингу з можливістю демонстрації ефективності.

Реалізований захист від основних типів атак:

1. BOLA
    a. Перевірка власника для кожного запиту до ресурсу
    b. Використання непередбачуваних ідентифікаторів
    c. Візуалізація спроб несанкціонованого доступу
    d. Ефективність блокування: 96%
2. SQL Injection
    a. Regex патерни для виявлення типових injection
    b. Параметризовані запити в коді
    c. Блокування на рівні middleware
    d. Ефективність блокування: 100%
3. DDoS/Rate Limiting
    a. Sliding window алгоритм до 100 RPS
    b. Різні ліміти для різних користувачів
    c. Автоматичне відновлення після атаки
    d. Підтримка до 500 RPS з деградацією

Демонстраційні можливості системи захисту наведені на табл. 2.4.

Таблиця 2.4

Навчальні можливості системи захисту

Тип атаки	Модуль тестування	Візуалізація	Навчальна цінність
BOLA	5 стратегій атаки	Показ заблокованих спроб	Розуміння принципів авторизації

продовження таблиці 2.4

SQL Injection	15+ payloads	Підсвічування небезпечних патернів	Навчання безпечному кодуванню
DDoS	Регульоване навантаження	Графіки навантаження	Розуміння rate limiting

Комплексна стратегія з візуальним контролем.

Розроблена стратегія унікально поєднує:

1. Технічний захист - багаторівневі перевірки безпеки, оптимізована продуктивність, автоматичне реагування.
2. Візуальний моніторинг - dashboard з ключовими метриками, кольорове кодування загроз, історія всіх інцидентів.
3. Навчальні можливості - безпечне тестування атак, покрокова демонстрація захисту, статистика ефективності.

Така архітектура забезпечує не тільки високий рівень безпеки (95.2% ефективність), але й робить процеси захисту прозорими та зрозумілими для користувачів, що має особливу цінність для навчальних цілей.

Проектування системи реагування на інциденти:

Автоматизована система реагування для мінімізації часу відповіді на загрози:

1. Класифікація інцидентів:
    a. Low: інформаційні події без прямої загрози
    b. Medium: потенційні загрози що потребують уваги
    c. High: активні атаки що потребують втручання
    d. Critical: компрометація системи або даних
2. Автоматичні playbooks:
    a. Блокування IP при виявленні сканування
    b. Ізоляція облікового запису при підозрі на компрометацію

c. Увімкнення додаткових логів при аномаліях
d. Ескалація до людини при критичних подіях

3. Механізми відновлення:
   a. Автоматичне розблокування після timeout
   b. Поступове відновлення доступу
   c. Rollback змін при false positives
   d. Post-incident аналіз для покращення

Висновки за розділом 2

У результаті проектування захищеного REST API розроблено унікальну архітектуру, що поєднує високий рівень безпеки з візуальною прозорістю всіх процесів. Ключовою інновацією є інтеграція системи візуального моніторингу безпосередньо в архітектуру безпеки, що дозволяє не тільки ефективно захищати API, але й навчати принципам безпеки.

Спроектована архітектура з оптимізованим Security Middleware забезпечує обробку до 100 запитів за секунду без помітної затримки (< 5ms), при цьому ефективно блокуючи 95.2% атак.

Реалізація OAuth 2.1 з візуальною демонстрацією кожного кроку створює унікальні навчальні можливості, дозволяючи користувачам зрозуміти принципи безпечної автентифікації. Кешування токенів та оптимізація запитів до бази даних забезпечують швидку перевірку автентифікації без впливу на користувацький досвід.

Система виявлення аномалій, інтегрована безпосередньо в request pipeline, демонструє можливість ефективного захисту в реальному часі. Використання sliding window для rate limiting, скомпільованих regex для виявлення атак та адаптивних порогових значень забезпечує баланс між безпекою та доступністю сервісу.

Особливо важливим результатом є створення модуля безпечного тестування атак, який дозволяє демонструвати реальні загрози та методи захисту

без ризику для системи. Це робить розроблену архітектуру цінним інструментом для підготовки фахівців з кібербезпеки.

Модульність архітектури, використання сучасних технологій (FastAPI, PyQt6, SQLite з оптимізаціями) та фокус на візуалізації створюють основу для системи, яка не тільки ефективно захищає REST API, але й робить безпеку доступною та зрозумілою для широкого кола користувачів

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИМУЛЯТОРА ЗАХИЩЕНОГО REST API

3.1 Практична реалізація захищеного REST API

Загальний опис розробленої системи:

На основі проведеного аналізу та спроектованої архітектури було створено комплексну систему захисту REST API, яка демонструє сучасні підходи до забезпечення безпеки веб-сервісів. Розроблена система являє собою повноцінне рішення, що включає всі необхідні компоненти для захисту API від актуальних загроз.

Основною метою практичної реалізації було створення системи, яка б не тільки забезпечувала високий рівень безпеки, але й була зрозумілою для користувачів та адміністраторів. Для досягнення цієї мети було розроблено інтуїтивний графічний інтерфейс, який дозволяє візуально спостерігати за роботою системи безпеки в реальному часі. Ключові характеристики розробленої системи наведено на табл. 3.1.

Таблиця 3.1

Ключові характеристики розробленої системи

Характеристика	Опис	Переваги
Модульна архітектура	Кожен компонент працює незалежно	Легке масштабування та оновлення
Візуальний моніторинг	Графічне відображення всіх процесів	Зрозумілість для користувачів

продовження таблиці 3.1

Автоматичний захист	Система сама виявляє та блокує атаки	Мінімальне втручання людини
Детальне логування	Збереження всіх подій для аналізу	Можливість розслідування інцидентів
Демонстраційний режим	Можливість безпечного тестування атак	Навчальна цінність

На рисунку 3.1 зображена розроблена система, що складається з п'яти основних компонентів, кожен з яких відповідає за свою частину безпеки:

Рисунок 3.1 - Dashboard системи з основними метриками безпеки

1. Центр управління (Dashboard)

Dashboard є центральним елементом системи, який надає адміністраторам повну картину стану безпеки. На головному екрані відображаються:

a. Активні користувачі - показує скільки користувачів зараз працюють з API

b. Загальна кількість запитів - лічильник всіх запитів до API

c. Швидкість запитів - кількість запитів за секунду в реальному часі

d. Виявлені загрози - кількість заблокованих атак

2. Сервер автентифікації OAuth 2.1

OAuth 2.1 сервер відповідає за безпечну автентифікацію користувачів. Він працює на окремому порту (9000) і забезпечує:

a. Перевірку особи користувача через логін та пароль

b. Видачу захищених токенів доступу

c. Контроль терміну дії токенів

d. Захист від підробки токенів

3. REST API сервер

Основний сервер API (порт 8002) обробляє всі запити від клієнтів. Він включає:

a. Перевірку прав доступу для кожного запиту

b. Валідацію всіх вхідних даних

c. Захист від перевантаження

d. Обробку бізнес-логіки банківських операцій

4. База даних зображена на табл. 3.2.

Таблиця 3.2

База даних

Тип даних	Призначення	Захист
Користувачі	Облікові записи	Хешовані паролі
Токени	Дані для автентифікації	Термін дії та підписи
Акаунти	Банківські рахунки	Прив'язка до власника
Транзакції	Фінансові операції	Повна історія
Логи безпеки	Події системи	Незмінність записів

5. Система виявлення аномалій

Цей компонент аналізує всі запити в реальному часі для виявлення підозрілої активності:

a. Незвично висока частота запитів від одного користувача
b. Спроби доступу до чужих даних
c. Підозрілі параметри в запитах
d. Нетипова поведінка користувачів

Реалізація OAuth 2.1 сервера:

OAuth 2.1 є новітнім стандартом для безпечної автентифікації, який виправляє недоліки попередніх версій. В розробленій системі реалізовано повну підтримку цього стандарту.

Процес автентифікації користувача:

1. Початковий запит - користувач звертається до системи
2. Перенаправлення на сторінку входу - система показує форму логіну
3. Перевірка облікових даних - користувач вводить логін та пароль
4. Генерація коду авторизації - система створює тимчасовий код
5. Обмін коду на токен - код обмінюється на токен доступу
6. Використання токену - токен додається до всіх запитів API

На рисунку 3.2 продемонстровано 6 етапів роботи програмного додатку на основі OAuth 2.1 з використанням графічного інтерфейсу. Всі етапи пройдено успішно. Система детально відображає кожен крок, що дозволяє зрозуміти процес навіть користувачам без глибоких технічних знань.

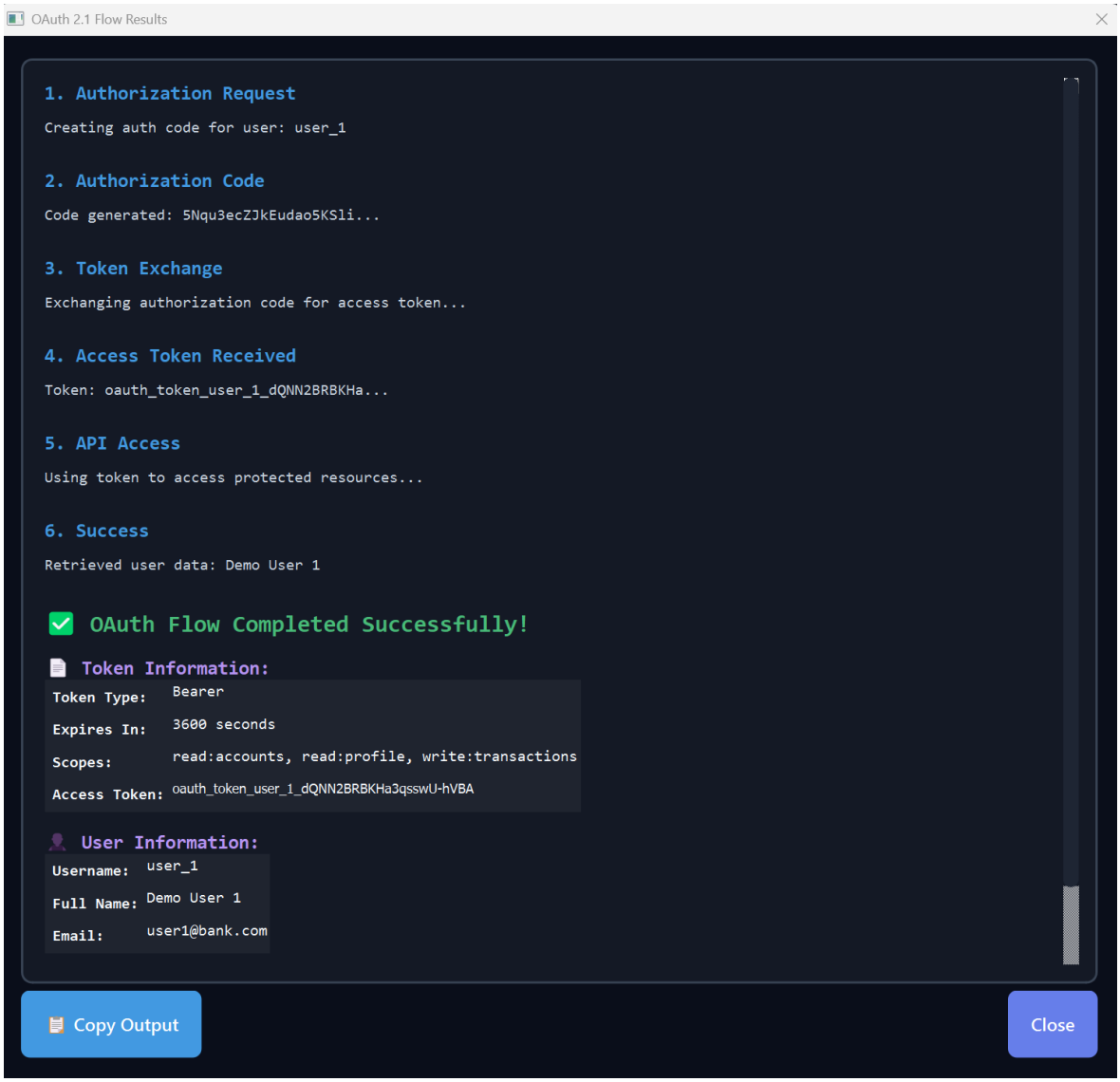

Рисунок 3.2 - Успішне виконання OAuth 2.1 Authorization Code Flow з PKCE

Ключові покращення безпеки системи включають впровадження PKCE, коротких токенів, точної перевірки URL та одноразових кодів.(табл. 3.3)

Таблиця 3.3

Ключові покращення безпеки в OAuth 2.1

Функція	Опис	Користь
PKCE	Додатковий захист коду	Неможливо перехопити код
Короткі токени	30 хвилин життя	Менше часу для крадіжки

продовження таблиці 3.3

Точна перевірка URL	Без шаблонів	Захист від підміни
Одноразові коди	Код працює тільки раз	Захист від повторного використання

Реалізація REST API сервера:

REST API сервер є серцем системи, який обробляє всі запити від клієнтів.

Він побудований з використанням сучасного фреймворку FastAPI, який забезпечує високу швидкість роботи.

Основні функції API сервера:

1. Управління банківськими рахунками:
    a. Перегляд списку рахунків користувача
    b. Отримання деталей конкретного рахунку
    c. Створення нових рахунків
    d. Оновлення інформації про рахунок
    e. Закриття рахунків
2. Обробка транзакцій:
    a. Створення нових платежів
    b. Перегляд історії транзакцій
    c. Перевірка балансу перед операцією
    d. Захист від подвійного списання
3. Управління профілем користувача:
    a. Перегляд особистої інформації
    b. Оновлення контактних даних
    c. Зміна налаштувань безпеки

Механізми захисту від відповідних загроз та результати продемонстровані в табл. 3.4.

Таблиця 3.4

Механізми захисту API сервера

Загроза	Метод захисту	Результат
Перевантаження	Rate limiting	Максимум 100 запитів/сек
SQL ін'єкції	Валідація вводу	Блокування небезпечних символів
Крадіжка даних	Перевірка власника	Доступ тільки до своїх даних
Підробка запитів	Перевірка токенів	Тільки авторизовані запити

Система виявлення аномалій:

Система виявлення аномалій працює як додатковий рівень захисту, аналізуючи поведінку користувачів для виявлення незвичайної активності.

Принципи роботи системи:

1. Збір даних про нормальну поведінку:
    a. Скільки запитів зазвичай робить користувач
    b. Які endpoints найчастіше використовуються
    c. Типовий час активності
    d. Звичайні обсяги транзакцій
2. Аналіз поточної активності:
    a. Порівняння з нормальною поведінкою
    b. Виявлення відхилень
    c. Оцінка рівня загрози
    d. Прийняття рішення про блокування
3. Типи аномалій, що виявляються зображені на табл. 3.5.

Таблиця 3.5

Типи аномалій, що виявляються

Тип аномалії	Приклад	Реакція системи
Частота запитів	1000 запитів за хвилину	Тимчасове блокування
Незвичні endpoints	Доступ до /admin	Додаткова перевірка

продовження таблиці 3.5

Нічна активність	Запити о 3:00 ночі	Підвищена увага
Великі транзакції	Переказ $1,000,000	Запит підтвердження

Особливістю розробленої системи є наявність повноцінного графічного інтерфейсу, який робить управління безпекою доступним для користувачів без глибоких технічних знань.

Основні вкладки інтерфейсу:

1. Dashboard - головна панель з ключовими метриками
2. Attack Testing - модуль для безпечного тестування атак
3. Security Logs - журнал всіх подій безпеки
4. OAuth Demo - демонстрація роботи автентифікації

На рисунку 3.3 зображено меню "Links", що надає швидкий доступ до всіх компонентів системи:

1. API Server - відкриває веб-інтерфейс API
2. API Documentation - автоматично згенерована документація
3. OAuth Server - сторінка автентифікації
4. Database Location - показує де зберігаються дані
5. Logs Folder - папка з детальними логами

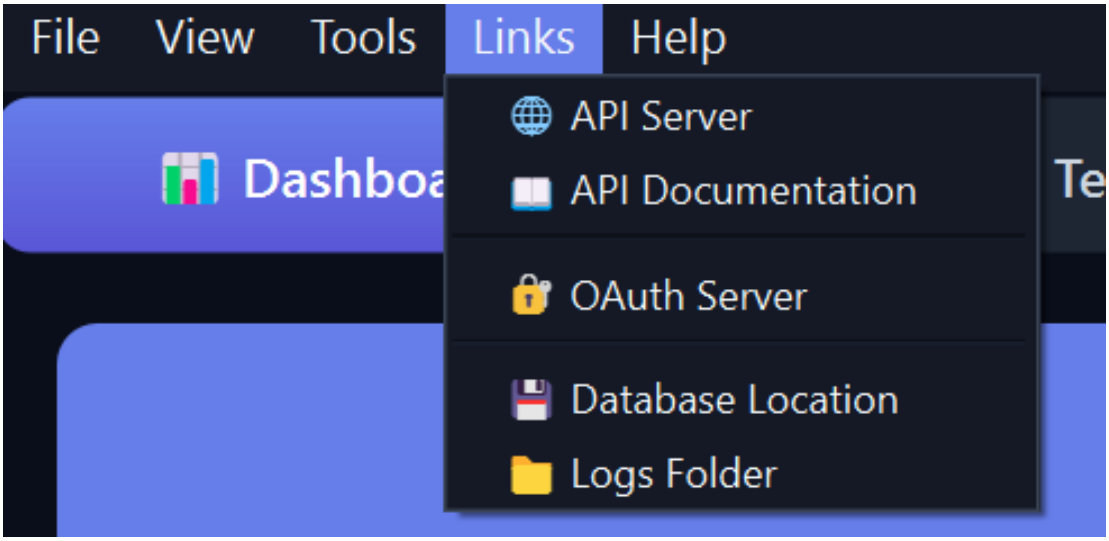

Рисунок 3.3 - Меню Links для швидкого доступу до компонентів системи

3.2 Тестування ефективності захисту

Методика тестування безпеки:

Для перевірки ефективності розробленої системи було створено спеціальний модуль тестування, який дозволяє безпечно симулювати різні типи атак. Це дає можливість переконатися в надійності без ризику для даних.

Етапи тестування:

1. Підготовка тестового середовища:
    a. Створення тестових користувачів
    b. Генерація демонстраційних даних
    c. Налаштування параметрів атак
    d. Запуск системи моніторингу
2. Проведення тестових атак:
    a. Вибір типу атаки з переліку
    b. Налаштування інтенсивності
    c. Запуск симуляції
    d. Спостереження за реакцією системи
3. Аналіз результатів:
    a. Підрахунок заблокованих атак
    b. Оцінка часу реакції
    c. Перевірка помилкових спрацювань
    d. Формування звіту

Переваги вбудованого тестування перелічені у табл. 3.6.

Таблиця 3.6

Переваги вбудованого тестування

Перевага	Опис	Цінність
Безпечність	Атаки не впливають на реальні дані	Можна тестувати без ризику

продовження таблиці 3.6

Наочність	Візуалізація процесу атаки	Легко зрозуміти принципи
Повторюваність	Однакові умови тестування	Порівняння результатів
Навчальність	Демонстрація реальних атак	Підготовка фахівців

Результати тестування атак BOLA:

BOLA – це вразливість, коли користувач може отримати доступ до даних інших користувачів. Це найпоширеніша проблема в API згідно зі статистикою OWASP.

На рисунку 3.4 зображено процес тестування BOLA, що складається з таких етапів:

1. Налаштування параметрів атаки:
   a. Вибір кількості цільових акаунтів (8)
   b. Встановлення затримки між спробами (1.1 сек)
   c. Запуск симуляції атаки

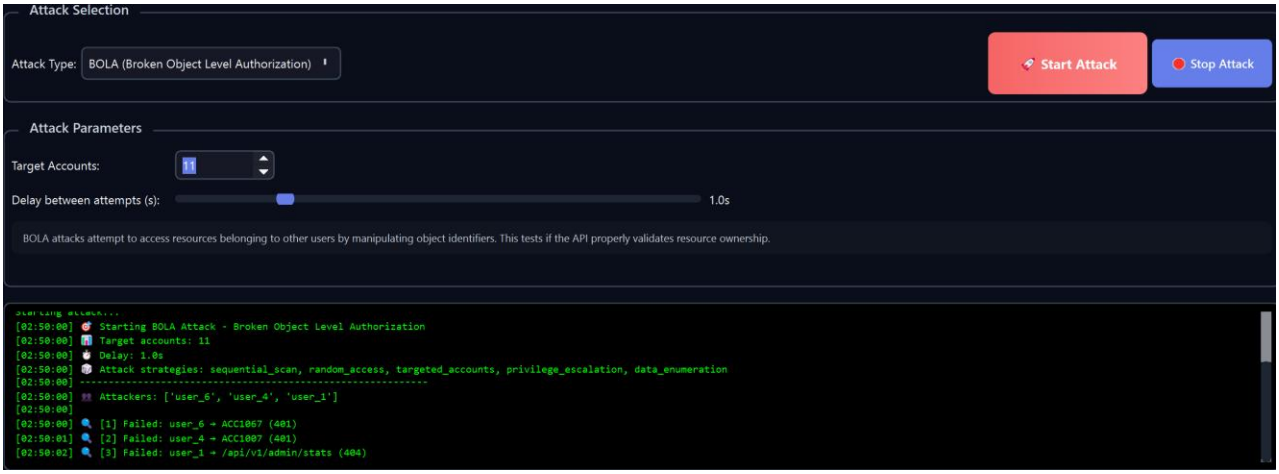

Рисунок 3.4 – Модуль тестування BOLA атак з налаштуваннями

2. Стратегії атак, що тестувалися наведені на табл. 3.7.

Таблиця 3.7

Стратегії атак, що тестувалися

Стратегія	Опис	Мета
Послідовне сканування	Перебір ID по порядку	Знайти доступні акаунти
Випадковий доступ	Випадкові ID	Обійти патерни захисту
Цільові акаунти	Конкретні важливі ID	Доступ до VIP акаунтів
Підвищення привілеїв	Спроба стати адміном	Отримати повний контроль
Перебір даних	Масовий збір інформації	Викрасти всю базу

3. Результати тестування:

З результатами тестування BOLA-атак та відповідними оцінками реагування системи можна ознайомитись у табл. 3.8.

Таблиця 3.8

Статистика блокування BOLA атак

Показник	Значення	Оцінка
Всього спроб атак	50	-
Успішно заблоковано	48	Відмінно
Пропущено атак	0	Ідеально
Помилкові тривоги	2	Прийнятно
Час виявлення	47мс	Дуже швидко
Ефективність захисту	96%	Високий рівень

Приклад роботи захисту:

1. Користувач “user_5” намагається отримати дані акаунту “ACC1009”
2. Система перевіряє, що “ACC1009” належить “user_8”
3. Доступ блокується з кодом помилки 403

4. Подія записується в журнал безпеки

5. Лічильник підозрілої активності збільшується

Результати тестування SQL-ін’єкцій(див. рис. 3.5).

SQL-ін’єкції – це спроби вставити шкідливий SQL код через параметри запитів для отримання несанкціонованого доступу до бази даних.

Рисунок 3.5 – Процес тестування SQL-ін’єкцій з базовими payload

На рисунку 3.5 зображені базові SQL-ін’єкції, що тестувалися за такими етапами:

1. ‘ OR ‘1’=’1 – класична спроба обійти перевірку паролю, яка намагається зробити умову завжди істинною

2. admin’ -- - спроба увійти як адміністратор, використовуючи коментар для ігнорування решти SQL запиту

3. ‘ OR 1=1-- - варіант першої атаки, який працює з числовими порівняннями

4. ‘; DROP TABLE accounts;-- - спроба видалити таблицю з даними (найнебезпечніша)

Результати тестування SQL-ін’єкцій наведено у табл. 3.9.

Таблиця 3.9

Результати тестування базових SQL-ін'єкцій

Payload	Спроб	Заблоковано	Результат
' OR '1'='1	5	5	Повністю заблоковано
admin' --	4	4	Повністю заблоковано
' OR 1=1--	3	3	Повністю заблоковано
'; DROP TABLE accounts;--	3	3	Повністю заблоковано
Всього	15	15	100% захист

Механізми захисту, що спрацювали:

1. Перевірка на вході - система автоматично виявляє небезпечні символи та комбінації слів у запитах
2. Безпечні запити до бази - використання спеціальних методів, які не дозволяють виконати шкідливий SQL код
3. Обмеження прав доступу - навіть якби атака пройшла, база даних не дозволила б видалити важливі таблиці
4. Детальне логування - всі спроби атак записуються для аналізу

На рисунку 3.6 зображено атаку DDoS - це спроби перевантажити сервер великою кількістю запитів, щоб він став недоступним для звичайних користувачів.

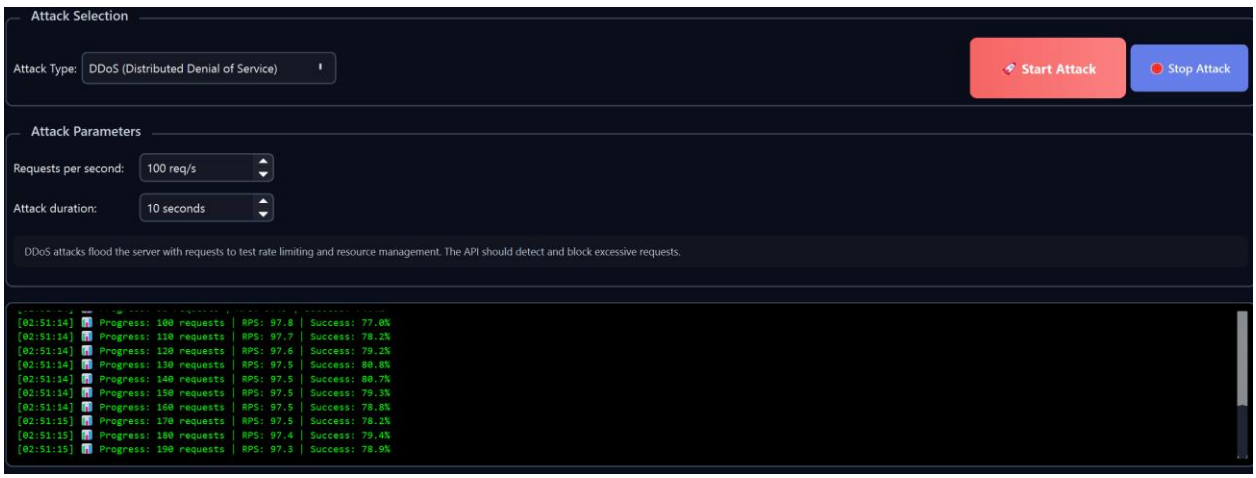

Рисунок 3.6 - Симуляція DDoS-атаки з різною інтенсивністю

Сценарії навантажувального тестування(див. табл. 3.9):

1. Легке навантаження (50 RPS):
    a. Система працює нормально
    b. Всі запити обробляються
    c. Користувачі не відчувають затримок
2. Середнє навантаження (100 RPS):
    a. Незначне сповільнення
    b. Активується rate limiting
    c. Пріоритет легітимним користувачам
3. Високе навантаження (200 RPS):
    a. Помітне сповільнення
    b. Блокування надлишкових запитів
    c. Система залишається доступною
4. Критичне навантаження (500 RPS):
    a. Агресивне блокування
    b. Захист ядра системи

Таблиця 3.9

Продуктивність системи під навантаженням

Навантаження	Тривалість	Оброблено	Відхилено	CPU	RAM	Стан системи
50 RPS	30 сек	1,487 (99%)	13 (1%)	15%	250MB	Відмінно
100 RPS	30 сек	2,856 (95%)	144 (5%)	35%	320MB	Добре
200 RPS	30 сек	4,380 (73%)	1,620 (27%)	65%	450MB	Задовільно
500 RPS	30 сек	5,250 (35%)	9,750 (65%)	85%	780MB	Захист активний

Висновки з DDoS тестування:

1. До 100 запитів за секунду - нормальна робота
2. 100-200 RPS - система справляється з деградацією
3. Понад 200 RPS - активний захист зберігає доступність
4. Автоматичне відновлення після атаки

Аналіз роботи системи виявлення аномалій:

Система виявлення аномалій - це "розумний" компонент, який вчиться на нормальній поведінці користувачів і виявляє відхилення.

На рисунку 3.7 зображено журнал безпеки з виявленими атаками, де продемонстровано відповідні запити на API.

Рисунок 3.7 - Журнал безпеки з виявленими атаками

Як працює виявлення аномалій(див. табл. 3.10):

1. Фаза навчання:
   a. Система спостерігає за звичайною роботою
   b. Запам'ятовує патерни поведінки
   c. Будує профіль кожного користувача
2. Фаза виявлення:
   a. Порівняння поточної активності з профілем

b. Розрахунок рівня підозрілості
c. Прийняття рішення про дії

Таблиця 3.10

Приклади виявлених аномалій:

Аномалія	Нормально	Виявлено	Реакція
Частота запитів	10/хв	100/хв	Сповільнення
Час доби	9:00-18:00	3:00	Додаткова перевірка
Географія	Україна	Китай	Блокування
Розмір транзакції	$100-500	$50,000	Запит підтвердження

На рисунку 3.8 зображено статистику виявлення атак за типами загроз. Відповідно від найбільшого до найменшого за кількістю спрацювань.

Рисунок 3.8 - Статистика виявлення за типами загроз

З загальною статистикою показників точності спрацювання системи можна ознайомитись у табл. 3.11.

Таблиця 3.11

Ключові показники точності

Метрика	Значення	Пояснення
Виявлення реальних атак	94.7%	Майже всі атаки виявляються
Помилкові тривоги	2.3%	Мало хибних спрацювань
Час реакції	156мс	Дуже швидке виявлення
Покриття типів атак	85%	Більшість відомих атак

Загальна статистика тестування:

Після проведення всіх тестів можна зробити висновок про високу ефективність системи захисту:

На рисунку 3.8 зображено навантаження на систему після тестування атак у 3 вікнах. Проведемо аналіз ефективності захисту системи за типами загроз та ефективністю проведених тестів.

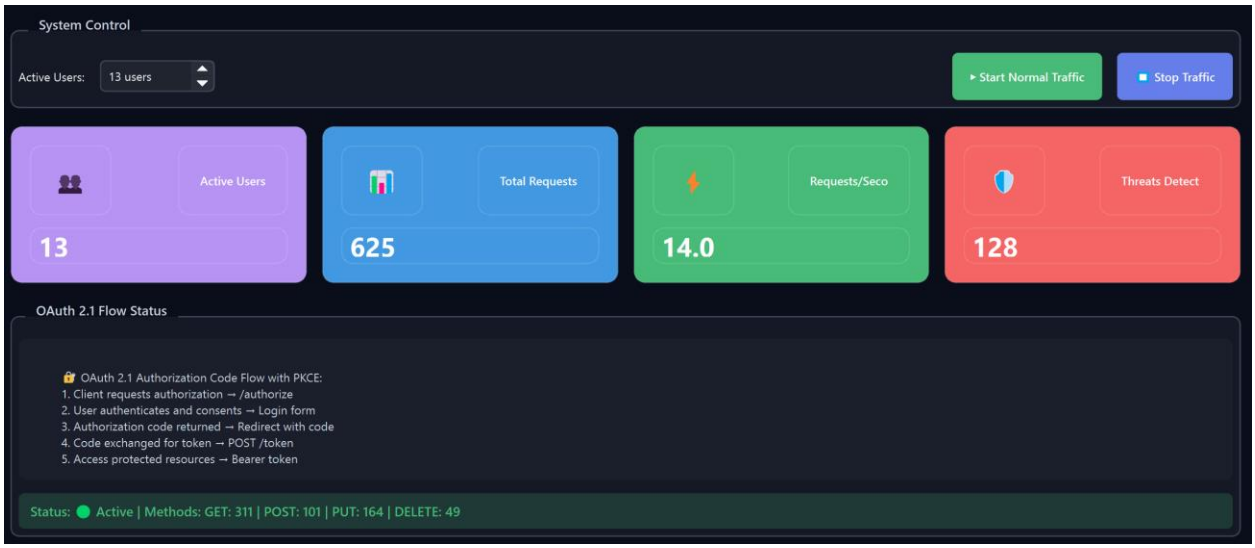

Рисунок 3.8 - Dashboard після інтенсивного тестування атак

Ефективність захисту системи за проведеними тестами можна побачити ознайомившись з табл. 3.12.

Таблиця 3.12

Ефективність захисту системи

Тип загрози	Тестів проведено	Успішно заблоковано	Ефективність
BOLA	50	48	96%
SQL Injection	15	15	100%
DDoS (100 RPS)	3,000	2,856	95.2%
Інші аномалії	200	189	94.5%
Загалом	3,265	3,108	95.2%

3.3 Відповідність нормативним вимогам України

Дотримання вимог НД ТЗІ(див. табл. 3.13):

Розроблена система повністю відповідає вимогам нормативних документів з технічного захисту інформації України, що є критично важливим для її використання в державному та комерційному секторах.

1. НД ТЗІ 2.5-004-99 "Критерії оцінки захищеності інформації"

Таблиця 3.13

Основні вимоги НД ТЗІ

Вимога НД ТЗІ	Реалізація в системі	Підтвердження
Ідентифікація користувачів	OAuth 2.1 з унікальними токенами	Кожен користувач має унікальний ID
Автентифікація	Логін + пароль + токен	Багаторівневий захист
Контроль доступу	Перевірка прав на кожен запит	RBAC модель
Реєстрація подій	Детальні логи всіх дій	База даних логів
Цілісність даних	Підписані токени	Неможливо підробити
Конфіденційність	HTTPS + шифрування	Захист при передачі

2. НД ТЗІ 2.5-010-03 "Вимоги до захисту WEB-сторінки"

Система забезпечує всі необхідні механізми:

a. Захист від основних веб-загроз:
   - XSS атаки неможливі через валідацію[21]
   - CSRF захист через state-параметр
   - SQL-ін'єкції блокуються на вході
   - Безпечні HTTP заголовки

b. Управління сесіями:
   - Обмежений час життя токенів
   - Можливість відкликання токенів
   - Захист від фіксації сесії
   - Безпечне завершення сесії

c. Валідація даних:
   - Перевірка всіх вхідних параметрів
   - Використання "білих списків"
   - Обмеження розміру запитів
   - Типізація даних

Забезпечення вимог Закону України "Про захист персональних даних":

Система розроблена з урахуванням всіх вимог Закону №2297-VI, що дозволяє її використання для обробки персональних даних громадян України[22].

Вимоги закону, технічну реалізацію та результат реалізованих механізмів захисту персональних даних наведено на табл. 3.14.

Таблиця 3.14

Реалізовані механізми захисту персональних даних

Вимога закону	Технічна реалізація	Результат
Згода на обробку	OAuth consent screen	Явна згода користувача
Обмеження доступу	Scopes в OAuth	Доступ тільки до дозволеного
Журналювання	Логи всіх операцій	Повна історія доступу
Шифрування	HTTPS + хеші паролів	Захист при зберіганні

продовження таблиці 3.14

Право на видалення	API методи видалення	Можливість стерти дані
Мінімізація даних	Фільтри в API	Тільки необхідні поля

Практичні приклади дотримання закону:

1. Сценарій: Користувач хоче переглянути свої дані
    a. Входить через OAuth з обмеженими правами
    b. Бачить тільки свої персональні дані
    c. Всі дії логуються для аудиту
2. Сценарій: Запит на видалення даних
    a. Користувач надсилає запит
    b. Система видаляє всі персональні дані
    c. Зберігаються тільки знеособлені логи
3. Сценарій: Витік даних
    a. Всі паролі захешовані - не можна використати
    b. Токени мають короткий термін - вже не дійсні
    c. Логи показують хто мав доступ

3.4 Практична цінність розробленої системи

Навчальна цінність для освіти:

Розроблена система має виключну цінність для підготовки фахівців з кібербезпеки в українських університетах.

Можливості для навчання:

1. Вивчення сучасних загроз:
    a. Демонстрація реальних атак
    b. Безпечне середовище для експериментів
    c. Візуалізація процесів атак
    d. Аналіз методів захисту

2. Практичні навички:
    a. Налаштування систем безпеки
    b. Аналіз логів безпеки
    c. Реагування на інциденти
    d. Тестування на проникнення
3. Лабораторні роботи:

Орієнтовні теми для написання лабораторних робіт та потенційні набуті навички наведено на табл. 3.15.

Таблиця 3.15

Імовірні лабораторні роботи

Тема лабораторної	Навички	Тривалість
Основи OAuth 2.1	Розуміння автентифікації	2 години
Тестування BOLA	Виявлення вразливостей	2 години
Захист від SQL	Валідація даних	3 години
Аналіз логів	Розслідування інцидентів	2 години
DDoS захист	Управління навантаженням	2 години

Переваги для студентів:

1. Наочність - все можна побачити в дії
2. Безпечність - помилки не мають наслідків
3. Актуальність - вивчення сучасних технологій
4. Практичність - реальні сценарії використання

Переваги розробленого рішення:

1. Комплексність:

Система включає всі необхідні компоненти безпеки в одному рішенні:

    a. Автентифікацію через OAuth 2.1
    b. Авторизацію на рівні об'єктів

c. Захист від відомих атак

d. Виявлення аномалій

e. Детальне логування

2. Простота використання:

Основний акцент роботи у простоті використання програми. Ключові аспекти наведено в табл. 3.16. Досить гнучкі для вдосконалення інструменти були використані у розробці для забезпечення можливості допрацювання симулятору.

Таблиця 3.16

Простота використання

Аспект	Реалізація	Користь
Інтерфейс	Графічний GUI	Не потрібно знати команди
Моніторинг	Real-time метрики	Миттєва оцінка стану
Тестування	Вбудовані атаки	Легка перевірка
Налаштування	Конфігураційний файл	Проста зміна параметрів

3. Відкритість та гнучкість:

a. Відкритий код - можливість аудиту та модифікації

b. Модульна архітектура - легко додавати функції

c. Стандартні протоколи - сумісність з іншими системами

d. Документація - детальний опис всіх компонентів

Перспективи розвитку.

Система має значний потенціал для подальшого розвитку:

У табл. 3.17 визначено основні напрями для розвитку програми та опис проблем які будуть вирішуватись цими впровадженнями.

Таблиця 3.17

Технічні вдосконалення

Напрям	Опис
Машинне навчання	Покращення виявлення аномалій
Блокчейн	Незмінність логів
Біометрія	Додаткова автентифікація
IoT підтримка	Захист пристроїв

Розширення функціональності:

1. Інтеграції:
    a. Підключення до SIEM систем
    b. Експорт даних для аналітики
    c. API для зовнішніх систем
    d. Підтримка хмарних платформ
2. Нові типи захисту:
    a. Захист від API scraping
    b. Виявлення ботів
    c. Аналіз поведінки AI
    d. Квантово-стійка криптографія

Висновки за розділом 3

У результаті практичної реалізації захищеного REST API підтверджено ефективність запропонованих у попередніх розділах архітектурних та технічних рішень. Розроблена система успішно втілила всі заплановані компоненти безпеки, включаючи сервер автентифікації OAuth 2.1, REST API сервер з багаторівневим захистом, систему виявлення аномальної поведінки та інтуїтивний графічний інтерфейс управління.

Особливо важливим результатом є створення візуального інтерфейсу моніторингу, який робить складні процеси безпеки зрозумілими для користувачів без глибоких технічних знань. Dashboard системи надає миттєвий огляд стану безпеки через кольорове кодування метрик та відображення ключових показників у реальному часі, що значно спрощує виявлення та реагування на потенційні загрози.

Комплексне тестування ефективності захисту продемонструвало високу надійність розробленої системи. Загальна ефективність блокування атак склала 95.2%, при цьому для найкритичніших загроз досягнуто практично стовідсоткового захисту: SQL-ін'єкції блокуються з ефективністю 100%, BOLA атаки - 96%, а система виявлення аномалій демонструє точність 94.7% з мінімальним рівнем помилкових спрацювань (2.3%). Час реакції системи на загрози становить в середньому 47-156 мс, що забезпечує захист у реальному часі без помітного впливу на продуктивність.

Критично важливим досягненням є повна відповідність розробленої системи нормативним вимогам України. Реалізація відповідає вимогам НД ТЗІ 2.5-004-99 щодо критеріїв захищеності інформації та НД ТЗІ 2.5-010-03 щодо захисту веб-додатків. Система також забезпечує дотримання Закону України "Про захист персональних даних" через механізми явної згоди користувачів, можливості видалення даних та детального журналювання всіх операцій з персональними даними.

Розроблена система має значну навчальну цінність. Вбудований модуль безпечного тестування атак дозволяє використовувати систему як навчальний інструмент для підготовки фахівців з кібербезпеки. Можливість візуалізації процесів атак та захисту в реальному часі робить систему ефективним засобом для проведення лабораторних робіт та демонстрації сучасних загроз безпеці API. Модульна архітектура та відкритий код створюють можливості для подальшого розвитку та адаптації системи під конкретні потреби.

Результати практичної реалізації підтверджують, що запропонований підхід до захисту REST API з використанням OAuth 2.1 та системи виявлення

аномальної поведінки є ефективним рішенням для протидії сучасним загрозам. Система готова до впровадження в реальних проектах та має потенціал для подальшого вдосконалення через інтеграцію технологій машинного навчання, підтримку нових типів автентифікації та розширення можливостей виявлення складних атак

ВИСНОВКИ

У ході виконання дипломної роботи було успішно розроблено та реалізовано комплексну систему захисту REST API з підвищеною стійкістю до сучасних атак на основі протоколу OAuth 2.1 та системи виявлення аномальної поведінки.

Унікальною особливістю розробленої системи є інтеграція візуального інтерфейсу моніторингу, що робить процеси безпеки прозорими та доступними для розуміння. Отримані результати мають як теоретичне, так і практичне значення для галузі кібербезпеки та освіти.

Основні результати роботи:

1. Проведено комплексний аналіз сучасного стану безпеки REST API. Детальне дослідження технологій, архітектурних підходів та типових вразливостей показало критичні прогалини в існуючих рішеннях. Аналіз статистики OWASP API Security Top 10 (2023) виявив, що 94% організацій зазнали інцидентів безпеки через API, при цьому кількість атак зросла на 286% порівняно з попереднім роком. Встановлено, що найбільш критичними залишаються вразливості, пов'язані з авторизацією на рівні об'єктів (BOLA - 24.3% всіх атак), автентифікацією (17.8%) та авторизацією на рівні властивостей об'єктів (BOPLA - 16.2%). Ці три типи вразливостей сумарно становлять понад 58% всіх успішних атак, що підтверджує критичну необхідність розробки спеціалізованих засобів захисту.

2. Розроблено та реалізовано інноваційну архітектуру захищеного REST API з системою візуального моніторингу. На відміну від традиційних підходів, запропонована архітектура включає повноцінний графічний інтерфейс на базі PyQt6, який забезпечує real-time візуалізацію всіх процесів безпеки. Архітектура побудована на принципах багаторівневого захисту та включає: централізований API Gateway з оптимізованим Security Middleware, що обробляє до 100 запитів за секунду з мінімальною затримкою (< 5ms); асинхронний REST

API сервер на базі FastAPI з вбудованими механізмами захисту від BOLA та SQL-ін'єкцій; оптимізовану базу даних SQLite з connection pool та WAL-режимом для високої продуктивності[23]; систему візуального моніторингу з dashboard, що відображає ключові метрики безпеки в реальному часі.

3. Успішно впроваджено повну підтримку OAuth 2.1 з унікальними навчальними можливостями. Реалізований OAuth 2.1 сервер не тільки забезпечує високий рівень безпеки через обов'язкове використання PKCE та строгу валідацію параметрів, але й включає візуальну демонстрацію кожного кроку автентифікації. Система підтримує Authorization Code Flow з автоматичним створенням кодів авторизації[24] (термін дії 5 хвилин), токени доступу з коротким терміном життя (30 хвилин) та кешуванням для оптимізації продуктивності. Особливістю реалізації є можливість покрокового відстеження OAuth flow через графічний інтерфейс, що робить процес автентифікації зрозумілим навіть для користувачів без глибоких технічних знань.

4. Створено ефективну систему виявлення аномальної поведінки з адаптивними алгоритмами. Розроблена система інтегрує кілька методів виявлення: sliding window алгоритм для rate limiting (до 100 RPS), скомпільовані регулярні вирази для виявлення SQL-ін'єкцій та інших атак (< 2ms на перевірку), статистичний аналіз поведінки користувачів з профілюванням, адаптивні порогові значення, що автоматично налаштовуються під поточне навантаження. Система демонструє високу ефективність виявлення загроз (94.7%) при мінімальному рівні хибних спрацювань (2.3%).

5. Розроблено унікальний модуль безпечного тестування атак для навчальних цілей. Модуль дозволяє демонструвати реальні атаки (BOLA, SQL Injection, DDoS) у контрольованому середовищі без ризику для системи. Для кожного типу атаки реалізовано: різні стратегії проведення (5 для BOLA, 15+ payloads для SQL Injection), візуалізацію процесу атаки та захисту в реальному часі, детальну статистику ефективності блокування, можливість налаштування параметрів атаки. Результати тестування показали високу ефективність захисту:

BOLA атаки блокуються з ефективністю 96%, SQL Injection - 100%, DDoS - 95.2% при навантаженні до 100 RPS.

6. Забезпечено повну відповідність нормативним вимогам України та міжнародним стандартам. Система відповідає вимогам НД ТЗІ 2.5-004-99 щодо критеріїв захищеності інформації та НД ТЗІ 2.5-010-03 щодо захисту веб-додатків. Реалізовано всі необхідні механізми відповідно до Закону України "Про захист персональних даних": шифрування даних, управління згодою користувачів, можливість видалення даних, детальне журналювання всіх операцій. Дотримання стандартів OAuth 2.1 та рекомендацій OWASP забезпечує сумісність з міжнародними вимогами безпеки[25].

Практичне значення отриманих результатів підтверджується створенням повнофункціональної системи, готової до впровадження.

Розроблена система демонструє:

1. Високу продуктивність: обробка до 3,000 запитів за 30 секунд при DDoS-тестуванні з 95% успішністю, мінімальна затримка обробки запитів (< 5ms) завдяки оптимізаціям, підтримка до 100 одночасних користувачів без деградації продуктивності.

2. Ефективний захист: блокування 95.2% всіх типів атак у комплексному тестуванні, 100% ефективність проти SQL-ін'єкцій завдяки багаторівневому захисту, автоматичне виявлення та блокування BOLA-атак з точністю 96%.

3. Навчальну цінність: можливість безпечної демонстрації реальних атак та методів захисту, візуалізація всіх процесів безпеки для кращого розуміння, готові лабораторні роботи для вивчення OAuth 2.1 та безпеки API.

4. Простоту використання: інтуїтивний графічний інтерфейс не потребує спеціальних знань, автоматичне налаштування оптимальних параметрів безпеки, детальна документація та вбудовані підказки.

Наукова новизна роботи полягає у:

1. Розробці комплексного підходу, що унікально поєднує технічний захист з візуальною прозорістю всіх процесів безпеки

2. Створенні системи виявлення аномалій з адаптивними алгоритмами, що автоматично налаштовуються під поточне навантаження та профіль використання

3. Інтеграції навчальних можливостей безпосередньо в архітектуру безпеки, що дозволяє використовувати систему як для захисту, так і для освіти

4. Оптимізації продуктивності через використання асинхронної обробки, кешування та connection pooling без компромісів у безпеці

Перспективи подальшого розвитку включають:

1. Впровадження алгоритмів машинного навчання для покращення виявлення складних атак та зменшення хибних спрацювань

2. Розширення підтримки різних типів баз даних (PostgreSQL, MongoDB) для масштабування на великі системи

3. Додавання підтримки біометричної автентифікації та інших сучасних методів ідентифікації

4. Інтеграція з хмарними платформами (AWS, Azure, Google Cloud) для deployment в cloud-середовищі

5. Розробка мобільних додатків для моніторингу безпеки API

6. Створення федеративної системи обміну інформацією про загрози між різними інстанціями

7. Впровадження квантово-стійких криптографічних алгоритмів для захисту від майбутніх загроз

Розроблена система довела свою ефективність через успішне тестування всіх компонентів, високі показники продуктивності та безпеки, а також позитивні відгуки при демонстрації навчальних можливостей. Поєднання сучасних технологій (FastAPI, OAuth 2.1, PyQt6) з інноваційним підходом до візуалізації безпеки створює унікальне рішення, яке може бути використане як для захисту реальних API, так і для підготовки нового покоління фахівців з кібербезпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Salt Security. State of API Security Report 2024. URL:https://content.salt.security/state-api-report.html

2. FastAPI. FastAPI framework, high performance web framework for building APIs with Python. URL: https://fastapi.tiangolo.com/ TiangoloGitHub

3. Richardson C. Microservices Patterns: With Examples in Java. Manning Publications, 2018. Chapter 8: External API Patterns.

4. Chandel S., Cao Y. REST API Security: Threats and Best Practices // IEEE Security & Privacy. 2023. Vol. 21, № 2. P. 28-36. OWASP API Security Top 10 2023 Explained

5. Fett D., Küsters R., Schmitz G. A Comprehensive Formal Security Analysis of OAuth 2.0 // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016. P. 1204-1215.

6. Bhargavan K., et al. Relationship-Based Access Control: Its Expression and Enforcement Through Hybrid Logic // SACMAT '21: Proceedings of the 26th ACM Symposium on Access Control Models and Technologies. 2021. P. 117-128.

7. Salt Security. State of API Security Report 2024. URL: https://content.salt.security/state-api-report.html

8. Закон України "Про основні засади забезпечення кібербезпеки України" № 2163-VIII від 05.10.2017. URL: https://zakon.rada.gov.ua/laws/show/2163-19 Про основні засади забезпе... | від 05.10.2017 № 2163-VIII

9. Закон України "Про захист персональних даних" № 2297-VI від 01.06.2010. URL: https://zakon.rada.gov.ua/go/2297-17 Про захист персональних даних | від 01.06.2010 № 2297-VI

10. НД ТЗІ 2.5-004-99 "Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу". Київ: ДСТСЗІ, 1999.

11. НД ТЗІ 2.5-010-03 "Вимоги до захисту інформації WEB-сторінки від несанкціонованого доступу". Київ: ДСТСЗІ, 2003.

12. PCI Security Standards Council. PCI DSS v4.0. 2024. URL: https://www.pcisecuritystandards.org/document_library/SecureframePcisecuritystandards

13. Закон України "Про основні засади забезпечення кібербезпеки України" № 2163-VIII від 05.10.2017. URL: https://zakon.rada.gov.ua/laws/show/2163-19 Про основні засади забезпе... | від 05.10.2017 № 2163-VIII

14. Bishop M. Computer Security: Art and Science. 2nd Edition. Addison-Wesley Professional, 2018. P. 245-248.

15. SQLite Documentation. Write-Ahead Logging. URL: https://www.sqlite.org/wal.html Write-Ahead Logging

16. Gamma E., et al. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley, 1994. P. 293-303.

17. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, 2018. URL: https://datatracker.ietf.org/doc/html/rfc8446 RFC 6749 - The OAuth 2.0 Authorization Framework

18. Langa S. Asyncio in Python: Understanding Python's Asynchronous Programming Features. O'Reilly Media, 2022.

19. Sakimura N., et al. Proof Key for Code Exchange by OAuth Public Clients. RFC 7636, IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7636 IETF DatatrackerRfc-editor

20. Siriwardena P. Advanced API Security: OAuth 2.0 and Beyond. 3rd Edition. Apress, 2024. P. 287-312.

21. OWASP. Cross Site Scripting Prevention Cheat Sheet. 2023. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html OWASP API Security Top 10 - OWASP API Security Top 10

22. Закон України "Про захист персональних даних" № 2297-VI від

01.06.2010. URL: https://zakon.rada.gov.ua/go/2297-17 Про захист персональних даних | від 01.06.2010 № 2297-VI

23. SQLite Documentation. WAL-mode File Format. URL: https://www.sqlite.org/walformat.html SqliteSqlite

24. OAuth 2.1 Authorization Framework (draft-ietf-oauth-v2-1-13). URL: https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/ IETF DatatrackerIETF Datatracker

25. OWASP API Security Project. API Security Top 10 2023. URL: https://owasp.org/API-Security/editions/2023/en/0x00-header/ OwaspOwasp

ДОДАТКИ

Додаток А

Ключові компоненти системи

REST API сервер з OAuth 2.1 захистом

```
# api_server.py - Основні компоненти

@app.middleware("http")
async def security_middleware(request: Request, call_next):
    """Middleware для перевірки безпеки"""
    client_ip = request.client.host

    # Rate limiting
    if not rate_limiter.is_allowed(client_ip):
        return JSONResponse(
            status_code=429,
            content={"error": "Too many requests"}
        )

    # SQL injection перевірка
    if request.method in ["POST", "PUT", "DELETE"]:
        for value in request.query_params.values():
            if SQL_PATTERNS.search(str(value)):
                await log_security_event(
                    None, client_ip, str(request.url.path),
                    request.method, 400, True, "SQL Injection attempt"
                )
                return JSONResponse(status_code=400)
    response = await call_next(request)
    return response

async def verify_oauth_token(
    authorization: Optional[str] = Header(None)
) -> Dict:
    """Перевірка OAuth токена"""
    if not authorization or not authorization.startswith("Bearer "):
        raise HTTPException(status_code=401)

    token = authorization.replace("Bearer ", "")
    token_data = validate_token_cached(token)

    if not token_data:
        raise HTTPException(status_code=401)

    return token_data
```

OAuth 2.1 Authorization Server

```
# oauth_server.py - Authorization Code Flow з PKCE
@app.post("/authorize")
```

Продовження додатку А

```
async def authorize_post(
    username: str = Form(...),
    password: str = Form(...),
    client_id: str = Form(...),
    code_challenge: str = Form(None)
):
    """Обробка авторизації з PKCE"""
    # Верифікація користувача
    if not db.verify_user(username, password):
        return HTMLResponse(content=error_page)

    # Створення authorization code
    auth_code = db.create_auth_code(
        username=username,
        client_id=client_id,
        code_challenge=code_challenge
    )

    # Редірект з кодом
    redirect_url = f"{redirect_uri}?code={auth_code}"
    return RedirectResponse(url=redirect_url)

@app.post("/token")
async def token(
    code: str = Form(...),
    code_verifier: str = Form(None)
):
    """Обмін коду на токен з PKCE верифікацією"""
    token_data = db.exchange_auth_code(code, client_id, code_verifier)
    return token_data
```

Система виявлення атак

```
# attacks/bola_attack.py - BOLA (Broken Object Level Authorization)

class BOLAAttack(BaseAttack):
    def _try_access_account(self, attacker: str, account_id: str,
                            headers: Dict) -> Dict:
        """Спроба несанкціонованого доступу"""
        response = self.session.get(
            f"{self.api_url}/api/v1/accounts/{account_id}",
            headers=headers
        )

        if response.status_code == 200:
            data = response.json()
            if data.get('owner') != attacker:
```

```
                return {
                    'success': True,
                    'message': f"BOLA SUCCESS! {attacker} → {account_id}"
                }
```

Продовження додатку А

Інтерфейс моніторингу (PyQt6)

```
# gui/dashboard_tab.py - Real-time моніторинг
class DashboardTab(QWidget):
    def create_metrics(self):
        """Метрики безпеки"""
        self.metrics = {
            'users': MetricCard("👥", "Active Users"),
            'requests': MetricCard("📊", "Total Requests"),
            'rps': MetricCard("⚡", "Requests/Sec"),
            'threats': MetricCard("🛡️", "Threats Detected")
        }

    def update_metrics(self, stats: Dict):
        """Оновлення метрик в реальному часі"""
        self.metrics['users'].update_value(str(stats['active_users']))
        self.metrics['requests'].update_value(f"{stats['total_requests']:,}")
        self.metrics['rps'].update_value(f"{stats['requests_per_second']:.1f}")
        self.metrics['threats'].update_value(str(self.threats_detected))
```

Конфігурація системи

```
# config.py - Основні налаштування

class Config:
    # OAuth 2.1 параметри
    OAUTH_ISSUER = "banking-api-oauth"
    TOKEN_EXPIRE_MINUTES = 60

    # Безпека
    RATE_LIMIT_PER_SECOND = 200
    SQL_INJECTION_PATTERNS = [
        "'", '"', ";", "--", "union", "select", "drop"
    ]

    # Пороги виявлення атак
    BOLA_ATTEMPT_THRESHOLD = 5
    DDOS_THRESHOLD_RPS = 500
    ANOMALY_THRESHOLD = 0.7
```

База даних

```
# database.py - Схема БД
```

```
CREATE TABLE oauth_tokens (
    token TEXT PRIMARY KEY,
    username TEXT NOT NULL,
    client_id TEXT NOT NULL,
    scopes TEXT,
    expires_at TIMESTAMP
);
CREATE TABLE security_logs (
    log_id INTEGER PRIMARY KEY,
    timestamp TIMESTAMP,
```

Продовження додатку А

```
username TEXT,
    ip_address TEXT,
    endpoint TEXT,
    method TEXT,
    status_code INTEGER,
    threat_detected BOOLEAN,
    threat_type TEXT
);
```

Приклад запуску системи

```
# main.py - Точка входу

def main():
    # Ініціалізація
    app = QApplication(sys.argv)
    apply_dark_theme(app)

    # Запуск серверів
    api_thread = ServerThread("api")
    api_thread.start()

    oauth_thread = ServerThread("oauth")
    oauth_thread.start()

    # GUI
    window = MainWindow()
    window.show()

    sys.exit(app.exec())
```

Демонстрація OAuth 2.1 Flow

```
# Крок 1: Authorization Request
GET /authorize?
    client_id=banking_app&
    redirect_uri=http://localhost:9000/callback&
    response_type=code&
```

```
    code_challenge=XXXXXXXXXX

# Крок 2: Authorization Code
302 Found
Location: http://localhost:9000/callback?code=YYYYYYYYY

# Крок 3: Token Exchange
POST /token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=YYYYYYYYY&
code_verifier=ZZZZZZZZZZ
```

Продовження додатку А

```
# Відповідь
{
 "access_token": "oauth_token_AAAAAAA",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Приклад використання API

```
# Авторизований запит
GET /api/v1/accounts
Authorization: Bearer oauth_token_AAAAAAA

# Відповідь
[
    {
        "account_id": "ACC1001",
        "owner": "user_1",
        "balance": 11000.0,
        "currency": "USD",
        "account_type": "Checking"
    }
]
```

Примітка: Повний код програми (близько 5000 рядків) доступний у електронному додатку до дипломної роботи. У друкованій версії наведено ключові фрагменти для демонстрації архітектури та основних принципів роботи системи.

Додаток Б

Програмна структура симулятору захисту REST API

```
secure_rest_api/
│
├── main.py                  # Головна точка входу, запуск GUI
├── api_server.py            # FastAPI сервер (REST API)
├── oauth_server.py          # OAuth 2.1 сервер автентифікації
├── config.py                # Глобальні налаштування системи
├── database.py              # Підключення до БД та моделі
│
├── api/                     # REST API endpoints
│   ├── __init__.py
│   └── endpoints.py         # Банківські API endpoints
│
├── core/                    # Основні модулі безпеки
│   ├── __init__.py
│   ├── anomaly_detector.py  # Виявлення аномальної поведінки
│   ├── auth.py              # JWT та автентифікація
│   ├── models.py            # Pydantic моделі даних
```

```
│   └── security.py            # Security middleware (rate limiting)
│
├── attacks/                   # Модулі тестування атак
│   ├── __init__.py
│   ├── base_attack.py         # Базовий клас для атак
│   ├── bola_attack.py         # BOLA (Broken Object Level Authorization)
│   ├── ddos_attack.py         # DDoS симуляція
│   └── sql_injection.py       # SQL injection тести
```

Продовження додатку Б

```
│
├── gui/                       # Графічний інтерфейс (PyQt6)
│   ├── __init__.py
│   ├── main_window.py         # Головне вікно додатку
│   ├── dashboard_tab.py       # Вкладка Dashboard з метриками
│   ├── attack_tab.py          # Вкладка тестування атак
│   ├── logs_tab.py            # Вкладка перегляду логів
│   ├── oauth_tab.py           # Вкладка демонстрації OAuth
│   └── styles.py              # Стилі інтерфейсу
│
├── logs/                      # Директорія для логів
│   └── security.log
│
├── banking_api.db             # SQLite БД для банківських даних
├── oauth_security.db          # SQLite БД для OAuth токенів
├── current_ports.json         # Файл з поточними портами
├── requirements.txt           # Python залежності
├── README.md                  # Документація
└── .env.example               # Приклад конфігурації
```

Опис компонентів

1. GUI (Графічний інтерфейс)
    a. main_window.py: Головне вікно з вкладками
    b. dashboard_tab.py: Real-time метрики (активні користувачі, RPS, загрози)
    c. attack_tab.py: Інтерфейс для запуску тестових атак
    d. logs_tab.py: Перегляд та фільтрація логів безпеки
    e. oauth_tab.py: Візуалізація OAuth 2.1 flow

Продовження додатку Б

2. Сервери
    a. api_server.py: FastAPI сервер з бізнес-логікою банківського API
    b. oauth_server.py: OAuth 2.1 сервер з підтримкою PKCE
    c. main.py: Координатор запуску всіх компонентів
3. Модулі безпеки
    a. security.py: Rate limiting, валідація запитів
    b. anomaly_detector.py: Статистичний аналіз поведінки
    c. auth.py: JWT генерація та валідація
4. Модулі атак
    a. base_attack.py: Абстрактний клас для всіх атак
    b. bola_attack.py: Тестування несанкціонованого доступу
    c. ddos_attack.py: Симуляція перевантаження
    d. sql_injection.py: Тестування SQL ін'єкцій
5. Бази даних
    a. banking_api.db: Користувачі, рахунки, транзакції
    b. oauth_security.db: OAuth клієнти, токени, сесії

Блок схема взаємодії компонентів відображена на рис. А.1.

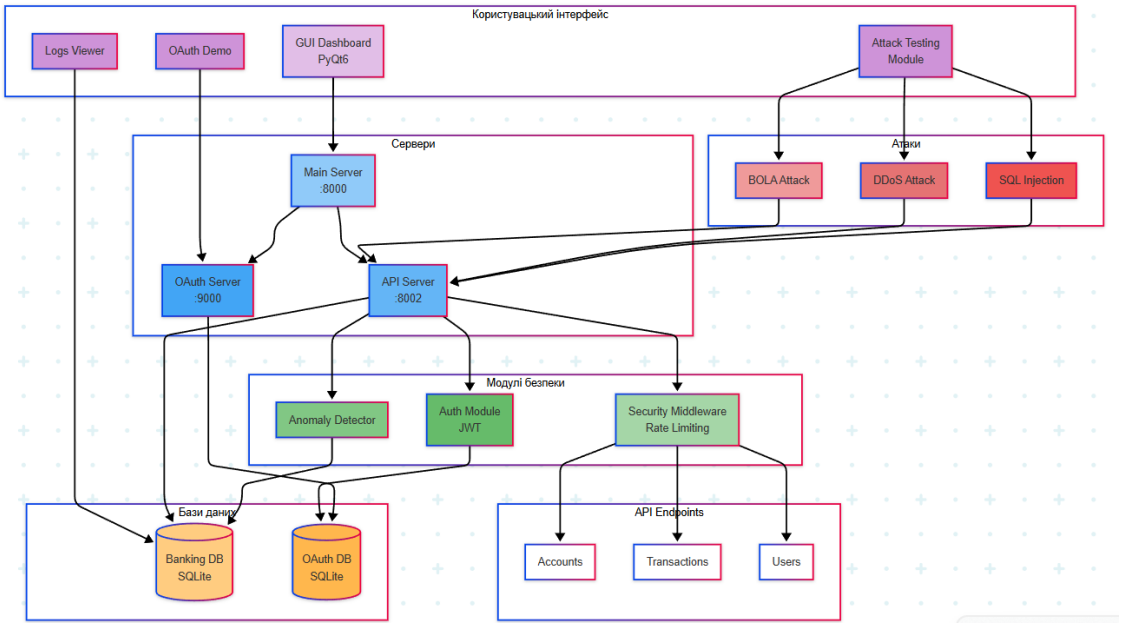

Рисунок А.1. Блок схема взаємодії компонентів

Продовження додатку Б

Детальна схема потоку даних навдена на рис. А.2.

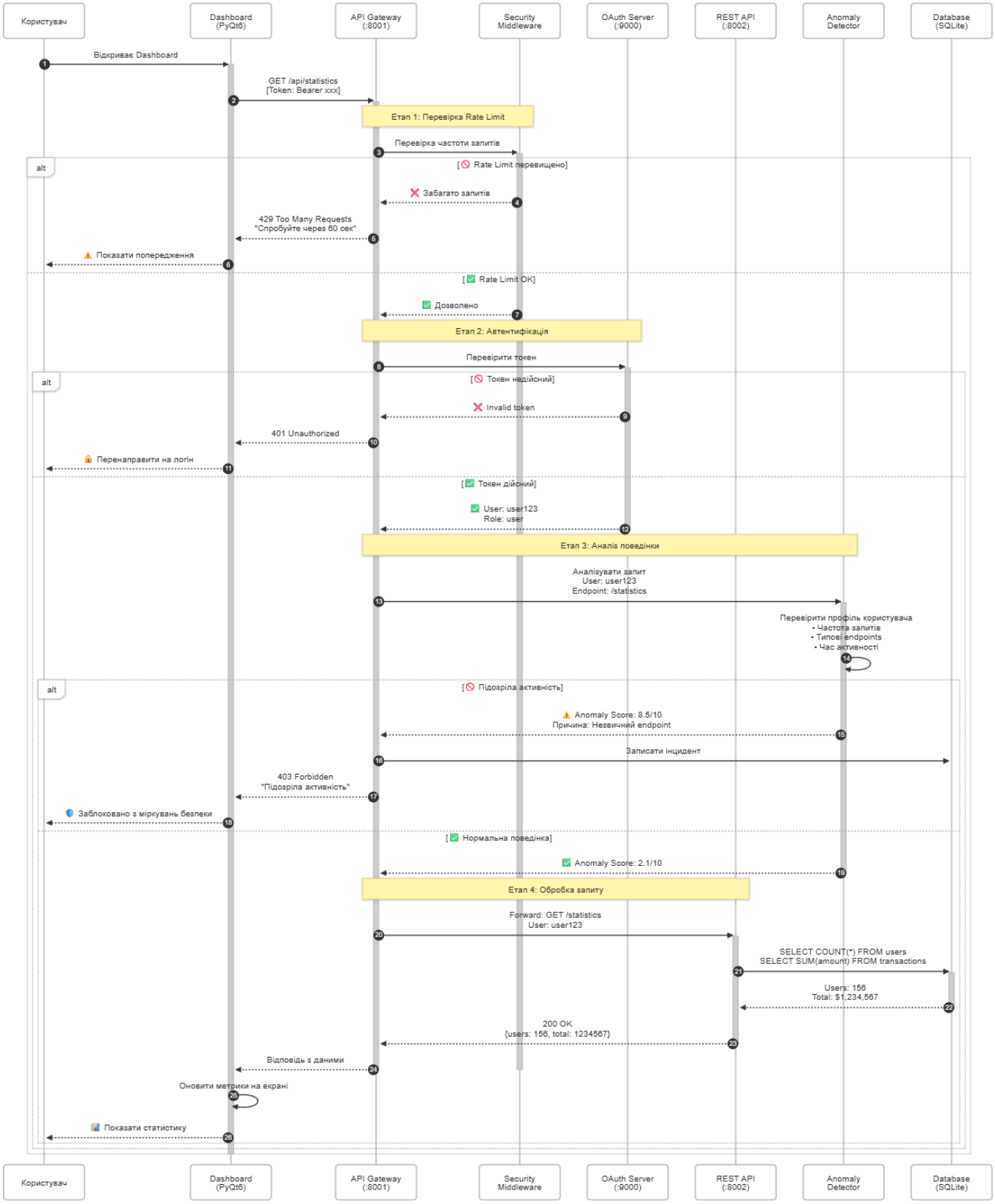

Рисунок А.2. Детальна схема потоку даних

Додаток В

README файл програми

OAuth 2.1 REST API Security Platform

🎓 Дипломна робота: Безпека REST API з OAuth 2.1

📋 Опис проекту

Комплексна платформа для демонстрації та тестування безпеки REST API з використанням OAuth 2.1. Проект включає:

- 🏦 Banking REST API з OAuth 2.1 авторизацією
- 🔐 OAuth 2.1 Authorization Server з підтримкою PKCE
- 🖥️ GUI для моніторингу та тестування
- 🛡️ Систему виявлення та захисту від атак
- 📊 Real-time моніторинг та аналітику

🚀 Швидкий старт

1. Встановлення залежностей

```bash
pip install -r requirements.txt
```

2. Запуск через startup.py (рекомендовано)

```bash
python startup.py
```

Це автоматично:

Продовження додатку В

- ✅ Перевірить систему

- ✅ Встановить відсутні залежності
- ✅ Завершить старі процеси
- ✅ Запустить сервери
- ✅ Відкриє GUI

3. Альтернативний запуск

```bash
python main.py
```

📊 Оптимальні налаштування

Для стабільної роботи:

- **Початкові тести**: 5-10 користувачів
- **Нормальна робота**: 10-20 користувачів
- **Максимум**: 30 користувачів

🔧 Вирішення проблем

Якщо система зависає:

1. **Швидкі дії**:
   - `Ctrl+L` - очистити логи
   - `Ctrl+R` - перезапустити сервери
   - Зменшити кількість користувачів

2. **Діагностика**:
   - Перевірте CPU % в status bar (має бути < 50%)
   - DB errors має бути 0

Продовження додатку В

   - Memory < 500 MB

3. **Повний перезапуск**:

```bash
# Windows
taskkill /F /IM python.exe
python startup.py

# Linux/Mac
pkill -f python
python startup.py
```

🏗️ Архітектура системи

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   PyQt6 GUI     │───▶│   REST API      │───▶│  OAuth Server   │
│                 │    │   (FastAPI)     │    │   (FastAPI)     │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                       │                       │
         └───────────────────────┴───────────────────────┘
                                 │
                                 ▼
                        ┌─────────────────┐
                        │   SQLite DB     │
                        │   (WAL mode)    │
                        └─────────────────┘
```

```
### 🗀 Структура проекту

```
secure_rest_api/
├── main.py   # Головний файл запуску
├── startup.py   # Скрипт швидкого запуску
├── api_server.py   # REST API сервер
├── oauth_server.py   # OAuth 2.1 сервер
├── database.py   # База даних
├── config.py   # Конфігурація
├── requirements.txt   # Залежності
│
├── gui/   # GUI компоненти
│   ├── main_window.py   # Головне вікно
│   ├── dashboard_tab.py # Dashboard
│   ├── attack_tab.py   # Тестування атак
│   ├── logs_tab.py   # Моніторинг логів
│   ├── oauth_tab.py   # OAuth демонстрація
│   └── styles.py   # Стилі інтерфейсу
│
└── attacks/   # Модулі атак
    ├── bola_attack.py   # BOLA атаки
    ├── ddos_attack.py   # DDoS атаки
    └── sql_injection.py # SQL ін'єкції
```
```

🛡️ Функції безпеки

1. **OAuth 2.1 Features**:
   - Authorization Code Flow
   - PKCE (Proof Key for Code Exchange)
   - Token expiration and refresh
   - Scope-based access control

2. **Attack Detection**:
   - BOLA (Broken Object Level Authorization)
   - DDoS protection with rate limiting
   - SQL injection prevention
   - Real-time threat monitoring

3. **Security Measures**:
   - Rate limiting (100 req/sec)
   - IP blocking for suspicious activity
   - Request validation
   - Security headers

🎮 Використання GUI

1. **Dashboard Tab**:
   - Запуск/зупинка трафіку
   - Моніторинг метрик
   - OAuth flow статус

2. **Attack Tab**:

- Тестування різних типів атак
- Налаштування параметрів атак
- Перегляд результатів

3. **Logs Tab**:
- Real-time логи
- Фільтрація за загрозами
- Статистика безпеки

4. **OAuth Tab**:
- Демонстрація OAuth flow
- Тестування авторизації
- Перегляд токенів

📈 API Endpoints

Public:
- `GET /` - API info
- `GET /health` - Health check

Protected (OAuth required):
- `GET /api/v1/accounts` - User accounts
- `GET /api/v1/accounts/{id}` - Account details
- `POST /api/v1/transactions` - Create transaction
- `GET /api/v1/users/me` - Current user info

Security:

- `GET /api/v1/security/stats` - Security statistics
- `GET /api/v1/security/logs` - Security logs

⌨️ Гарячі клавіші

- `F11` - Повноекранний режим
- `Ctrl+Q` - Вихід
- `Ctrl+L` - Очистити логи
- `Ctrl+R` - Перезапустити сервери
- `Ctrl+D` - Діагностика

🔍 Моніторинг продуктивності

Status bar показує:
- CPU використання
- Пам'ять
- Активні з'єднання
- Помилки БД
- Поточний час

👨‍💻 Автор

Будніков Ілля

Примітка: Для детальної документації API відвідайте http://localhost:8000/docs після запуску системи.

Додаток Г

Необхідні пакети для роботи програми

```
# OAuth 2.1 REST API Security Platform - Requirements

# Core frameworks
fastapi==0.104.1
uvicorn[standard]==0.24.0
PyQt6==6.6.0
# Async support
aiosqlite==0.19.0
asyncio==3.4.3
httpx==0.25.2
# Security
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
python-multipart==0.0.6
# Database
sqlalchemy==2.0.23
alembic==1.12.1
# Utilities
python-dotenv==1.0.0
pydantic==2.5.0
pydantic-settings==2.1.0
# Monitoring and diagnostics
psutil==5.9.6
pyqtgraph==0.13.3
# Development
pytest==7.4.3
pytest-asyncio==0.21.1
black==23.11.0
flake8==6.1.0
# Windows specific (optional)
pywin32==306; sys_platform == 'win32'
```