МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ
«КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри кібербезпеки

_____Анна ІЛЬЄНКО
"_____"_____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"

Тема: Система аналізу архітектури програмних застосунків з метою виявлення вразливостей пов'язаних з використанням штучного інтелекту

Виконавець: Вікторія ПОГОРІЛА

Керівник: к.т.н. Олена ВИСОЦЬКА

Нормоконтролер: к.т.н., доцент Андрій ПЕТРЕНКО

Київ 2024

ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ
«КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет комп'ютерних наук та технологій
Кафедра кібербезпеки
Освітній ступінь магістр
Спеціальність 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки

_____Анна ІЛЬЄНКО
«30»___08_____2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Погорілої Вікторії Миколаївни

1. Тема кваліфікаційної роботи: Система аналізу архітектури програмних застосунків з метою виявлення вразливостей пов'язаних з використанням штучного інтелекту
затверджена наказом ректора від 30.08.2024 №1696/ст.
2. Термін виконання роботи: з 30.08.2024 по 15.12.2024
3. Вихідні дані до роботи: проаналізувати вразливості програмних застосунків, зокрема пов'язані з використанням штучного інтелекту, та архітектурні особливості, що сприяють їх виникненню, на основі результатів проведеного аналізу обрати оптимальні методи та технології для розробки системи, спрямованої на виявлення таких вразливостей, розробити програмне рішення для аналізу архітектури та виявлення вразливостей програмних застосунків, протестувати розроблену систему на контрольних сценаріях.
4. Зміст пояснювальної записки: аналіз вразливостей програмних застосунків та їх архітектури, методи та технології архітектури програмних застосунків, вибір

методів та технологій для системи аналізу архітектури програмних застосунків з метою виявлення вразливостей пов’язаних з використанням штучного інтелекту, розробка та тестування розробленої системи.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: презентація.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Провести аналіз вразливостей програмних застосунків та їх архітектури	30.08.2024 – 05.09.2024	*Виконано*
2.	Провести аналіз методів та технологій аналізу програмних застосунків	06.09.2024 – 10.09.2024	*Виконано*
3.	Обґрунтувати вибір методу машинного навчання та тренувального набору даних для виявлення вразливостей в архітектурі програмних застосунків	11.09.2024 – 25.09.2024	*Виконано*
5.	Розробити математичну модель вирішення задачі	26.09.2024 – 01.10.2024	*Виконано*
6.	Побудувати алгоритми для навчання моделей та виявлення вразливостей	02.10.2024 – 07.10.2024	*Виконано*
7.	Розробити програмне рішення аналізу архітектури за компонентами, яке буде виявляти вразливості пов’язані зі штучним інтелектом	08.10.2024 – 03.11.2024	*Виконано*
8.	Протестувати систему: експерименти, їх опис та аналіз результатів	04.11.2024 – 15.11.2024	*Виконано*

7. Дата видачі завдання: «30» __08__ 2024 р.

Керівник кваліфікаційної роботи: _____ Олена ВИСОЦЬКА
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання: _____ Вікторія ПОГОРІЛА
(підпис здобувача вищої освіти) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система аналізу архітектури програмних застосунків з метою виявлення вразливостей пов’язаних з використанням штучного інтелекту»: загальний обсяг кваліфікаційної роботи складає 150 сторінок, основний текст роботи викладено на 118 сторінках., 45 рис., 8 табл., 50 літературних джерел.

Об’єктом дослідження є процес аналізу архітектури програмних застосунків з метою виявлення вразливостей пов’язаних з використанням штучного інтелекту.

Предметом дослідження є методи та засоби виявлення вразливостей в архітектурі програмних застосунків, з акцентом на використанні машинного навчання для аналізу вразливостей, які асоціюються зі штучним інтелектом.

Метою даної роботи є: розробити систему, яка дозволить виявляти вразливості в архітектурі програмних застосунків, що використовують штучний інтелект, із застосуванням сучасних методів машинного навчання. Досягнення цієї мети передбачає виконання таких основних завдань:

- проаналізувати існуючі вразливості програмних застосунків, зокрема пов’язані з використанням штучного інтелекту, та архітектурні особливості, що сприяють їх виникненню, визначити оптимальні методи і технології для розробки системи, а також обґрунтувати вибір методу машинного навчання для аналізу архітектури програмних застосунків;
- розробити систему для аналізу архітектури програмних застосунків на основі використання алгоритмів машинного навчання (за допомогою бібліотеки ML.NET), що дасть змогу автоматизувати процес виявлення вразливостей, особливо тих, які пов’язані з використанням штучного інтелекту;
- провести тестування розробленої системи на контрольних сценаріях, що дасть змогу оцінити її ефективність та надати рекомендації для вдосконалення.

Методи дослідження: аналіз літературних джерел (для визначення сучасних підходів до виявлення вразливостей у програмному забезпеченні та застосування методів штучного інтелекту), моделювання (для створення математичної моделі системи, що відображає процеси виявлення та класифікації вразливостей), експериментальний метод (для тестування системи на реальних та синтетичних наборах даних і оцінки її ефективності), статичний аналіз (для обробки та інтерпретації результатів експериментів з метою визначення точності, чутливості та специфічності системи), порівняльний аналіз (для зіставлення результатів з існуючими системами та оцінки конкурентоспроможності).

Практична цінність: розроблена система для автоматизованого аналізу вразливостей програмних застосунків з елементами штучного інтелекту, що дозволить забезпечити більш надійний захист даних, адаптивно реагувати на нові загрози та підвищити загальний рівень кібербезпеки в інформаційних системах різних організацій.

Наукова новизна: полягає у вдосконаленні підходів до виявлення вразливостей в архітектурі програмних застосунків шляхом інтеграції методів машинного навчання з алгоритмом L-BFGS, що дозволяє точно класифікувати загрози для систем, які використовують штучний інтелект, та адаптивно реагувати на нові типи вразливостей.

Результати роботи можуть бути використані для створення ефективних систем захисту, здатних ідентифікувати та класифікувати вразливості у програмних застосунках з використанням штучного інтелекту. Розроблене програмне забезпечення забезпечує інтеграцію з алгоритмами машинного навчання, що дозволяє адаптувати систему до нових загроз та забезпечувати актуальну безпеку для різноманітних програмних архітектур.

АНАЛІЗ ВРАЗЛИВОСТЕЙ, МАШИННЕ НАВЧАННЯ, ПРОГРАМНІ ЗАСТОСУНКИ, ШТУЧНИЙ ІНТЕЛЕКТ, КІБЕРБЕЗПЕКА, КЛАСИФІКАЦІЯ ЗАГРОЗ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ШІ	–	штучний інтелект.
CSRF	–	Cross-Site Request Forgery.
CVE	–	Common Vulnerabilities and Exposures.
CWE	–	Common Weakness Enumeration.
CVSS	–	Common Vulnerability Scoring System.
DoS	–	Denial of Service.
DDoS	–	Distributed Denial of Service.
L-BFGS	–	Limited-memory Broyden–Fletcher–Goldfarb–Shanno.
MITM	–	Man-In-The-Middle.
XSS	–	Cross-Site Scripting.
SCAP	–	Security Content Automation Protocol.

ВСТУП

Швидкий розвиток сучасних інформаційних технологій та їх інтеграція в усі сфери діяльності людини ставить нові виклики перед науковою спільнотою. Однією з найбільш значущих проблем є забезпечення надійності та безпеки програмних застосунків, особливо тих, що використовують методи штучного інтелекту. З розвитком машинного навчання та глибокого навчання, все більше програмних рішень базуються на автоматизованих системах прийняття рішень, які часто стають важливими компонентами критичних інфраструктур, таких як фінансові установи, охорона здоров'я, транспортні системи та державні установи [1]. Використання штучного інтелекту надає численні переваги, включаючи можливість аналізувати великі обсяги даних, підвищувати ефективність операцій та автоматизувати рутинні процеси. Однак, разом із впровадженням таких технологій зростає кількість загроз, пов'язаних із потенційними вразливостями в архітектурі таких програмних застосунків.

Однією з основних проблем є те, що з використанням штучного інтелекту збільшується складність програмних систем, що, в свою чергу, створює нові ризики для інформаційної безпеки. Часто вразливості залишаються невиявленими або ігноруються на стадії розробки через недостатню увагу до безпеки архітектури. Багато досліджень у цій сфері зосереджені на проблемах розпізнавання патернів атак або захисту від зовнішніх загроз, однак внутрішні вразливості, які виникають через неправильну або недостатньо ретельну архітектуру програмного застосунку, залишаються менш дослідженими [2]. Ці вразливості можуть включати неефективне використання ресурсів, некоректну роботу з даними, а також можливості для обходу безпекових механізмів, що створює додаткові ризики для організацій.

У контексті України, актуальність цього питання набуває ще більшої ваги з огляду на важливість розвитку інформаційних технологій у державному та приватному секторах. Захист інформаційних ресурсів є одним із пріоритетних напрямів у забезпеченні національної безпеки, особливо в умовах активних

кібератак та гібридної війни. Вразливості у програмних системах можуть призвести не лише до фінансових збитків, але й до компрометації критично важливих об’єктів інфраструктури. Окрім того, із зростанням кількості програмних рішень, які використовують штучний інтелект для автоматизації процесів у державному управлінні, економіці та оборонній сфері, необхідність у дослідженні архітектури таких систем стає вкрай важливою.

Аналіз існуючих методів і засобів захисту програмних систем показує, що велика частина досліджень фокусується на зовнішніх атаках або безпосередньо на захисті даних. Водночас, проблема захисту архітектури, яка використовується для інтеграції штучного інтелекту, залишається менш дослідженою. Існуючі рішення не завжди здатні виявляти або ефективно усувати внутрішні вразливості, що виникають на рівні архітектури, особливо коли мова йде про складні системи з машинним навчанням, де логіка роботи стає менш прозорою для розробників.

Необхідність дослідження архітектур програмних застосунків з використанням штучного інтелекту зумовлена зростаючою кількістю нових вразливостей, які звичайні методи захисту не можуть повністю вирішити. Критичний аналіз існуючих підходів до виявлення вразливостей та їх порівняння із сучасними методами проектування програмних систем, що базуються на штучному інтелекті, дає можливість виявити недоліки в наявних рішеннях та обґрунтувати доцільність розробки нових підходів для підвищення надійності таких систем.

Актуальність обумовлена стрімким розвитком штучного інтелекту та його впровадженням у критично важливі сфери, що потребує підвищеної уваги до безпеки програмних систем. Особливо актуальним це питання є в умовах зростання кількості атак на інформаційні ресурси, що створює ризики для фінансових, державних та інших систем, важливих для національної безпеки України. Таким чином, розробка нових підходів до аналізу архітектури програмних застосунків з використанням сучасних методів машинного навчання є актуальним завданням для забезпечення надійності та стійкості таких систем.

Об’єктом дослідження є процес аналізу архітектури програмних застосунків з метою виявлення вразливостей пов’язаних з використанням штучного інтелекту.

Предметом дослідження є методи та засоби виявлення вразливостей в архітектурі програмних застосунків, з акцентом на використанні машинного навчання для аналізу вразливостей, які асоціюються зі штучним інтелектом.

Метою даної роботи є: розробити систему, яка дозволить виявляти вразливості в архітектурі програмних застосунків, що використовують штучний інтелект, із застосуванням сучасних методів машинного навчання. Досягнення цієї мети передбачає виконання таких основних завдань:

- проаналізувати існуючі вразливості програмних застосунків, зокрема пов’язані з використанням штучного інтелекту, та архітектурні особливості, що сприяють їх виникненню, визначити оптимальні методи і технології для розробки системи, а також обґрунтувати вибір методу машинного навчання для аналізу архітектури програмних застосунків;
- розробити систему для аналізу архітектури програмних застосунків на основі використання алгоритмів машинного навчання (за допомогою бібліотеки ML.NET), що дасть змогу автоматизувати процес виявлення вразливостей, особливо тих, які пов’язані з використанням штучного інтелекту;
- провести тестування розробленої системи на контрольних сценаріях, що дасть змогу оцінити її ефективність та надати рекомендації для вдосконалення.

Методи дослідження використані у роботі: аналіз літературних джерел (для визначення сучасних підходів до виявлення вразливостей у програмному забезпеченні та застосування методів штучного інтелекту), моделювання (для створення математичної моделі системи, що відображає процеси виявлення та класифікації вразливостей), експериментальний метод (для тестування системи на наборах даних і оцінки її ефективності), статичний аналіз (для обробки та інтерпретації результатів експериментів з метою визначення точності та

чутливості системи), порівняльний аналіз (для зіставлення результатів з існуючими системами та оцінки конкурентоспроможності).

Галузь застосування: розроблена система може бути застосована в кібербезпеці, критичній інфраструктурі, фінансовому секторі, охороні здоров'я, державному управлінні, транспорті, логістиці та електронній комерції. Вона забезпечує виявлення вразливостей у програмних застосунках із елементами штучного інтелекту, захист конфіденційних даних, аналіз ризиків і підвищення безпеки критичних інфраструктур. Це робить її незамінною для галузей, де ШІ використовується для автоматизації, прогнозування та управління даними.

Наукова новизна одержаних результатів полягає у вдосконаленні підходів до виявлення вразливостей в архітектурі програмних застосунків шляхом інтеграції методів машинного навчання з алгоритмом L-BFGS, що дозволяє точно класифікувати загрози для систем, які використовують штучний інтелект, та адаптивно реагувати на нові типи вразливостей.

Практичне значення одержаних результатів полягає у можливості використання розробленої системи для автоматизованого аналізу вразливостей програмних застосунків з елементами штучного інтелекту, що дозволить забезпечити більш надійний захист даних, адаптивно реагувати на нові загрози та підвищити загальний рівень кібербезпеки в інформаційних системах різних організацій.

Апробація отриманих результатів: результати дослідження, включені до кваліфікаційної роботи, були оприлюднені на VI Всеукраїнській студентській конференції «Науковий простір: аналіз, сучасний стан, тренди та перспективи» у формі доповіді на тему «Аналіз архітектури програмних застосунків з метою виявлення вразливостей, пов'язаних з використанням штучного інтелекту».

Публікації: Погоріла В. М. АНАЛІЗ АРХІТЕКТУРИ ПРОГРАМНИХ ЗАСТОСУНКІВ З МЕТОЮ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ПОВ'ЯЗАНИХ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ [Електронний ресурс] / Вікторія Миколаївна Погоріла // І Всеукраїнська студентська наукова конференція «НАУКОВИЙ ПРОСТІР: АНАЛІЗ, СУЧАСНИЙ СТАН, ТРЕНДИ ТА

ПЕРСПЕКТИВИ». – 2024. – Режим доступу до ресурсу: https://archive.liga.science/index.php/conference-proceedings/issue/view/ukr-18.10.2024., с. 337-340.

РОЗДІЛ 1
АНАЛІЗ ВРАЗЛИВОСТЕЙ ПРОГРАМНИХ ЗАСТОСУНКІВ ТА ЇХ АРХІТЕКТУРИ

1.1 Аналіз вразливостей програмних застосунків

Програмна вразливість – це недолік у системі, який дозволяє зловмиснику обійти впроваджені засоби захисту. Помилки та баги завжди існували у світі програмного забезпечення; складні та масштабні програмні системи можуть містити велику кількість помилок, і деякі з них можуть бути використані зловмисниками для нанесення шкоди або отримання вигоди. Проблеми безпеки, викликані програмними вразливостями, можуть бути особливо небезпечними, коли йдеться про компрометацію приватної інформації, наприклад, даних про стан здоров'я пацієнтів.

Для протидії програмним вразливостям були створені інтернет-бази даних, які фіксують всі виявлені вразливості з метою інформування компаній і розробників. Платформа національної бази вразливостей зберігає, підтримує та поширює інформацію про вразливості, виявлені в реальних комп'ютерних системах [3]. Такі бази даних дозволяють вимірювати рівень безпеки та управляти вразливостями. Крім того, кожна вразливість може бути занесена в базу з унікальним ідентифікатором, що полегшує обмін інформацією про неї.

Національна база вразливостей (National Vulnerability Database, NVD) є однією з найбільших у світі баз даних вразливостей, яка розробляється та підтримується Національним інститутом стандартів і технологій США (NIST). Станом на 2024 рік, NVD містить понад 210 000 записів про вразливості, що регулярно оновлюються та класифікуються за різними параметрами, такими як рівень загрози, вплив на конфіденційність, цілісність, доступність та інші [3].

На рис. 1.1 зображена класифікація вразливостей, які містяться в Національній базі вразливостей, зокрема програмні, апаратні, мережеві та конфігураційні.

Рис. 1.1. Класифікація вразливостей

Усі вразливості поділяються на чотири основні категорії:

1. Програмні вразливості. Цей тип вразливостей стосується помилок або недоліків у програмному забезпеченні, які можуть бути використані для атак. До цієї категорії входять:

– переповнення буфера – ситуація, коли програма зберігає дані за межами виділеної їй пам'яті, що може призвести до аварійної зупинки або виконання зловмисного коду;
– SQL-ін'єкції – вразливість, що дозволяє впроваджувати зловмисні SQL-запити в програму, обминаючи захист бази даних;
– міжсайтові сценарії (XSS) – атака, що полягає у впровадженні зловмисного коду на веб-сторінки, що дозволяє викрадати інформацію або втручатися в роботу сайту.

2. Апаратні вразливості. Вразливості цього типу виникають через недоліки в апаратному забезпеченні, що можуть бути використані для атак на систему. До цієї категорії належать:

– вразливість процесорів (Meltdown) – проблема в архітектурі процесора, яка дозволяє зчитувати пам'ять інших процесів, навіть якщо вона захищена;

– вразливість спекулятивного виконання (Spectre) – експлуатація механізму спекулятивного виконання команд у процесорах, що дозволяє зловмиснику зчитувати конфіденційні дані.

3. Мережеві вразливості. Цей тип вразливостей стосується недоліків у налаштуванні та функціонуванні мережевих систем, що можуть бути використані для атак. До цієї категорії входять:

– атака посередника (MITM) – атака, при якій зловмисник перехоплює та змінює трафік між двома сторонами без їхнього відома;
– відмова в обслуговуванні (DoS) – атака, метою якої є виведення з ладу мережевого ресурсу шляхом надмірного навантаження системи запитами.

4. Конфігураційні вразливості. Ці вразливості виникають через неправильні або стандартні налаштування систем та їх компонентів. До цієї категорії входять:

– стандартні налаштування – вразливості, що виникають при використанні стандартних конфігурацій, які часто відомі зловмисникам і можуть бути легко експлуатовані;
– слабкі паролі – використання ненадійних або передбачуваних паролів, що полегшує доступ до системи.

Дані Національної бази вразливостей надаються відповідно до специфікацій протоколу автоматизації безпекового контенту (SCAP). SCAP є набором стандартів, розроблених NIST для стандартизованого представлення та обробки інформації про вразливості та налаштування систем [4].

Центральною частиною Національної бази вразливостей є CVE – словник вразливостей, що містить унікальні ідентифікатори для кожної відомої вразливості. У цьому словнику для кожної вразливості присвоюється спеціальний CVE-ідентифікатор, який дозволяє легко відслідковувати та класифікувати загрози. CVE слугує основою для класифікації всіх типів вразливостей у NVD [5]. Структура CVE включає кілька ключових елементів:

– CVE ID – унікальний ідентифікатор у форматі CVE-РІК-ЧИСЛО (наприклад, CVE-2024-12345);

– опис вразливості, що пояснює її суть;
– дата публікації, коли вразливість була додана до словника;
– посилання на джерела інформації або патчі для усунення вразливості.

Для оцінки рівня загрози використовується Common Vulnerability Scoring System (CVSS), яка присвоює кожній вразливості числову оцінку, що дозволяє визначити ступінь її критичності. Окрім того, вразливості класифікуються за допомогою Common Weakness Enumeration (CWE), що забезпечує необхідну структуру для категоризації програмних та системних слабких місць [6]. Це також відображено на схемі, де представлено різні типи вразливостей, такі як програмні, апаратні, мережеві та конфігураційні.

І CVE, і CWE управляються корпорацією MITRE, яка підтримує та розвиває ці стандарти в інтересах уряду США та міжнародної спільноти.

Вразливості інформаційних систем є предметом досліджень науковців у різних країнах світу, адже вони становлять загрозу як для окремих користувачів, так і для цілих організацій. Завдяки постійним зусиллям, фахівці розробляють методики для класифікації вразливостей та способів їх усунення. Так, наприклад, у своєму дослідженні Лі та ін. [7] класифікували вразливості за складністю їх ідентифікації, виправлення та експлуатації. Вразливості були поділені на такі категорії: легкі для ідентифікації та експлуатації (вразливості типу Bohr, BOV); складні для виявлення та експлуатації (вразливості, не пов’язані зі старінням, NMV); вразливості, що використовуються для погіршення продуктивності системи (вразливості, пов’язані зі старінням, ARV); і вразливості, які не потрапляють під жодну з трьох попередніх категорій (невідомі вразливості, UNK). Результати дослідження показали, що найпоширенішим типом вразливостей є NMV. Крім того, дослідження виявило, що середній час на виправлення вразливості залежно від її типу становить 66,8 днів для BOV, 70,5 днів для NMV та 60,6 днів для ARV.

Вентер та ін. [8] зосередилися на категоризації вразливостей за наступним процесом: збір джерел даних (наприклад, списку CVE); попередня обробка даних для додавання важливої інформації та виключення неактуальних вразливостей;

зберігання даних за допомогою самоорганізованої карти (SOM); та інспекція та маркування кластерів для класифікації бази даних CVE. У цьому дослідженні вразливості були поділені на сім категорій: переповнення буфера, атаки на відмову в обслуговуванні (DoS), метасимволи сценаріїв, підвищення привілеїв, пошкодження даних, збір інформації та вразливості конфігурації. Результати показали, що з 167 виявлених вразливостей найпоширенішими були атаки DoS (61 випадок), а найрідкіснішими – переповнення буфера (4 випадки).

Помилки типу переповнення буфера виникають, коли обсяг даних, що записуються в буфер, перевищує його ємність (рис. 1.2) [9]. Під час виконання програми дані можуть записуватися за межами виділеної області пам’яті. У результаті операції читання або запису можуть здійснюватися з адрес пам’яті, які виходять за межі буфера. Така вразливість дозволяє неавторизованим користувачам скористатися помилкою для виконання зловмисного коду, читання конфіденційної інформації або зміни потоку керування в програмі.

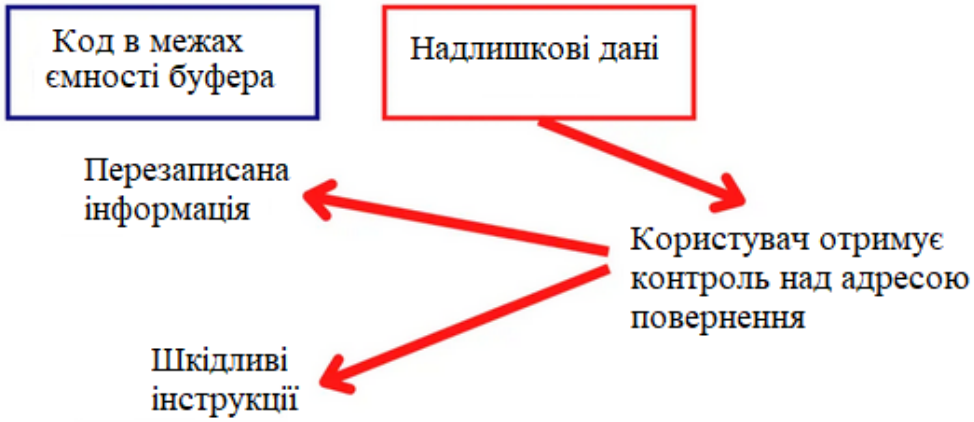

Рис. 1.2. Помилки типу переповнення буфера

Неправильне розподілення або використання системних ресурсів є ще однією поширеною вразливістю. Помилки цього типу можуть виникати через обмеженість пам’яті, простору на диску або обчислювальних потужностей CPU. Зазвичай такі помилки трапляються, коли важливі ресурси системи [10]:

– не вивільняються після завершення життєвого циклу програмного забезпечення;

– використовуються після попереднього вивільнення;

– не контролюються належним чином системою.

Ці проблеми виникають через нечітке розуміння того, який саме компонент програми відповідає за конкретні ресурси. Це створює «прогалину» або вразливість у системі, через яку зловмисники можуть виконати атаку. Такі атаки можуть призвести до відмови в обслуговуванні (DoS-атака), аварійного завершення роботи програми або виконання зловмисного коду. Ще одним потенційним наслідком таких помилок є атака, спрямована на виснаження пам’яті, що може суттєво уповільнити роботу програми та її операційної системи.

Вид вразливості програмного забезпечення Cross-Site Request Forgery (CSRF) належить до фішингових атак, де використовується соціальна інженерія для введення користувача в оману та викрадення його облікових даних [11]. На рис. 1.3 зображена схема, яка ілюструє процес цієї атаки.

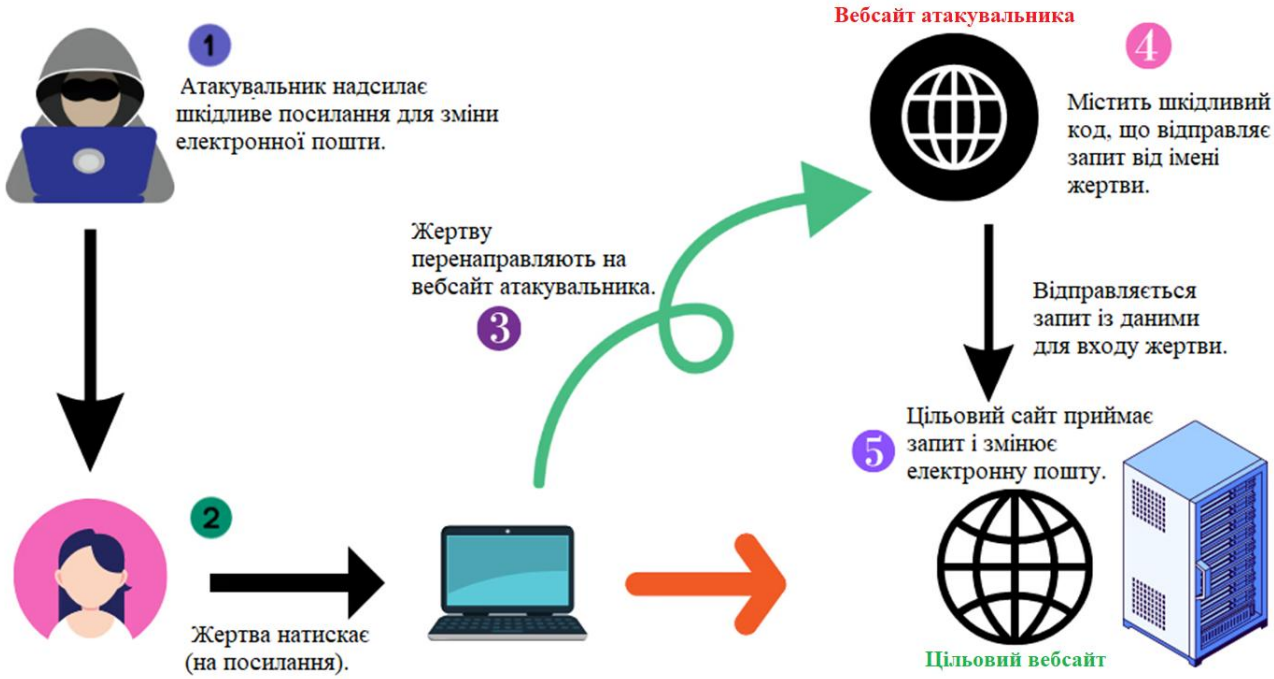

Рис. 1.3. Схема реалізації CSRF атаки

Спочатку атакувальник надсилає жертві зловмисне посилання, що імітує запит на зміну електронної пошти або іншої важливої інформації облікового

запису. Жертва, не підозрюючи небезпеки, натискає на це посилання і потрапляє на сайт атакувальника, який виглядає як легітимний. Цей сайт містить зловмисний код, який автоматично відправляє запит на зміну облікових даних від імені жертви. В результаті, легітимний сервер отримує запит, обробляє його і вносить зміни в обліковий запис жертви, не підозрюючи про несанкціоноване втручання.

На рис. 1.4 представлена DDoS-атака (Distributed Denial of Service), що здійснюється через ботнет. Спочатку атакувальник надсилає команду на запуск атаки з командного та контрольного сервера [12]. Цей сервер керує великою кількістю інфікованих пристроїв, відомих як ботнет, які можуть складатися з сотень або тисяч хостів. Після отримання команди боти одночасно починають надсилати великий обсяг зловмисного трафіку на сервер жертви. Внаслідок цього сервер жертви перевантажується й стає нездатним обробляти легітимні запити від звичайних користувачів. Така атака може призвести до тимчасового або повного припинення роботи вебресурсу чи сервісу.

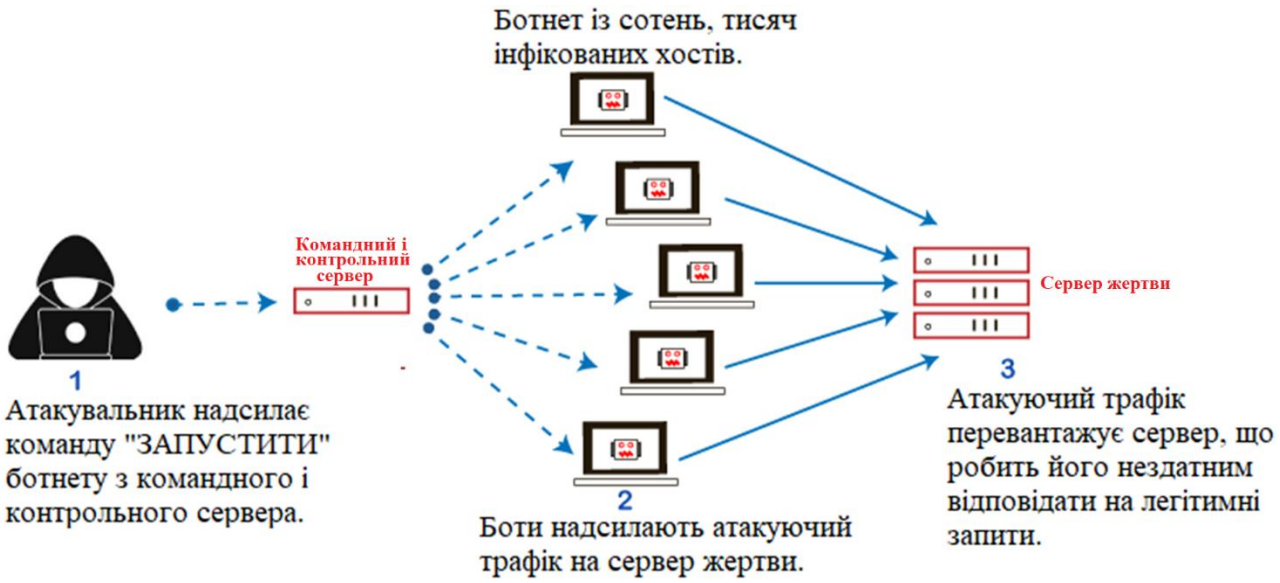

Рис. 1.4. Схема реалізації DDoS-атаки

Атака відома як Cross-Site Scripting (XSS), одна з найбільш поширених вразливостей вебдодатків, яка виникає внаслідок недостатнього контролю та перевірки даних, що вводяться користувачами [13]. Атака XSS дозволяє

зловмисникам впроваджувати зловмисний JavaScript-код у вебсторінки, які потім виконуються у браузері інших користувачів. Це може призвести до викрадення конфіденційних даних, таких як файли cookie, облікові дані користувачів, сесійні токени тощо. Зазвичай ця атака пов'язана з вебдодатками, які неправильно обробляють або фільтрують користувацькі введені дані, даючи можливість хакерам впроваджувати зловмисні скрипти.

На рис. 1.5 представлено схему XSS-атаки, яка демонструє основні етапи її виконання. Спочатку хакер надсилає зловмисний скрипт на сервер, що обробляє вебсайт. Цей зловмисний код впроваджується у вміст вебсторінки, наприклад у форму введення або інші елементи сторінки, які не проходять належної перевірки на безпеку. Коли користувач заходить на заражену сторінку вебсайту, браузер завантажує її вміст разом із зловмисним JavaScript-кодом.

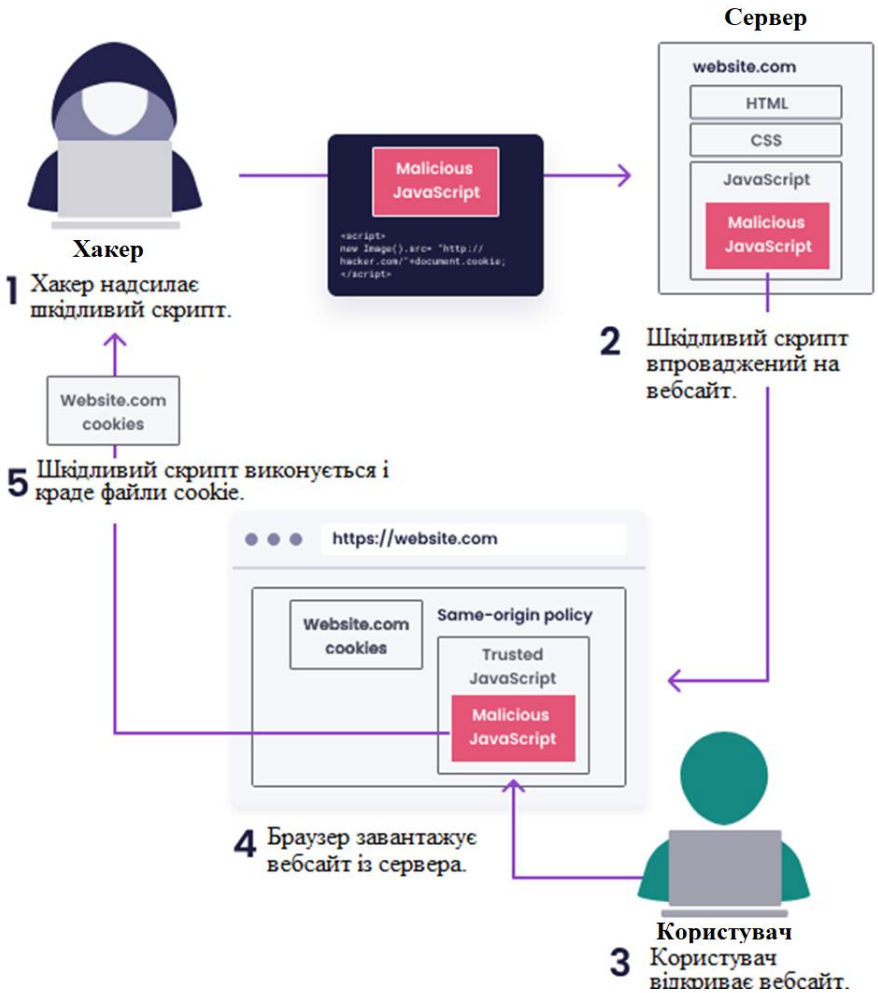

Рис. 1.5. Схема реалізації XSS -атаки

Як тільки сторінка завантажується, зловмисний код автоматично виконується в браузері користувача. У результаті цього коду можуть бути викрадені файли cookie або інші конфіденційні дані. Це особливо небезпечно, оскільки файли cookie зазвичай використовуються для аутентифікації сесій, що дозволяє зловмисникам перехопити сесію користувача і отримати доступ до його облікових записів без необхідності вводити логін чи пароль.

SQL-ін’єкція дозволяє зловмисникам отримати несанкціонований доступ до даних або керування базами даних вебдодатків [14]. Ця вразливість виникає, коли система неправильно обробляє введені користувачами дані і дає змогу зловмисникам впроваджувати зловмисні SQL-запити в операції, що виконуються сервером. SQL-ін’єкція пов’язана з неправильним забезпеченням безпеки на рівні бази даних, що може призвести до витоку конфіденційної інформації, маніпуляції з даними або повного руйнування системи.

На рис. 1.6 зображена схема цієї атаки. Спочатку зловмисник надсилає зловмисну SQL-команду на вебсервер, який відповідає за обробку запитів користувачів і взаємодію з базою даних.

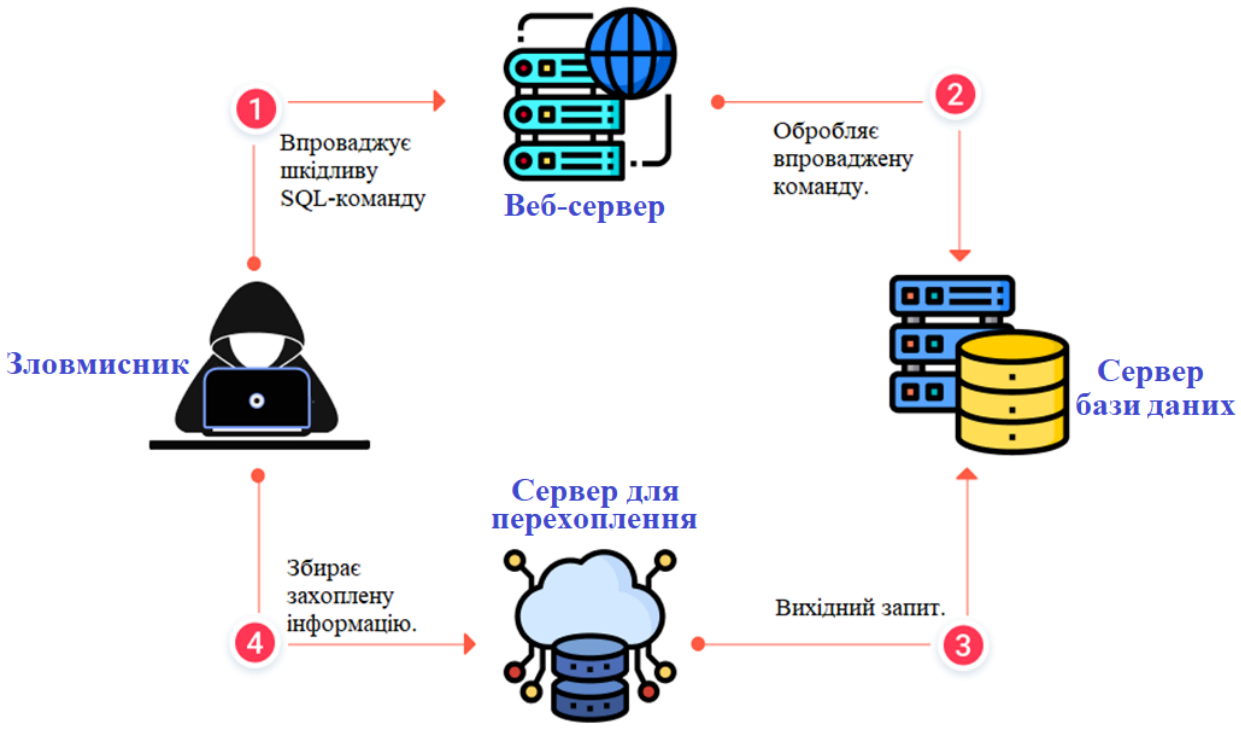

Рис. 1.6. Схема реалізації атаки SQL-ін’єкції

Вебсервер обробляє цей запит і передає його на сервер бази даних для виконання. Внаслідок неправильної обробки SQL-запиту сервер бази даних виконує зловмисну команду, що може призвести до витоку конфіденційної інформації або інших небажаних дій. Після цього захоплена інформація надсилається назад зловмиснику через вихідний запит. Зловмисник отримує доступ до конфіденційних даних або інших ресурсів бази даних через так званий сервер прослуховування, який збирає та фільтрує ці дані для подальшого використання.

SQL-ін’єкція є однією з найнебезпечніших вразливостей вебдодатків, оскільки вона може серйозно вплинути на конфіденційність, цілісність і доступність даних у базах даних, які використовуються вебсервером.

Підсумовуючи аналіз вразливостей програмних застосунків, можна стверджувати, що найбільш поширеними та небезпечними є DDoS атаки, атаки на основі SQL-ін’єкцій, XSS та інші методи, що експлуатують недостатній захист на рівні обробки користувацьких даних. Ці вразливості виникають через неправильну або недостатню перевірку введених даних, що дозволяє зловмисникам маніпулювати запитами до серверів або баз даних. Наслідки таких атак можуть бути дуже серйозними, включаючи витік конфіденційної інформації, руйнування даних або навіть повне захоплення контролю над системою. Для мінімізації ризиків необхідно впроваджувати надійні методи захисту, такі як валідація та фільтрація введених даних, використання захищених запитів та постійний моніторинг безпеки програмного забезпечення.

1.2 Аналіз вразливостей, пов’язаних з використанням штучного інтелекту

Штучний інтелект активно впроваджується в різні сфери сучасних інформаційних технологій, надаючи можливості для автоматизації процесів, аналізу великих обсягів даних та прийняття рішень у реальному часі. Програмні застосунки, що використовують ШІ, стали невід’ємною частиною таких галузей,

як медицина, фінансові послуги, транспорт, кібербезпека та багато інших. Завдяки здатності ШІ моделювати складні патерни і знаходити приховані закономірності, ці системи значно підвищують ефективність і точність роботи у відповідних сферах.

Однак, як і будь-які інші технології, штучний інтелект не позбавлений вразливостей. Специфіка ШІ, пов’язана з використанням великих обсягів даних і складних алгоритмів машинного навчання, породжує нові типи загроз, яких не було у класичних програмних системах. Вразливості можуть виникати на різних рівнях – від навчальних даних до самої моделі та її інтеграції в загальну архітектуру програмного застосунку. Більше того, звичайні методи кібербезпеки не завжди можуть забезпечити належний рівень захисту вразливих місць ШІ-систем.

У табл. 1.1 представлено основні вразливості програмних застосунків, що використовують штучний інтелект. Ці вразливості охоплюють широкий спектр загроз, які можуть впливати на коректність роботи моделей ШІ, їхню безпеку та надійність.

Таблиця 1.1.

Основні вразливості ПЗ із використанням ШІ

№	Назва вразливості	Опис	Приклади
1	Exec Code	Виконання довільного коду у системі.	Використання зловмисного коду для зміни моделі або впровадження зловмисного сценарію.
2	DoS	Відмова в обслуговуванні через перевантаження системи.	Атака на моделі ШІ з метою виведення їх з ладу під час обробки великих даних.
3	DoS Overflow	Переповнення буфера, що призводить до відмови в обслуговуванні.	Введення надмірно великих даних у модель для порушення її функціонування.
4	Bypass	Обхід механізмів автентифікації або контролю доступу.	Отримання доступу до моделей ШІ без відповідної авторизації.

Закінчення таблиці 1.1.

№	Назва вразливості	Опис	Приклади
5	Overflow	Переповнення буфера, що призводить до некоректної роботи системи.	Атака на моделі з метою викликати збій через переповнення вхідних даних.
6	Exec Code Overflow	Виконання коду через переповнення буфера.	Комбінована атака, що дозволяє виконувати зловмисний код через переповнення.
7	Dir. Trav.	Доступ до файлів поза межами дозволених директорій.	Перехоплення критичних даних через маніпуляції з файловою системою моделі ШІ.
8	XSS	Вставка зловмисних скриптів у веб-застосунки.	Використання зловмисних скриптів для зміни вхідних даних, що подаються в ШІ-модель.
9	SQL-ін’єкції	Ін’єкції SQL-запитів у бази даних.	Використання SQL-ін’єкцій для зміни або видалення навчальних даних моделі.
10	Exec Code XSS	Виконання коду через XSS-атаки.	Вставка зловмисного коду у веб-застосунок для доступу до моделі ШІ та її компрометації.
11	CSRF	Виконання небажаних дій користувачем через маніпуляцію запитами.	Атака на моделі через підроблені запити до сервера для виконання деструктивних дій.

Вибір цих вразливостей для подальшого дослідження ґрунтується на їхній поширеності та критичності в контексті сучасних програмних застосунків, що використовують штучний інтелект. Вразливості, представлені в табл. 1.1, охоплюють широкий спектр загроз, які є важливими для розуміння та усунення в межах безпеки ШІ-систем.

Такі вразливості, як Exec Code, DoS, Overflow і Bypass, стосуються базових аспектів безпеки програмного забезпечення, що працює з великими обсягами даних та складними алгоритмами. Використання ШІ робить ці системи привабливими для атак, оскільки порушення в обробці даних або виконанні кодів

може призвести до серйозних наслідків, як-от некоректні рішення або компрометація всієї системи. Системи, що автоматично приймають рішення на основі алгоритмів машинного навчання, особливо вразливі до таких загроз, тому вивчення цих типів вразливостей є необхідним для підвищення надійності ШІ.

Вразливості, пов’язані з SQL-ін’єкціями, XSS та CSRF, виявляють слабкі місця в аспекті взаємодії ШІ з зовнішніми джерелами даних та інтерфейсами. ШІ-системи, які обробляють великі обсяги інформації з баз даних або інтегруються з веб-застосунками, стають особливо вразливими до маніпуляцій з боку зловмисників. Ці загрози дозволяють маніпулювати вхідними даними, що безпосередньо впливає на результативність моделі та призводить до некоректних висновків.

Важливим фактором вибору цих вразливостей є їх універсальність: вони можуть проявлятися як у звичайних програмних системах, так і в системах з ШІ, але саме в останніх вони набувають специфічних форм та наслідків, що значно посилює їхню небезпеку. У середовищі ШІ моделі здатні адаптуватися до нових даних, що дає змогу атакувальникам впливати на внутрішні структури і поведінку системи за допомогою ворожих змін, наприклад, під час атак типу Model Extraction.

У таких випадках вразливості, як Exec Code Overflow, можуть бути використані для несанкціонованого модифікування навчальних моделей, а також для витоку конфіденційної інформації, яка використовується в процесі навчання. Це створює загрозу не лише для даних, але й для цілісності архітектури всієї системи. Тому саме універсальність цих загроз у поєднанні з унікальними проявами у середовищі ШІ обґрунтовує актуальність обраної теми дослідження. Дослідження цих загроз допоможе не тільки виявити слабкі місця в архітектурі сучасних програмних рішень, але й розробити нові методи захисту, що враховують специфіку роботи систем з машинним навчанням і штучним інтелектом.

1.3 Дослідження принципів побудови архітектур програмних застосунків

Архітектура програмного застосунку визначає структуру та спосіб взаємодії компонентів системи, створюючи базис для її ефективної роботи та розвитку. Вона встановлює, як саме компоненти будуть функціонувати разом, обмінюватися даними та виконувати завдання, забезпечуючи стабільність і надійність програмного забезпечення. Це проєктування дозволяє програмі не лише відповідати поточним вимогам, але й бути готовою до майбутніх змін або масштабування без значних переробок.

Правильно спроєктована архітектура значно спрощує підтримку та розширення системи, робить її гнучкою та здатною адаптуватися до нових вимог. Вона допомагає ізолювати зміни в одній частині системи від інших частин, що зменшує ризики при внесенні нових функцій або при налагодженні існуючих. Це забезпечує стабільність програмного продукту та полегшує його модернізацію.

Архітектурні шаблони є перевіреними рішеннями для типових задач, які виникають під час розробки програмних систем. Вони надають стандартні підходи до організації компонентів і управління даними, що дозволяє уникати поширених помилок у проєктуванні та підвищує ефективність розробки. Використання шаблонів робить систему більш надійною та полегшує інтеграцію різних компонентів.

Шаблони також допомагають досягти кращої масштабованості системи, зокрема у випадках, коли програма має обробляти великі обсяги даних або витримувати значні навантаження. Вони дозволяють легко додавати нові компоненти або сервери без порушення роботи системи. Окрім цього, архітектура, побудована на основі перевірених шаблонів, підвищує рівень безпеки, оскільки включає чітко визначені механізми контролю доступу та захисту даних.

Компонентний підхід є одним із базових принципів побудови архітектури програмних систем, що полягає в розподілі на незалежні модулі, кожен з яких

виконує окрему функцію та взаємодіє з іншими через чітко визначені інтерфейси (рис. 1.7) [15]. Це забезпечує створення масштабованих та легко підтримуваних систем, де зміни або оновлення одного компонента не впливають на роботу інших частин, знижуючи складність і підвищуючи надійність усієї архітектури.

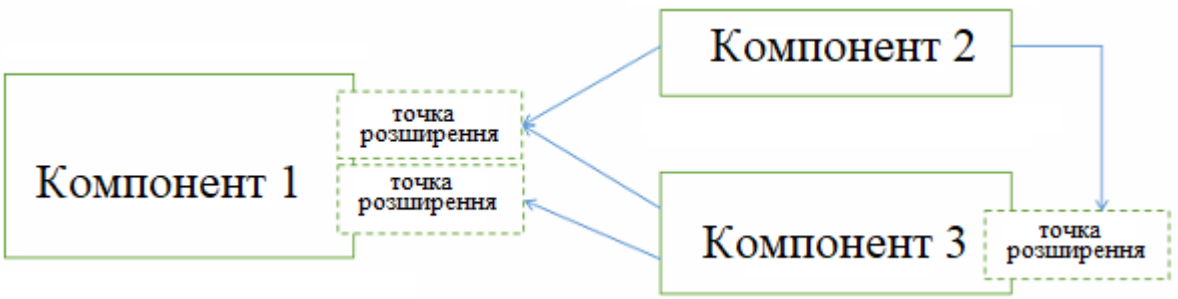

Рис. 1.7. Компонентний підхід

Прикладами реалізації компонентного підходу є наступні архітектури застосунків [16]:

- модульна архітектура – кожен модуль виконує певну функцію та може бути використаний повторно в інших проєктах, що сприяє зниженню витрат на розробку нових систем;
- сервіс-орієнтована архітектура (SOA) – реалізована як набір незалежних сервісів, що взаємодіють через стандартизовані протоколи, такі як HTTP, SOAP, REST, що підвищує гнучкість системи та забезпечує легку інтеграцію нових функцій;
- мікросервісна архітектура – дозволяє створювати незалежні сервіси, які розгортаються та масштабуються окремо, що є важливим для сучасних хмарних систем і підвищує здатність системи швидко адаптуватися до змін.

Кожен із цих підходів реалізує компонентний принцип через розподіл на незалежні модулі чи сервіси, що дозволяє будувати стійкі до змін і гнучкі програмні системи.

Шарова архітектура та архітектура на основі подій є окремими підходами до побудови архітектури, які, хоча й відрізняються від попередньо згаданих компонентних архітектур, також можуть доповнювати їх в різних системах,

залежно від вимог проєкту. Компонентні архітектури, такі як модульна, сервіс-орієнтована та мікросервісна, забезпечують гнучкість та незалежність модулів. Шарова архітектура, з іншого боку, реалізує логічне розмежування функцій (презентаційний шар, шар бізнес-логіки, шар доступу до даних) та може використовуватися як основа для організації сервісів в SOA чи мікросервісах.

Шарова архітектура є одним із класичних підходів до проєктування програмних систем, де функціональність розділена на кілька шарів (рис. 1.8) [17]. Це дозволяє забезпечити чітке розмежування між різними аспектами роботи застосунку, що спрощує процес розробки, тестування та підтримки.

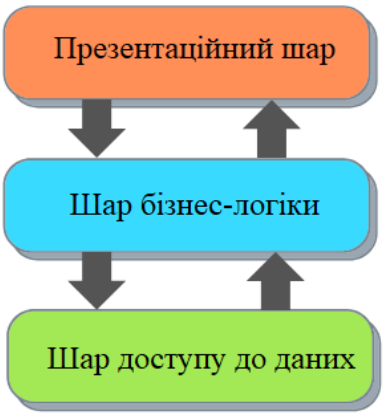

Рис. 1.8. Шарова архітектура

Шарові архітектура програмних систем зазвичай складаються із:

– презентаційного шару – відповідає за інтерфейс користувача та взаємодію з кінцевими користувачами. Сюди входять веб-сторінки, мобільні інтерфейси або будь-який інший фронтенд;
– шар бізнес-логіки – містить бізнес-логіку застосунку, де відбувається обробка даних і виконуються основні функції програми;
– шар доступу до даних – взаємодіє з базами даних або іншими сховищами даних, відповідальний за збереження, вибірку та оновлення даних.

Таке розмежування дозволяє знизити взаємозалежність між частинами системи і спрощує інтеграцію нових функціональних можливостей. Наприклад,

зміни у базі даних або в інтерфейсі користувача не вплинуть на бізнес-логіку програми, якщо вона реалізована належним чином.

У сучасних динамічних системах важливим стає принцип архітектури, орієнтованої на події. Вона широко застосовується у сучасних програмних застосунках, таких як системи інтернету речей (IoT), великі розподілені системи або мікросервіси, де необхідно обробляти події в режимі реального часу [18]. Основна перевага цієї архітектури полягає в тому, що вона дозволяє знижувати залежність компонентів один від одного, що сприяє підвищенню гнучкості та масштабованості системи. На рис. 1.9 представлена схема архітектури на основі подій.

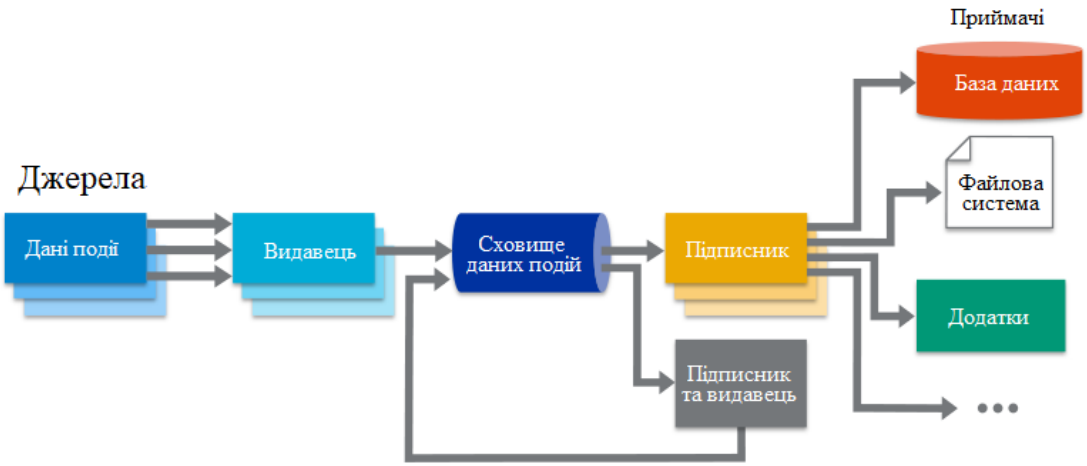

Рис. 1.9. Архітектура, орієнтована на події

Джерела, що генерують події, надсилають ці події до Видавця, який потім передає їх до Сховища даних подій. Це сховище виконує роль буфера, дозволяючи зберігати та обробляти події асинхронно. Потім підписники, які можуть бути різними компонентами системи, такими як база даних, файлова система або додатки, отримують ці події з сховища та обробляють їх відповідно до своїх потреб. Підписник і видавець можуть виконувати функцію як відправника, так і отримувача подій, що забезпечує двосторонній обмін даними між компонентами системи. Події можуть записуватись у базу даних, файлову систему або оброблятись додатками для подальших дій. Така архітектура

дозволяє системам бути більш чуйними та ефективними в умовах великої кількості подій і запитів.

Штучний інтелект відіграє все більшу роль у сучасних програмних застосунках, оскільки він дозволяє програмам адаптуватися до нових умов і виконувати складні обчислювальні завдання, які раніше були недосяжні. Впровадження ШІ в архітектуру програмного забезпечення дозволяє створювати системи, що здатні аналізувати великі обсяги даних, робити прогнози та приймати рішення на основі отриманих результатів. Це відкриває нові можливості для автоматизації процесів та підвищення ефективності.

В архітектурі застосунків із ШІ ключову роль відіграють модулі, які відповідають за обробку даних та навчання моделей. Ці модулі повинні бути високопродуктивними, щоб забезпечувати ефективну роботу з великими обсягами інформації в режимі реального часу. Важливо, щоб такі модулі були інтегровані в загальну архітектуру, враховуючи потребу в постійній обробці великих обсягів інформації, передбачення результатів на основі аналізу та адаптації до нових даних. Це потребує використання відповідних архітектурних шаблонів, таких як мікросервісна архітектура або архітектура, орієнтована на події.

Інтеграція ШІ в архітектуру програмних систем також змінює підхід до масштабованості та безпеки. У зв’язку з цим, важливо забезпечити можливість динамічного розширення ресурсів для підтримки високих обчислювальних навантажень. ШІ-системи потребують більшої потужності для обробки даних і можуть бути вразливими до нових типів атак, таких як маніпуляція навчальними даними або атак на моделі. Архітектура таких систем повинна враховувати ці особливості та включати механізми для захисту як даних, так і моделей, що є ключовим аспектом для забезпечення надійності та безпеки сучасних ШІ-застосунків.

1.4 Дослідження відмінностей вразливостей між застосунками з використанням штучного інтелекту та застосунками без його використання

Застосування ШІ в програмних системах значно впливає на їхню архітектуру, функціонування та підходи до забезпечення безпеки. Зокрема, у ШІ-системах основні вразливості виникають не лише на рівні вихідного коду або інфраструктури, як у звичайних системах, але й у моделях машинного навчання, обробці даних і способах прийняття рішень. Важливо зазначити, що загрози для безпеки у програмах з ШІ є більш динамічними, оскільки залежать від якості даних, налаштувань моделі та її навчання, що створює нові потенційні вразливості, яких не існує у застосунках без ШІ.

Звичайні програмні застосунки мають статичну структуру, де основна увага приділяється логічним помилкам у коді, вразливостям у процесах аутентифікації та обробки даних, а також захисту від загальних кіберзагроз, таких як SQL-ін’єкції, XSS, або DoS-атаки. Однак, у ШІ-застосунках окрім цих загроз, виникають специфічні загрози, пов’язані з обробкою великих обсягів даних, вразливостями моделей та їхньої архітектури. У табл. 1.2 наведено порівняння між основними характеристиками вразливостей у застосунках з використанням ШІ та без нього.

Таблиця 1.2.

Характеристики вразливостей у застосунках з використанням ШІ та без нього

Критерій	Застосунки з ШІ	Застосунки без ШІ
Обробка даних	Використовують великі обсяги даних для навчання моделей, що створює ризики витоків даних або маніпуляцій	Дані обробляються статично на основі чітко визначених правил і алгоритмів
Вразливості при обробці даних	Ризики атак на навчальні дані (Data poisoning), підробка даних для зміни результатів моделі [19]	Загрози, пов’язані з некоректною валідацією вхідних даних
Моделі та алгоритми	Вразливі до атак на модель (Model inversion, Adversarial attacks)	Не використовують адаптивні моделі; основні загрози пов’язані з помилками коду

Закінчення таблиці 1.2.

Критерій	Застосунки з ШІ	Застосунки без ШІ
Прийняття рішень	Прогнози на основі моделей машинного навчання, які можуть бути скомпрометовані або некоректно навчені	Рішення приймаються на основі заздалегідь визначених логічних умов
Вразливості при прийнятті рішень	Можливі атаки на моделі (Model extraction) або маніпуляції результатами прогнозів [20]	Помилки у бізнес-логіці, звичайні атаки
Масштабованість	Необхідна потужна обчислювальна інфраструктура для обробки великих даних і навчання моделей	Масштабування базується на підвищенні потужності серверів для виконання детермінованих алгоритмів
Вимоги до інфраструктури	Використання GPU, високопродуктивних обчислювальних кластерів, хмарних сервісів	Локальні сервери або хмарні рішення без високих вимог до обчислювальних ресурсів
Безпека даних	Ризики компрометації конфіденційних даних через атаки на моделі та обробку даних	Захист даних через класичні методи шифрування та контролю доступу
Вразливості даних	Вразливість до атак на метадані та внутрішні системи через неправильну обробку даних	Типові вразливості: неправильна валідація даних, SQL-ін’єкції
Вразливості моделей	Можливі атаки з маніпуляції навчанням (data poisoning), атаки на витік моделі (Model extraction)	Вразливостей на рівні моделей немає, оскільки відсутня їх адаптивна структура
Адаптивність системи	Система постійно оновлюється через навчання моделі на нових даних, що вимагає додаткових заходів безпеки	Система не адаптується, тому вразливості фіксуються лише у вихідному коді
Безпека моделей	Потребує захисту від атак на моделі (наприклад, вилучення моделей) і захисту результатів прогнозів	Відсутність адаптивних моделей, безпека орієнтована на захист програмного коду

Хоча програми, що використовують ШІ, стикаються з тими ж проблемами безпеки, що й традиційні системи (наприклад, SQL-ін’єкції, атаки відмови в обслуговуванні та інші типи вразливостей), ШІ-застосунки піддаються додатковим загрозам через особливості їхньої структури і функціонування. Зокрема, ризики виникають через обробку великих обсягів даних, навчання моделей та адаптацію поведінки в реальному часі.

Одна з ключових відмінностей у вразливостях ШІ-систем полягає у залежності від навчальних даних. Якщо дані скомпрометовані, це може призвести до того, що модель навчиться на зловмисних або неправильних даних, що спричинить помилки у прогнозах або навіть дозволить зловмисникам керувати результатами моделі. Традиційні системи також можуть бути вразливими до атак, пов’язаних із введенням некоректних даних, але для ШІ-систем ця проблема стає критичною, оскільки моделі постійно навчаються та оновлюються на основі нових даних.

Окрім цього, система з ШІ може бути піддана атакам на рівні моделей або під час їхнього використання. Наприклад, маніпуляції з параметрами моделей можуть призвести до некоректної поведінки, що може бути використано для обходу захисних механізмів. У звичайних системах таких загроз немає, оскільки моделі там не використовуються.

Вразливості ШІ-застосунків у реальному часі також викликають додаткові складнощі. У системах, що працюють з великими потоками даних і приймають рішення в реальному часі, критично важливо відслідковувати зміни в поведінці моделі та виявляти потенційні загрози або аномалії у прийнятті рішень. Це ставить перед системами ШІ нові вимоги до безпеки, оскільки звичайні підходи до моніторингу часто не здатні вчасно виявити ці загрози.

Таким чином, хоча ШІ-системи мають ті ж основні загрози, що й традиційні застосунки, їхні особливості вимагають розширених підходів до виявлення вразливостей. Спеціалізовані системи для аналізу архітектури програм, що використовують ШІ, мають враховувати всі етапи життєвого циклу моделі – від збору даних і навчання до прийняття рішень і взаємодії з

користувачем у реальному часі. Це забезпечить підвищення рівня безпеки та допоможе захистити системи від нових типів атак, що виникають разом із розвитком штучного інтелекту.

1.5 Висновок до розділу 1

У даному розділі було проведено аналіз вразливостей програмних застосунків та їхньої архітектури, що дозволило виявити ключові загрози для сучасних програмних систем. Розглянуто класичні вразливості, такі як переповнення буфера, SQL-ін’єкції, XSS та інші, які є основними загрозами для будь-яких програмних застосунків. Було визначено, що, попри універсальний характер цих вразливостей, системи, що використовують штучний інтелект, мають додаткові специфічні загрози. Зокрема, обробка великих обсягів даних і навчання моделей підвищують ризик маніпуляцій з даними та параметрами моделі. Такі загрози, як Exec Code або DoS Overflow, є особливо небезпечними, оскільки вони можуть безпосередньо впливати на результати моделі й безпеку всього застосунку.

Розглянуто архітектурні підходи, зокрема компонентний, шаровий та орієнтований на події. Визначено, що правильний вибір архітектури є ключовим для забезпечення надійності, масштабованості та стійкості системи до загроз. Порівняння традиційних програмних застосунків і застосунків із використанням ШІ показало, що останні потребують особливих заходів захисту через їхню адаптивність, динамічність моделей і вразливість до атак під час навчання або експлуатації моделей у реальному часі.

Отримані результати дозволять перейти до дослідження методів та технологій аналізу архітектури програмних застосунків, включаючи застосування машинного навчання та нейронних мереж для підвищення ефективності виявлення вразливостей.

РОЗДІЛ 2
МЕТОДИ ТА ТЕХНОЛОГІЇ АНАЛІЗУ АРХІТЕКТУРИ ПРОГРАМНИХ ЗАСТОСУНКІВ

2.1 Методи аналізу вразливостей в архітектурі застосунків

В сучасних умовах, коли застосунки дедалі частіше використовують розподілені системи, хмарні сервіси та мікросервісну архітектуру, надзвичайно важливо забезпечити їхню стійкість до атак. Для виявлення та аналізу вразливостей існує декілька методів, що можуть застосовуватися як на етапі проектування архітектури, так і під час її реалізації та експлуатації, серед яких:

- статичний аналіз коду – дозволяє виявляти потенційні вразливості шляхом перевірки коду без його виконання, що є ефективним методом для виявлення помилок, пов’язаних з логікою програми або недоліками безпеки;
- динамічний аналіз – забезпечує аналіз програми під час її виконання, що дозволяє виявляти проблеми, які виникають лише в реальних умовах роботи системи, включаючи неочікувані взаємодії між компонентами;
- моделювання загроз – передбачає ідентифікацію можливих шляхів атак і створення сценаріїв для оцінки ризиків, що дозволяє оцінити безпеку системи ще на етапі проектування;
- машинне навчання – застосовується для виявлення аномалій та потенційних вразливостей через аналіз великих обсягів даних, що дозволяє автоматизувати процес моніторингу та підвищити ефективність виявлення загроз.

Кожен із перелічених методів надає можливість вчасно виявляти та запобігати вразливостям, підвищуючи рівень захищеності програмних застосунків на різних етапах їх життєвого циклу.

2.1.1 Статичний аналіз коду

Статичний аналіз коду – це метод перевірки вихідного коду програмного забезпечення без його виконання. Він дозволяє виявити потенційні вразливості та помилки на ранніх етапах розробки, аналізуючи структуру і логіку програми [21]. Основною перевагою цього методу є можливість виявлення проблем до того, як код буде запущений, що дозволяє розробникам оперативно усувати помилки та забезпечувати вищий рівень безпеки застосунків.

Статичний аналіз коду передбачає автоматичну або напівавтоматичну перевірку вихідного коду програмного забезпечення за допомогою спеціалізованих інструментів або сканерів. Ці інструменти сканують код на предмет відповідності встановленим правилам безпеки та кодування, шукають уразливості та перевіряють дотримання стандартів розробки. Статичний аналіз є ключовим компонентом забезпечення якості програмного забезпечення, оскільки дозволяє виявити проблеми, які можуть не проявитися під час тестування чи під час реальної експлуатації програми.

В архітектурі застосунків статичний аналіз коду відіграє важливу роль у виявленні таких поширених вразливостей, як SQL-ін’єкції, міжсайтові сценарії (XSS), переповнення буфера, неправильне керування пам’яттю, некоректні перевірки доступу тощо [22]. Завдяки тому, що статичний аналіз виконується без запуску програми, він дозволяє охопити всі можливі шляхи виконання коду, включаючи ті, що можуть бути важкими для тестування під час динамічного аналізу.

На рис. 2.1 представлено процес аналізу вразливостей з використанням статистичного методу. Першим кроком є збір даних про відомі вразливості та атаки, що включає інформацію з різних джерел, таких як бази даних CVE, CWE та інші. Після цього зібрані дані об'єднуються та класифікуються, що дозволяє структурувати їх для подальшого аналізу. На основі цих даних здійснюється виявлення закономірностей у виникненні вразливостей, що дозволяє встановити потенційні зв’язки між різними типами атак та компонентами архітектури.

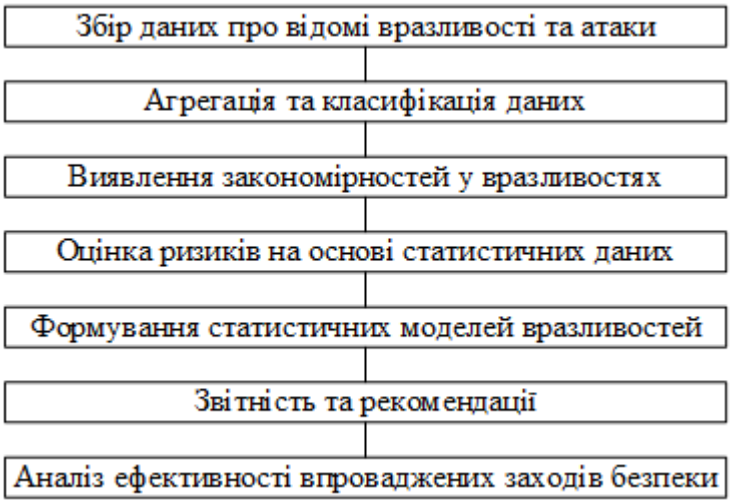

Рис. 2.1. Процес аналізу вразливостей за допомогою статистичного методу

Оцінювання ризиків виконується за допомогою статистичних даних, що допомагає визначити ймовірність виникнення нових вразливостей. Після цього формується статистична модель вразливостей, яка дозволяє зробити прогноз для кожного компонента архітектури. На основі цієї моделі створюється звіт з рекомендаціями щодо заходів для підвищення безпеки системи. Завершується процес аналізом ефективності впроваджених заходів безпеки, що дає змогу оцінити, наскільки дієвими були прийняті рішення та чи потрібні додаткові заходи для покращення захисту.

Існує багато інструментів, які дозволяють виконувати статичний аналіз коду. Відомі серед них: SonarQube, Fortify Static Code Analyzer, Checkmarx, Veracode, OWASP Dependency-Check. Ці інструменти інтегруються в середовища розробки і можуть бути частиною процесу CI/CD, забезпечуючи автоматичну перевірку коду після кожної зміни.

Переваги статистичного методу аналізу вразливостей:

– прогнозування потенційних загроз. Статичний метод дозволяє виявляти закономірності та тенденції у виникненні вразливостей, що допомагає прогнозувати можливі загрози та планувати превентивні заходи;

– аналіз великих обсягів даних. Цей метод здатен працювати з великими обсягами історичних даних про вразливості та атаки, що дозволяє охопити широкий спектр можливих сценаріїв загроз;
– оцінка ризиків на основі реальних даних. Статичний аналіз базується на реальних даних про відомі атаки та вразливості, що підвищує точність оцінки ризиків та надійність прогнозів.

До недоліків статистичного методу аналізу вразливостей відносяться:

– обмежена можливість виявлення нових типів атак. Статичний метод ефективний для аналізу відомих загроз, однак він має обмежену здатність виявляти нові, невідомі вразливості або атаки;
– залежність від якості даних. Ефективність цього методу значною мірою залежить від якості та повноти зібраних даних. Неповні або застарілі дані можуть призвести до неправильних висновків і прогнозів;
– часова затримка. Обробка великих обсягів історичних даних може зайняти значний час, що робить цей метод менш ефективним для швидкого реагування на нові загрози.

Отже, статичний метод аналізу вразливостей є досить потужним інструментом для виявлення закономірностей та оцінки ризиків на основі історичних даних. Він дозволяє систематизувати інформацію про відомі загрози та допомагає прогнозувати можливі атаки. Однак його ефективність обмежується здатністю виявляти нові типи атак і залежить від наявності якісних даних. Тому цей метод варто використовувати як частину комплексної стратегії забезпечення безпеки, поєднуючи його з іншими підходами, зокрема динамічними методами аналізу.

2.1.2 Динамічний аналіз

Динамічний аналіз коду – це метод перевірки програмного забезпечення під час його виконання. На відміну від статичного аналізу, який аналізує вихідний код без його запуску, динамічний аналіз дозволяє досліджувати поведінку

програми в реальних умовах її роботи, що дає можливість виявити вразливості, які проявляються лише під час виконання. Основною метою цього методу є виявлення проблем із безпекою, такими як витоки пам’яті, некоректне управління ресурсами, неконтрольований доступ до даних, та інші проблеми, що можуть бути непомітними під час статичного аналізу.

Динамічний аналіз дозволяє тестувати архітектуру застосунків у реальному часі, що робить його ефективним для виявлення вразливостей, які пов’язані з некоректною роботою компонентів під навантаженням або в різних середовищах [23]. Цей метод широко використовується під час тестування на проникнення, коли імітуються реальні атаки на систему для виявлення вразливих місць у її архітектурі. Динамічний аналіз також дає змогу досліджувати інтеграцію різних компонентів системи та перевіряти, як вони взаємодіють у реальному середовищі [24]. Наприклад, під час виконання може бути виявлено некоректну роботу системи автентифікації або надмірну витрату ресурсів при роботі з базами даних.

Окрім тестування під навантаженням, динамічний аналіз може виявляти помилки в роботі системи, які з’являються через неочікувану поведінку користувача або використання програмного забезпечення в некоректних умовах. Це дозволяє виявити і виправити проблеми до того, як вони будуть експлуатовані зловмисниками в реальних атаках.

Для виконання динамічного аналізу існує низка спеціалізованих інструментів, які дозволяють тестувати застосунки на наявність вразливостей під час їх роботи. Одним з найвідоміших інструментів є OWASP ZAP (Zed Attack Proxy), який надає можливість виконувати тестування на проникнення та аналізувати веб-додатки на наявність вразливостей у реальному часі. Цей інструмент імітує різні типи атак, включаючи SQL-ін’єкції, XSS та інші, щоб виявити потенційні загрози. Інший відомий інструмент – Burp Suite, який використовується для дослідження безпеки веб-додатків. Він дозволяє перевіряти роботу застосунків під різними сценаріями використання, що допомагає виявити вразливості на етапі взаємодії користувача з системою.

На рис. 2.2 представлено основні етапи проведення динамічного аналізу, які включають кілька послідовних стадій.

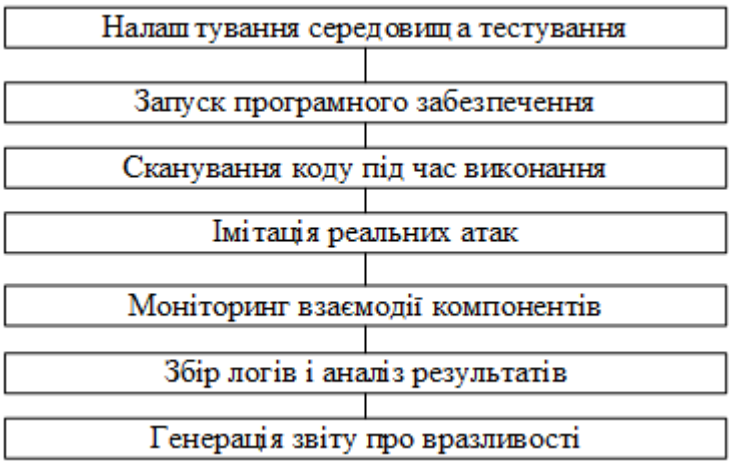

Рис. 2.2. Основні етапи динамічного аналізу для виявлення вразливостей

Початковим етапом динамічного аналізу є налаштування тестового середовища, яке має відповідати реальним умовам функціонування системи. Далі відбувається запуск програмного забезпечення, після чого інструмент починає сканування коду під час його виконання, що дозволяє перевіряти поведінку системи під навантаженням або під час взаємодії з користувачами. Наступний етап включає симуляцію реальних атак для визначення реакції системи на потенційні загрози. Далі відбувається моніторинг взаємодії компонентів, що дає можливість виявити можливі проблеми в комунікації та передачі даних між модулями. Завершальними етапами є збір логів та аналіз результатів, що дає змогу сформувати звіт про виявлені загрози і надати рекомендації щодо їх усунення. Такий підхід забезпечує поглиблене розуміння безпеки системи та дозволяє оперативно вжити заходів для її вдосконалення.

Переваги динамічного аналізу:

– виявлення вразливостей під час виконання. Динамічний аналіз дозволяє виявляти проблеми, які можуть проявлятися тільки під час реального виконання програми, такі як витоки пам’яті, некоректне керування ресурсами або порушення у взаємодії між компонентами;

– імітація реальних атак. Метод дозволяє перевірити, як система реагує на реальні сценарії атак, що робить його ефективним для оцінки стійкості системи до потенційних загроз;
– моніторинг взаємодії компонентів. Динамічний аналіз дає змогу перевірити, як окремі компоненти системи взаємодіють один з одним, що дозволяє виявити проблеми у комунікаціях між ними.

Недоліки динамічного аналізу:

– часові затрати. Процес динамічного аналізу може бути тривалим, оскільки система повинна бути виконана та протестована в різних умовах і сценаріях, що потребує значного часу;
– обмеження на виявлення прихованих вразливостей. Динамічний аналіз може пропустити деякі вразливості, які не проявляються під час виконання програми або не активуються в конкретних сценаріях;
– ресурсоємність. Цей метод вимагає багато ресурсів, оскільки тестування в реальному часі потребує реального середовища виконання, що може потребувати значних обчислювальних потужностей та інфраструктури.

Динамічний аналіз коду є важливим інструментом для виявлення вразливостей у програмному забезпеченні під час його виконання. Він дозволяє імітувати реальні атаки та моніторити взаємодію компонентів, що робить його ефективним для тестування програмних систем у реальних умовах. Однак цей метод потребує значних ресурсів та часу, а також може не виявити всі можливі загрози. Попри це, динамічний аналіз залишається важливою частиною процесу забезпечення безпеки програмного забезпечення, особливо у поєднанні з іншими методами.

2.1.3 Моделювання загроз

Моделювання загроз – це метод, який використовується для систематичного виявлення потенційних загроз та вразливостей на ранніх етапах проектування системи [25]. Моделі загроз допомагають прогнозувати можливі

сценарії атак, аналізувати шляхи зловмисного впливу на систему та надавати рекомендації щодо захисту. Основна мета цього методу полягає в тому, щоб виявити вразливі місця в архітектурі програмного забезпечення до його розробки або під час її планування. Це дозволяє знизити ризики та захистити систему від потенційних атак, ще до її реалізації та експлуатації.

Метод аналізу архітектури за допомогою моделей загроз дозволяє оцінити кожен компонент системи з точки зору безпеки. Під час аналізу визначаються потенційні загрози, які можуть виникнути в результаті роботи системи або під час взаємодії з іншими компонентами. Це включає вивчення сценаріїв атак, таких як підміна ідентифікаційних даних, порушення конфіденційності, маніпуляція даними, або атаки на відмову в обслуговуванні.

Один із найпоширеніших підходів до моделювання загроз – це метод STRIDE, який використовується для класифікації загроз за категоріями [26]:

- S – Spoofing (підміна особи);
- T – Tampering (підробка даних);
- R – Repudiation (відмова від відповідальності);
- I – Information Disclosure (розкриття інформації);
- D – Denial of Service (відмова в обслуговуванні);
- E – Elevation of Privilege (ескалація привілеїв).

Кожна загроза з цього переліку розглядається окремо для кожного компоненту системи. Таким чином, за допомогою моделі загроз можна визначити найбільш вразливі місця системи та відповідно спланувати заходи для їх захисту.

Для аналізу архітектури системи за допомогою моделей загроз існують спеціалізовані інструменти, які автоматизують цей процес. Одним з найвідоміших інструментів є Microsoft Threat Modeling Tool, який допомагає будувати моделі загроз для систем, що використовують різні архітектурні підходи. Цей інструмент дозволяє розробникам систематично аналізувати архітектуру, визначати загрози і автоматично генерувати звіти з рекомендаціями щодо заходів безпеки.

Інші інструменти для моделювання загроз включають OWASP Threat Dragon та IriusRisk. OWASP Threat Dragon є відкритим інструментом для моделювання загроз, який дозволяє створювати діаграми системи та визначати можливі сценарії атак. IriusRisk, у свою чергу, автоматизує процес моделювання загроз та інтегрується з іншими інструментами забезпечення безпеки, такими як сканери вразливостей.

Моделювання загроз дозволяє ідентифікувати потенційні вразливості на ранніх етапах розробки, що сприяє своєчасному впровадженню заходів захисту. На рис. 2.3 представлено основні етапи процесу моделювання загроз, який використовується для систематичного аналізу архітектури з метою виявлення вразливостей.

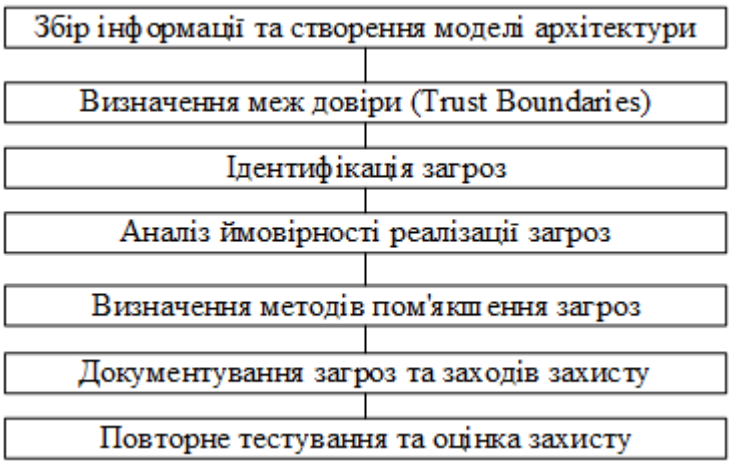

Рис. 2.3. Етапи аналізу вразливостей за допомогою моделювання загроз

Перший етап охоплює збір інформації та створення моделі архітектури, що забезпечує чітке розуміння всіх компонентів системи та їхньої взаємодії. Далі визначаються межі довіри між компонентами, що дозволяє ідентифікувати області з потенційними загрозами. Наступний крок – це виявлення загроз для кожного компонента, що допомагає зрозуміти, які типи атак можуть бути спрямовані на різні ділянки архітектури. Після цього здійснюється оцінка ймовірності реалізації кожної загрози, що дає змогу визначити рівень ризику.

На основі отриманих даних розробляються методи пом'якшення загроз, що можуть включати технічні або організаційні заходи для зниження ризиків. Потім

здійснюється документування загроз та відповідних заходів захисту, що створює основу для подальшого вдосконалення безпеки системи. Завершальний етап передбачає повторне тестування та оцінку захисту, що дозволяє перевірити ефективність впроваджених заходів і забезпечити стійкість системи до можливих атак.

Переваги методу моделювання загроз:

– раннє виявлення загроз. Моделювання загроз дозволяє ідентифікувати потенційні вразливості ще на етапі проектування архітектури, що знижує витрати на усунення проблем у майбутньому;
– систематичний підхід. Використання структурованих методів, таких як STRIDE, забезпечує чітке розуміння можливих загроз і допомагає організувати процес безпеки;
– підвищення рівня безпеки. Завдяки моделюванню загроз розробники можуть вжити запобіжних заходів для захисту системи від можливих атак до її впровадження.

Серед недоліків можна виділити:

– велика залежність від точності початкової інформації. Якщо архітектура системи не повністю документована або модель має неточності, це може призвести до пропущення важливих загроз;
– затрати часу і ресурсів. Моделювання загроз вимагає значних зусиль для аналізу кожного компонента системи, що може уповільнити процес розробки;
– не всі загрози можуть бути виявлені. Метод базується на відомих загрозах і сценаріях атак, тому нові або складні загрози можуть бути пропущені.

Моделювання загроз є потужним методом для аналізу безпеки архітектури програмного забезпечення на ранніх етапах проектування. Воно дозволяє ідентифікувати вразливі компоненти та вжити заходів для їх захисту до реального впровадження системи. Проте цей метод має свої обмеження, зокрема потребу в точних даних і значні витрати часу, але загалом сприяє підвищенню рівня безпеки та захисту системи від атак.

2.1.4 Машинне навчання

Машинне навчання (МН) – це підгалузь штучного інтелекту, яка дає змогу комп'ютерам самостійно навчатися, використовуючи дані, та приймати рішення або прогнози без необхідності їхнього явного програмування. Замість того, щоб заздалегідь передбачати всі можливі сценарії, моделі машинного навчання аналізують велику кількість даних, виявляючи патерни та аномалії [27]. В контексті аналізу вразливостей в архітектурі програмних систем, МН використовується для автоматичного виявлення відхилень у роботі системи, які можуть вказувати на потенційні загрози. Наприклад, машинне навчання здатне аналізувати історичні дані про атаки, їхні характеристики та поведінку, щоб передбачати появу нових загроз, що схожі на попередні випадки. Завдяки цьому системи на основі МН можуть постійно вдосконалюватися, покращуючи свою здатність виявляти аномалії та реагувати на нові загрози. Система може виявляти незвичні дії користувачів або компонентів, які відрізняються від стандартних моделей поведінки, і сигналізувати про потенційну атаку, запобігаючи таким чином можливим збоям або витокам даних. Такий підхід дозволяє знижувати кількість хибних позитивних сповіщень і покращувати загальний рівень безпеки системи.

Для проведення аналізу вразливостей із використанням машинного навчання існує кілька спеціалізованих інструментів, які здатні обробляти великі обсяги даних і виявляти аномалії, що можуть сигналізувати про потенційні загрози. Одним із таких інструментів є Darktrace, який використовує алгоритми машинного навчання для аналізу мережевого трафіку та поведінки користувачів у реальному часі [28]. Цей інструмент аналізує величезні обсяги даних і здатен виявляти незвичні дії в мережі, які можуть вказувати на загрози безпеці, такі як спроби проникнення, зловмисні програми або внутрішні атаки. Darktrace адаптується до середовища, в якому працює, і навчається на нових даних, що дає змогу виявляти загрози навіть тоді, коли вони відрізняються від уже відомих сценаріїв атак.

Інший приклад – Cylance, який застосовує штучний інтелект і машинне навчання для передбачення загроз і запобігання атакам. Cylance працює на основі аналізу поведінкових патернів, вивчаючи дії користувачів та компоненти системи для виявлення відхилень від норми, що можуть свідчити про наявність загрози [29]. Інструмент також використовує історичні дані про атаки для покращення точності передбачень і вчасного реагування на нові загрози.

Метод МН дозволяє ефективно аналізувати великі обсяги даних та виявляти приховані аномалії, що можуть свідчити про потенційні загрози. На рис. 2.4 представлено основні етапи застосування машинного навчання для аналізу вразливостей в архітектурі системи.

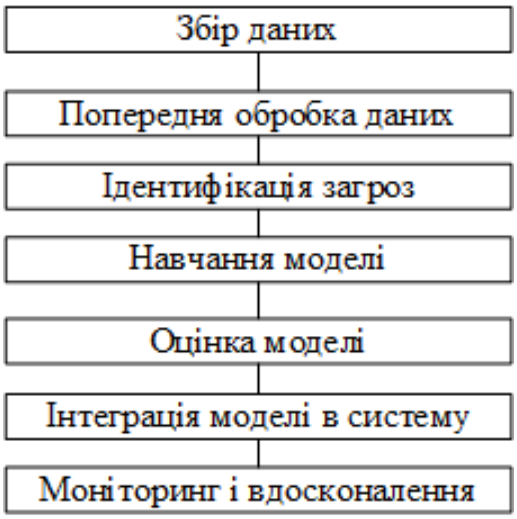

Рис. 2.4. Етапи аналізу вразливостей за допомогою машинного навчання

Процес розпочинається зі збору даних, що включає великий масив інформації про попередні атаки, поведінкові моделі системи та її користувачів. Наступним етапом є попередня обробка даних, яка передбачає очищення, нормалізацію та структурування даних для подальшого аналізу. Далі здійснюється ідентифікація загроз на основі аналізу зібраної інформації, що дозволяє виявити основні загрози та ризики.

Наступний крок – це навчання моделі, де алгоритм машинного навчання використовує підготовлені дані для виявлення аномалій або прогнозування нових загроз. Після цього виконується оцінка моделі на тестових даних для перевірки

її точності та надійності. Після досягнення бажаних результатів модель інтегрується в систему для постійного моніторингу та виявлення загроз у реальному часі. Завершальний етап передбачає постійний моніторинг і вдосконалення моделі, яка оновлюється на основі нових даних та змін у системі, що забезпечує її актуальність і ефективність у виявленні потенційних загроз.

Переваги методу машинного навчання для аналізу загроз складаються із:

– адаптивність. Моделі машинного навчання можуть адаптуватися до нових типів атак, оскільки вони навчаються на нових даних, що дозволяє їм виявляти нові загрози;
– аналіз великих обсягів даних. Машинне навчання дозволяє аналізувати величезні обсяги даних, що неможливо було б зробити вручну або за допомогою традиційних методів;
– автоматизація процесу. Моделі МН можуть працювати в автоматичному режимі, постійно скануючи систему та сигналізуючи про можливі аномалії, що знижує людський фактор у процесі аналізу.

До недоліків машинного навчання можна віднести:

– вимога великих даних. Машинне навчання потребує великих обсягів якісних даних для навчання, і без достатньої кількості даних моделі можуть бути неефективними;
– часове навантаження на навчання. Навчання моделей може бути тривалим і ресурсозатратним процесом, особливо при аналізі великих обсягів інформації.
– можливість хибних позитивних результатів. Моделі можуть виявляти аномалії, які не є реальними загрозами, що призводить до хибних сигналів про небезпеку.

Машинне навчання є потужним інструментом для аналізу вразливостей в архітектурі програмних систем. Воно дозволяє виявляти нові загрози на основі аналізу поведінки користувачів і системних компонентів, що підвищує загальний рівень безпеки. Незважаючи на потребу в значних обсягах даних і ресурсів для навчання, моделі МН є ефективними для автоматизації процесу виявлення загроз

і адаптації до нових атак. Проте важливо враховувати ризик хибних позитивних сигналів і постійно оновлювати моделі для підвищення їхньої точності та надійності.

Отже, методи аналізу вразливостей в архітектурі застосунків дозволяють систематично підходити до виявлення потенційних загроз на різних етапах життєвого циклу програмного забезпечення. Використання статичного та динамічного аналізу, моделювання загроз, автоматизованих інструментів, а також сучасних підходів на основі машинного навчання, дозволяє створювати надійні та безпечні програмні системи. Інтеграція безпеки на всіх етапах розробки є ключовим фактором для захисту сучасних застосунків від потенційних атак.

2.2 Аналіз існуючих рішень та систем для виявлення вразливостей

У сучасних умовах розвитку інформаційних технологій забезпечення безпеки програмних застосунків є однією з найважливіших задач. Зокрема, виявлення вразливостей в архітектурі програмних систем стало критично важливим для зниження ризиків кібератак та мінімізації збитків від можливих порушень. Для цього розроблено велику кількість рішень та систем, що дозволяють автоматизувати процес виявлення вразливостей, аналізувати структури програмного коду, мережеві з’єднання та інші елементи архітектури.

2.2.1. SonarQube

SonarQube – це інструмент для статичного аналізу коду, який призначений для виявлення вразливостей, помилок, дублювання коду та інших проблем якості в програмному забезпеченні (рис. 2.5). Основною метою SonarQube є забезпечення постійного покращення якості коду шляхом виявлення дефектів на ранніх етапах розробки, що дозволяє зменшити кількість вразливостей і помилок у готовому продукті [30].

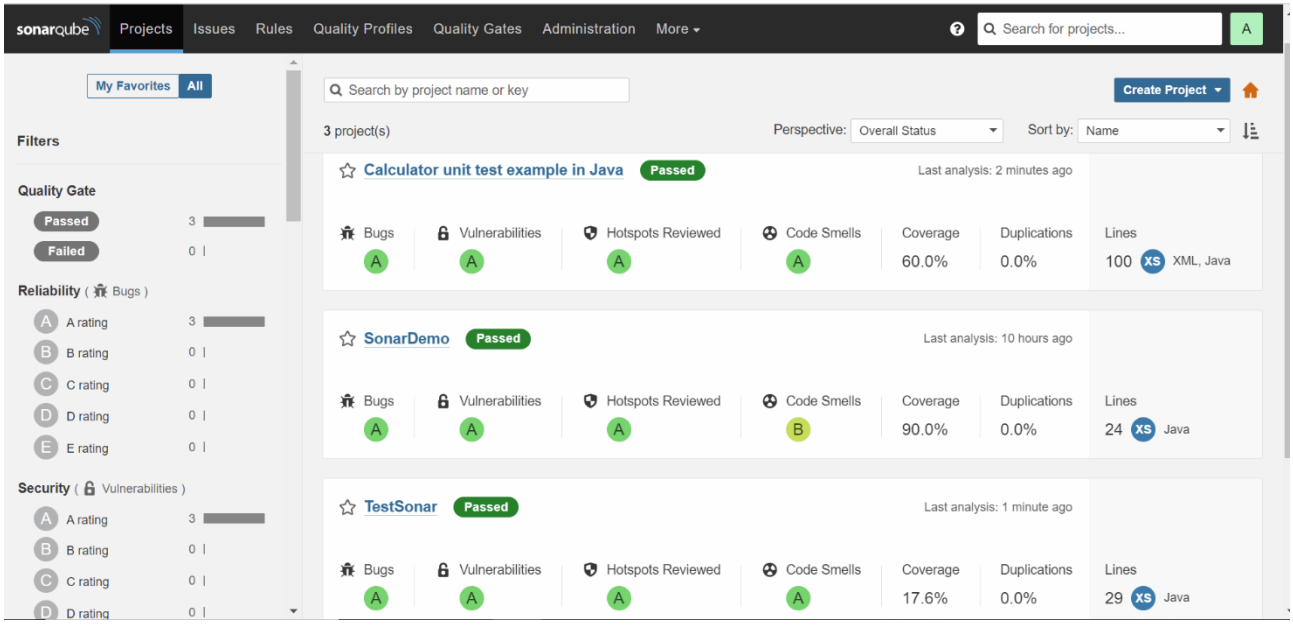

Рис. 2.5. Приклад інтерфейсу системи «SonarQube» [31]

Архітектура SonarQube складається з декількох основних компонентів. Центральним елементом є сервер, що керує процесами аналізу, збором метрик і наданням результатів через веб-інтерфейс. Інструмент також використовує базу даних для зберігання результатів аналізу та показників якості коду. Ще однією важливою частиною є аналізатор, який інтегрується з процесами CI/CD і аналізує код на різних мовах програмування, використовуючи плагіни. SonarQube підтримує інтеграцію з багатьма DevOps інструментами, забезпечуючи зручний механізм контролю якості коду на кожному етапі розробки.

Переваги SonarQube:

– широка підтримка мов програмування [32]. SonarQube підтримує понад 25 мов, що робить його універсальним інструментом для різних проєктів;
– інтеграція з CI/CD. Застосунок легко інтегрується з процесами безперервної інтеграції та розгортання, що дозволяє автоматизувати перевірку якості коду на кожному етапі;
– веб-інтерфейс і звіти. SonarQube надає зручний веб-інтерфейс для перегляду результатів аналізу, з детальними звітами про виявлені вразливості, помилки та дублювання коду;

– постійний моніторинг якості коду. Він забезпечує постійний аналіз і покращення якості коду протягом усього життєвого циклу розробки програмного забезпечення.

Недоліки SonarQube:

– обмежена підтримка для виявлення динамічних вразливостей [33]. SonarQube фокусується на статичному аналізі коду і не покриває динамічні вразливості, що можуть виникати під час виконання програми;
– потреба в налаштуванні. Для досягнення точних результатів необхідно правильно налаштувати правила та плагіни для кожного конкретного проєкту, що може вимагати додаткових зусиль;
– високі системні вимоги. Для великих проєктів або постійного аналізу великих кодових баз SonarQube може вимагати значних апаратних ресурсів;
– чутливість до якості правил. Точність аналізу сильно залежить від правил і плагінів, тому їх неправильне налаштування може призвести до помилкових попереджень або пропуску важливих вразливостей;
– кількість встановлених плагінів. Система підтримує понад 50 плагінів, які додають можливості інтеграції, розширення функціоналу та налаштування аналізу під потреби конкретного проєкту;
– частота виявлених вразливостей: За даними SonarQube, середній рівень виявлення критичних вразливостей на початковому етапі аналізу досягає 80%.

SonarQube є потужним інструментом для статичного аналізу коду, який забезпечує високу якість програмного забезпечення та допомагає виявляти вразливості на ранніх етапах розробки. Він відрізняється широкою підтримкою мов програмування та інтеграцією з процесами CI/CD. Однак, через фокус на статичному аналізі та потребу в налаштуванні, його слід використовувати в комбінації з іншими інструментами для забезпечення комплексного аналізу безпеки.

2.2.2. Fortify Static Code Analyzer

Fortify Static Code Analyzer (SCA) – це потужний інструмент для статичного аналізу коду, що використовується для виявлення вразливостей у програмному забезпеченні на етапі розробки (рис. 2.6). Його основне призначення – автоматизоване виявлення можливих проблем безпеки в коді, таких як SQL-ін'єкції, міжсайтові сценарії (XSS), неправильне управління пам'яттю та інші типи загроз [34]. Fortify SCA дозволяє розробникам і командам безпеки ідентифікувати та усувати вразливості ще до того, як програмне забезпечення буде розгорнуте у виробництво.

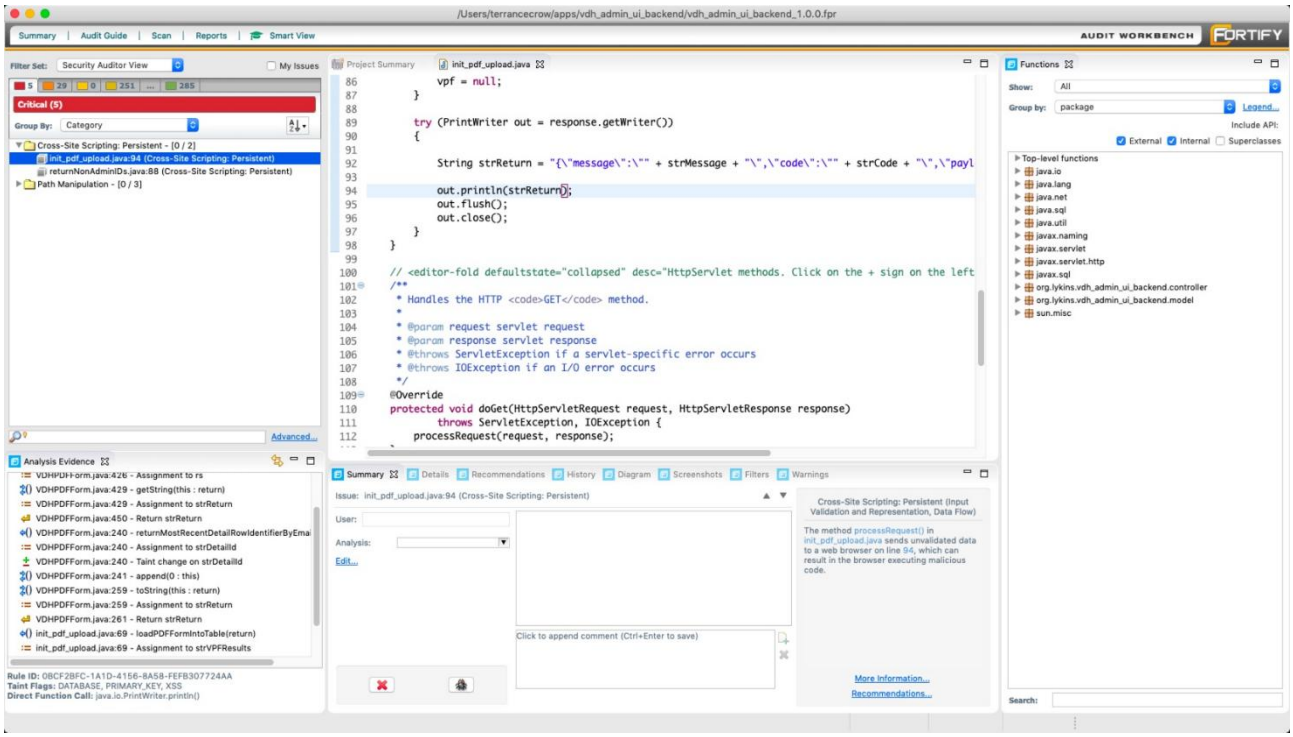

Рис. 2.6. Приклад інтерфейсу системи «Fortify SCA» [35]

Архітектура Fortify SCA складається з кількох основних компонентів. Основний аналізатор здійснює глибокий статичний аналіз вихідного коду, шукаючи вразливості за допомогою великої бази правил безпеки. Результати аналізу передаються у вигляді звітів, які містять детальні описи виявлених проблем, рекомендації щодо їх усунення та ступінь критичності. Fortify SCA

інтегрується з різними системами управління вихідним кодом і може бути легко включений у CI/CD-процеси для автоматичного аналізу коду після кожного коміту або розгортання. Він підтримує широкий спектр мов програмування та платформ, що робить його універсальним інструментом для великих і різноманітних проєктів.

Переваги Fortify SCA:

- глибокий аналіз безпеки. Fortify SCA надає можливість глибокого аналізу коду на наявність вразливостей, використовуючи широку базу правил безпеки, що охоплює різні типи загроз;
- рекомендації щодо усунення вразливостей [36]. Інструмент надає детальні рекомендації щодо виправлення виявлених вразливостей, що допомагає розробникам швидко реагувати на проблеми безпеки;
- час на аналіз коду. Fortify SCA може аналізувати проєкти середнього розміру (до 500 тисяч рядків коду) протягом 1-3 годин. Для великих проєктів час аналізу може збільшитися до 12-24 годин, залежно від обсягу та складності;
- частота виявлених вразливостей. За статистикою, Fortify SCA виявляє приблизно 65% критичних вразливостей під час першого аналізу.

Недоліки Fortify SCA:

- висока вартість ліцензії [37]. Fortify SCA є комерційним продуктом із високою вартістю ліцензії, що може бути проблемою для малих і середніх компаній;
- вимоги до ресурсів. Для аналізу великих проєктів Fortify SCA потребує значних апаратних ресурсів, що може уповільнювати процес розробки;
- час обробки. Глибокий аналіз коду може займати значний час, особливо в масштабних проєктах із великою кодовою базою;
- складність налаштування. Для досягнення точних результатів інструмент потребує детального налаштування правил та конфігурації під конкретні вимоги проєкту.

Fortify SCA є потужним інструментом для виявлення вразливостей у програмному забезпеченні, забезпечуючи глибокий аналіз безпеки коду на етапі розробки. Завдяки широкій підтримці мов програмування та інтеграції з CI/CD-процесами, він дозволяє постійно моніторити якість та безпеку програмних продуктів. Однак висока вартість, потреба в ресурсах та складність налаштування можуть стати викликами для компаній із обмеженими можливостями. Fortify SCA є важливим інструментом для великих проєктів з високими вимогами до безпеки, особливо в умовах корпоративного середовища.

2.2.3. Veracode

Veracode – це хмарна платформа для забезпечення безпеки програмного забезпечення, яка пропонує широкий спектр інструментів для аналізу коду на вразливості (рис. 2.7). Її головне призначення – допомагати розробникам і командам безпеки виявляти та усувати вразливості на різних етапах розробки, включно зі статичним і динамічним аналізом коду, а також аналізом компонентів з відкритим кодом [38]. Veracode дозволяє автоматизувати процеси безпеки, інтегрувати їх у життєвий цикл розробки програмного забезпечення та забезпечувати безперервний моніторинг безпеки.

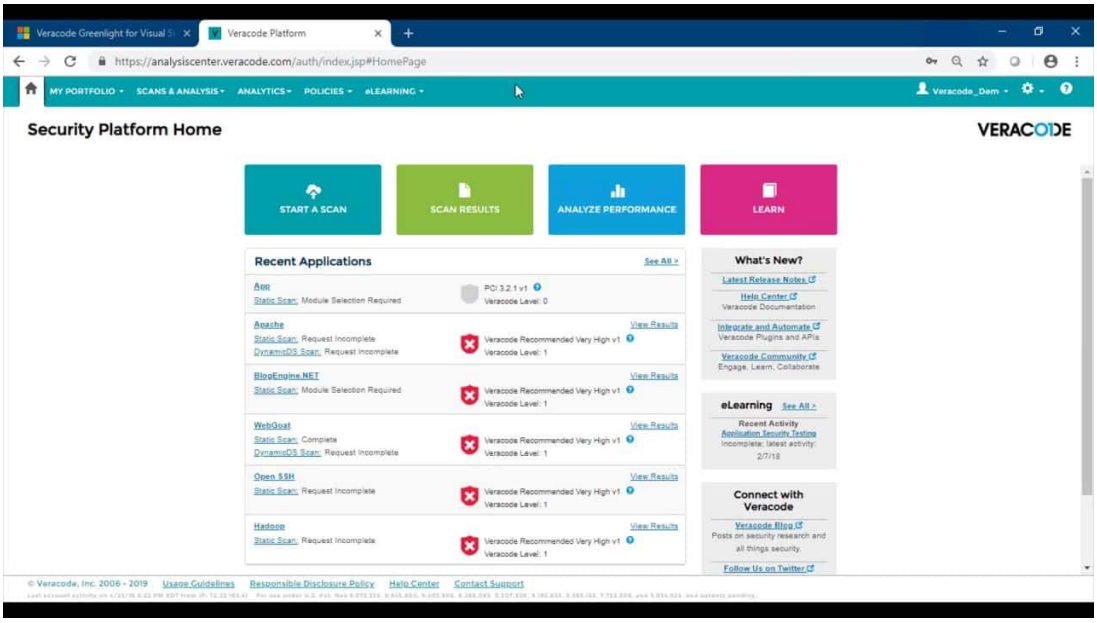

Рис. 2.7. Приклад інтерфейсу системи «Veracode» [39]

Архітектура Veracode базується на хмарній інфраструктурі, що дозволяє виконувати аналізи без необхідності розгортати складне обладнання на стороні користувача. Основний компонент – це хмарний сервер, де відбувається аналіз коду, а результати доступні через веб-інтерфейс. Veracode інтегрується з CI/CD-інструментами, забезпечуючи автоматизацію перевірок безпеки під час кожного оновлення коду. Платформа також пропонує API для інтеграції з іншими системами розробки, що робить її гнучкою та зручною для використання в різних середовищах.

Переваги Veracode:

- хмарна інфраструктура. Не потребує локального розгортання, що дозволяє швидко почати аналіз коду без витрат на налаштування інфраструктури;
- широкий спектр аналізу [40]. Платформа підтримує статичний, динамічний аналіз коду та аналіз компонентів з відкритим кодом, забезпечуючи всебічний підхід до безпеки;
- гнучкість у використанні. Завдяки API та готовим інтеграціям Veracode може бути налаштована для різних середовищ і робочих процесів, що підвищує її зручність для команд розробників;
- час на аналіз коду. Veracode зазвичай проводить статичний аналіз коду за кілька годин для середніх проєктів обсягом до 500 тисяч рядків коду. Для великих проєктів час може збільшитися до 24 годин;
- широкий спектр підтримуваних мов. Платформа підтримує понад 25 різних мов програмування та фреймворків, серед яких Java, .NET, JavaScript, Python, Ruby, C/C++, Swift та інші.

Недоліки Veracode:

- висока вартість. Veracode є комерційним продуктом із високою ціною, що може бути недосяжним для малих компаній;
- обмежений доступ до локальних ресурсів [41]. Оскільки платформа працює в хмарі, доступ до конфіденційних локальних ресурсів може бути складним або потребувати додаткових налаштувань;

– час аналізу. Деякі види аналізу, особливо для великих проєктів, можуть займати багато часу через віддалену обробку;
– залежність від інтернет-з’єднання. Оскільки платформа хмарна, стабільність роботи залежить від якості та надійності інтернет-з’єднання.

Отже Veracode – це потужний хмарний інструмент для забезпечення безпеки програмного забезпечення, який пропонує комплексний підхід до виявлення вразливостей через статичний, динамічний аналіз і аналіз компонентів з відкритим кодом. Основними перевагами є зручність хмарної інфраструктури, можливість інтеграції з CI/CD та гнучкість у використанні. Однак висока вартість, залежність від інтернету та обмежений доступ до локальних ресурсів можуть бути суттєвими обмеженнями для деяких користувачів

Для порівняння ключових характеристик та можливостей різних інструментів аналізу безпеки коду було розглянуто три популярні системи: SonarQube, Fortify Static Code Analyzer (SCA), Veracode, а також розроблене програмне забезпечення. У табл. 2.1 представлено порівняння цих систем за основними характеристиками, що дозволяє оцінити їхні можливості та обмеження для ефективного аналізу безпеки коду.

Таблиця 2.1.

Порівняння характеристик наявних інструментів аналізу безпеки коду

Характеристика	SonarQube	Fortify SCA	Veracode	Розроблене ПЗ
Підтримка мов програмування	Понад 30 мов	Понад 25 мов	Понад 25 мов	Можливість додавання специфічних мов
Інтеграція з CI/CD	Є інтеграція з популярними системами	Підтримка інтеграції з CI/CD	Підтримка CI/CD	Можливість інтеграції через API
Тип аналізу	Статичний аналіз коду	Статичний аналіз	Статичний та динамічний, аналіз	Статичний аналіз із елементами ML

Закінчення таблиці 2.1

Характеристика	SonarQube	Fortify SCA	Veracode	Розроблене ПЗ
Оновлення баз вразливостей	Щомісячне оновлення правил	Щотижневе оновлення баз	Щоденне оновлення баз	Можливість регулярного оновлення вручну
Можливість локального розгортання	Локальне та хмарне	Локальне розгортання	Хмарна платформа	Локальне розгортання
Якість розпізнавання помилок	80%	65%	70%	68%
Частота оновлення бази правил	Щомісячне оновлення	Щотижневе оновлення	Щоденне оновлення	Налаштовується вручну

Основними недоліками порівняних інструментів є:

- обмежена здатність до виявлення нових вразливостей: багато інструментів орієнтовані на застарілі методи аналізу, які спираються на статичні правила та шаблони, що ускладнює виявлення нових загроз у системах, які постійно змінюються;
- незадовільна інтеграція з сучасними методами машинного навчання: наявні рішення не завжди дозволяють використовувати мультикласову класифікацію або адаптивні алгоритми, що знижує їхню ефективність у динамічних умовах;
- складність налаштування та адаптації: більшість комерційних інструментів не дозволяють легко додавати або змінювати правила аналізу для специфічних мов програмування, що обмежує їхнє використання для проєктів з різними технологіями;
- відсутність можливості локального розгортання та перенавчання моделей: багато систем не підтримують локальне розгортання або постійне

оновлення моделей, що є критично важливим у високозахищених середовищах, де постійні зміни в моделях безпеки є обов'язковими.

Отже, перелічені недоліки обґрунтовують необхідність розробки нового програмного забезпечення, яке б долало ці обмеження та підвищувало ефективність і гнучкість процесу виявлення загроз.

2.5 Висновок до розділу 2

У рамках даного розділу було проведено детальний аналіз методів та технологій для аналізу архітектури програмних застосунків, що дозволило визначити найбільш ефективні підходи для виявлення вразливостей. Розглянуто та порівняно кілька основних методів, зокрема статичний та динамічний аналіз коду, моделювання загроз і машинне навчання, що дало змогу окреслити їхні можливості та обмеження у контексті виявлення вразливостей в архітектурі застосунків.

Також було проведено аналіз існуючих рішень для виявлення вразливостей, таких як SonarQube, Fortify Static Code Analyzer і Veracode, що дозволило оцінити їхній функціонал, архітектуру, а також переваги й недоліки. На основі цього аналізу проведено порівняння цих рішень із розробленим програмним забезпеченням, що дозволило визначити конкурентні переваги створеної системи. Серед таких переваг – висока якість розпізнавання нових вразливостей завдяки використанню машинного навчання, гнучкість у налаштуванні та можливість локального розгортання з подальшим перенавчанням моделей. Ці результати є важливими для наступного розділу, в якому буде детально розглянуто проєктування та розробку системи, зокрема інтеграцію проаналізованих технологій і методів у її архітектуру.

РОЗДІЛ 3
ВИБІР МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ СИСТЕМИ АНАЛІЗУ АРХІТЕКТУРИ ПРОГРАМНИХ ЗАСТОСУНКІВ З МЕТОЮ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ПОВ'ЯЗАНИХ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Аналіз методів машинного навчання для виявлення вразливостей в архітектурі програмних застосунків

При розробці системи виявлення вразливостей важливим етапом є вибір методу машинного навчання, який забезпечить високу точність та адаптивність системи до нових загроз. Для обґрунтованого вибору методу необхідно проаналізувати різні підходи, враховуючи їхні можливості у виявленні патернів та аномалій у великих обсягах даних, які можуть свідчити про потенційні вразливості. У цьому розділі детально розглянуто та проаналізовано методи, які можуть бути застосовані для ефективного виявлення загроз і прогнозування їхнього впливу на систему.

Машинне навчання дозволяє системам самостійно навчатися на великих обсягах даних, виявляти приховані патерни та аномалії. До розгляду методів машинного навчання включено метод обмеженої пам'яті Бройдена – Флетчера – Голдфарба – Шанно із максимальним ентропійним підходом, метод стохастичної дуальної координаційної оптимізації та наївний баєсівський класифікатор. Кожен із цих методів має свої унікальні властивості та підходить для різних сценаріїв аналізу даних і прогнозування загроз.

Метод Бройдена – Флетчера – Голдфарба – Шанно з максимальним ентропійним підходом (Limited-memory Broyden–Fletcher–Goldfarb–Shanno, L-BFGS) є одним із найпотужніших і популярних квазіньютонівських методів для оптимізації великих моделей машинного навчання, зокрема в задачах класифікації та регресії [42]. Він поєднує принципи класичного методу BFGS,

що базується на обчисленні похідних, з підходом максимізації ентропії, який дозволяє моделювати імовірнісні процеси для досягнення оптимальних рішень.

L-BFGS використовує обмежену кількість інформації про попередні ітерації для мінімізації пам’яті, необхідної для збереження матриці Гессе (другої похідної). Це робить його ідеальним для роботи з великими наборами даних та моделями з великою кількістю параметрів, де збереження повної інформації про похідні неможливе через обмеження ресурсів.

В архітектурі програмних застосунків L-BFGS із максимальним ентропійним підходом може бути використаний для оптимізації моделей машинного навчання, які вирішують задачі класифікації, прогнозування та виявлення загроз. Основне завдання полягає в тому, щоб навчити модель на основі історичних даних про атаки або вразливості, оптимізуючи параметри моделі для покращення її здатності передбачати нові загрози.

Максимальний ентропійний підхід, який інтегрований у L-BFGS, дозволяє враховувати ймовірності різних результатів, намагаючись знайти модель, яка є найбільш «невпередженою» щодо можливих варіантів розвитку подій. Це особливо важливо в задачах виявлення вразливостей, де потрібно враховувати різні типи загроз і їхню ймовірність на основі неповних або неточних даних. Максимізація ентропії дозволяє моделі бути більш гнучкою та чутливою до нових або рідкісних типів загроз, які можуть не бути представлені у великій кількості в навчальних даних.

На рис. 3.1 представлено етапи даного алгоритму для навчання моделі. Процес починається з ініціалізації параметрів, де задаються початкові значення для функції оптимізації. Після цього проводиться обчислення градієнтів, яке вказує на напрямок найбільшого зменшення функції втрат. Алгоритм зберігає обмежену кількість попередніх градієнтів і змін параметрів для економії пам’яті. Далі виконується оновлення наближення матриці Гессе, що допомагає визначити напрямок оптимізації.

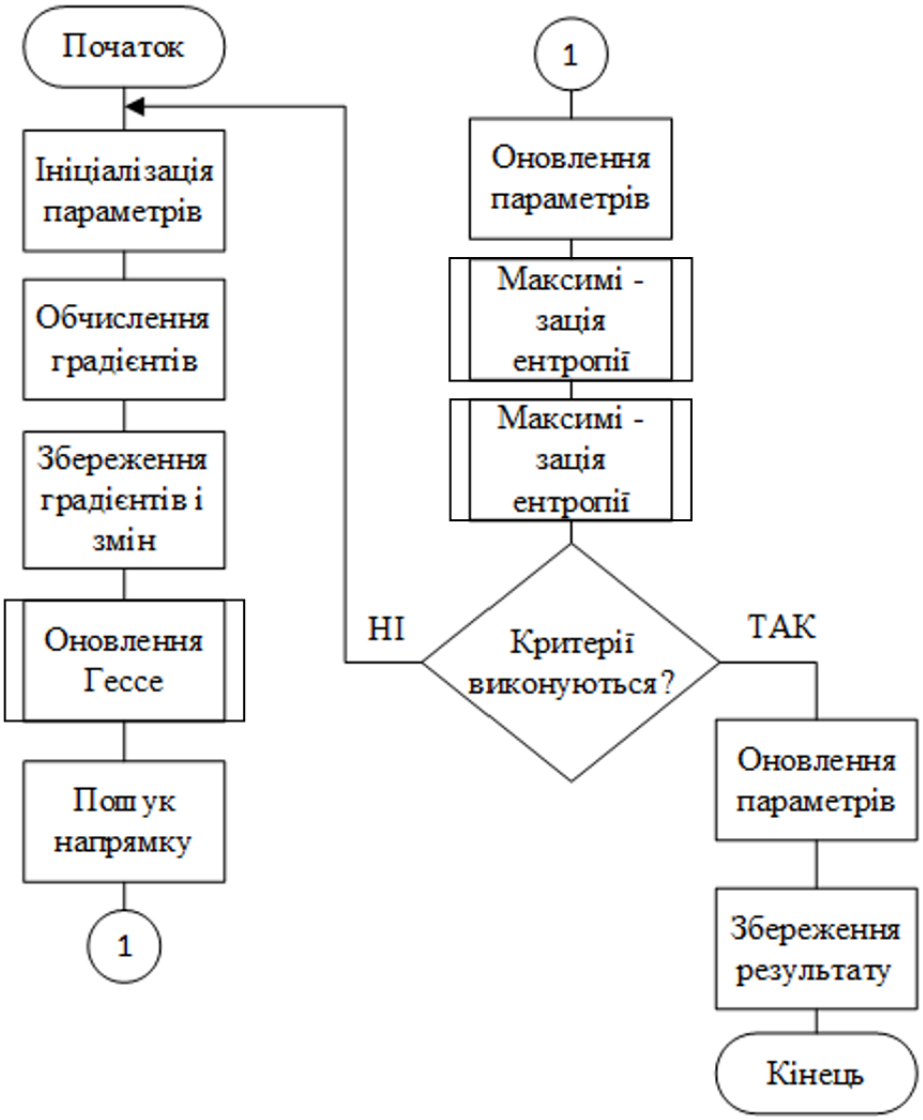

Рис. 3.1. Алгоритм L-BFGS для навчання моделі

На основі цього наближення обчислюється напрямок пошуку, за яким оновлюються параметри моделі, щоб мінімізувати функцію втрат. Після цього відбувається максимізація ентропії, яка дозволяє збільшити адаптивність моделі до нових загроз. На кожній ітерації перевіряються критерії зупинки: якщо ці критерії не виконуються, алгоритм продовжує цикл із обчислення нових градієнтів. Якщо критерії виконуються, алгоритм завершується, і результатом стає оновлення параметрів моделі та збереження результату.

Цей алгоритм дозволяє ефективно адаптувати моделі для виявлення вразливостей в архітектурі застосунків, підвищуючи точність прогнозування та зменшуючи ризики кібератак.

Переваги методу L-BFGS із максимальним ентропійним підходом:

– економія пам’яті. Метод L-BFGS зберігає лише обмежену кількість попередніх градієнтів і змін параметрів, що робить його ефективним для роботи з великими даними, не перевантажуючи пам’ять системи;
– швидка конвергенція. Метод швидко збігається до оптимальних рішень, що є важливим при роботі зі складними архітектурами та великими наборами даних, які вимагають високої продуктивності;
– гнучкість завдяки максимальному ентропійному підходу. Максимізація ентропії дозволяє адаптувати модель до невизначеності в даних, що покращує здатність виявляти нові або рідкісні загрози.

Недоліки методу L-BFGS із максимальним ентропійним підходом:

– вимога обчислення градієнтів. Метод вимагає обчислення градієнтів на кожній ітерації, що може бути ресурсоємним при роботі з великими наборами даних або складними функціями;
– чутливість до початкових умов. Алгоритм може бути чутливим до вибору початкових параметрів, що впливає на швидкість і точність збіжності;
– не завжди підходить для нерівномірних функцій. Метод L-BFGS найкраще працює з гладкими диференційованими функціями, але може виявитися менш ефективним для оптимізації функцій із великою кількістю нерівномірностей або розривів.

Отже, метод L-BFGS із максимальним ентропійним підходом є ефективним інструментом для оптимізації моделей, які використовуються в аналізі вразливостей архітектури програмних застосунків. Його здатність економити ресурси пам’яті та забезпечувати швидке зближення робить його особливо корисним для великих систем. Проте, для досягнення найкращих результатів важливо враховувати обчислювальні витрати на градієнти та чутливість до початкових умов. Цей метод може стати ефективною складовою комплексних систем безпеки, що потребують адаптації до нових загроз та високої продуктивності.

Метод стохастичної дуальної координаційної оптимізації (Stochastic Dual Coordinate Ascent, SDCA) – це метод оптимізації, який широко застосовується в задачах машинного навчання, зокрема для класифікації та регресії [43]. Цей метод орієнтований на розв'язання задач оптимізації у дуальній формі, що дозволяє значно прискорити процес навчання моделей за рахунок використання стохастичного підходу, коли оптимізація проводиться не над усією вибіркою даних, а над її випадковими підмножинами. Це дає можливість швидше досягати оптимальних параметрів моделей, що особливо важливо при роботі з великими наборами даних.

SDCA оптимізує цільову функцію шляхом чергування між двома координатами (перемінними) у дуальній задачі, кожна з яких представляє точку, що покращується в процесі оптимізації. Метод вирішує дуальну форму оптимізації, що дає можливість виконувати операції з високою обчислювальною ефективністю та паралельністю. Стохастичний компонент методу полягає в тому, що на кожній ітерації вибирається лише підмножина даних, яка використовується для оновлення параметрів моделі, що дозволяє уникнути обробки всього набору даних одразу і, відповідно, знижує часові витрати.

Метод SDCA може бути корисним для побудови моделей МН, призначених для виявлення вразливостей в архітектурі програмних застосунків, особливо в системах з великими обсягами даних і високою частотою подій (наприклад, мережевий трафік або журнали подій). Завдяки стохастичному підходу, SDCA дозволяє проводити швидке навчання моделей на випадкових підмножинах даних, що знижує час обробки і дозволяє моделі швидко адаптуватися до нових загроз або змін у системі.

Для аналізу вразливостей в архітектурі застосунків SDCA може використовуватися для оптимізації класифікаторів, що виявляють аномалії або загрози на основі поведінкових даних. Наприклад, цей метод може бути використаний для навчання моделей, які визначають можливі атаки на систему на основі аналізу потоку мережевих даних або логів безпеки. Оскільки SDCA

дозволяє швидко оновлювати параметри моделей, він підходить для задач, де важливі швидкість і адаптивність до змінних умов.

На рис. 3.2 представлено алгоритм цього методу для тренування моделей машинного навчання.

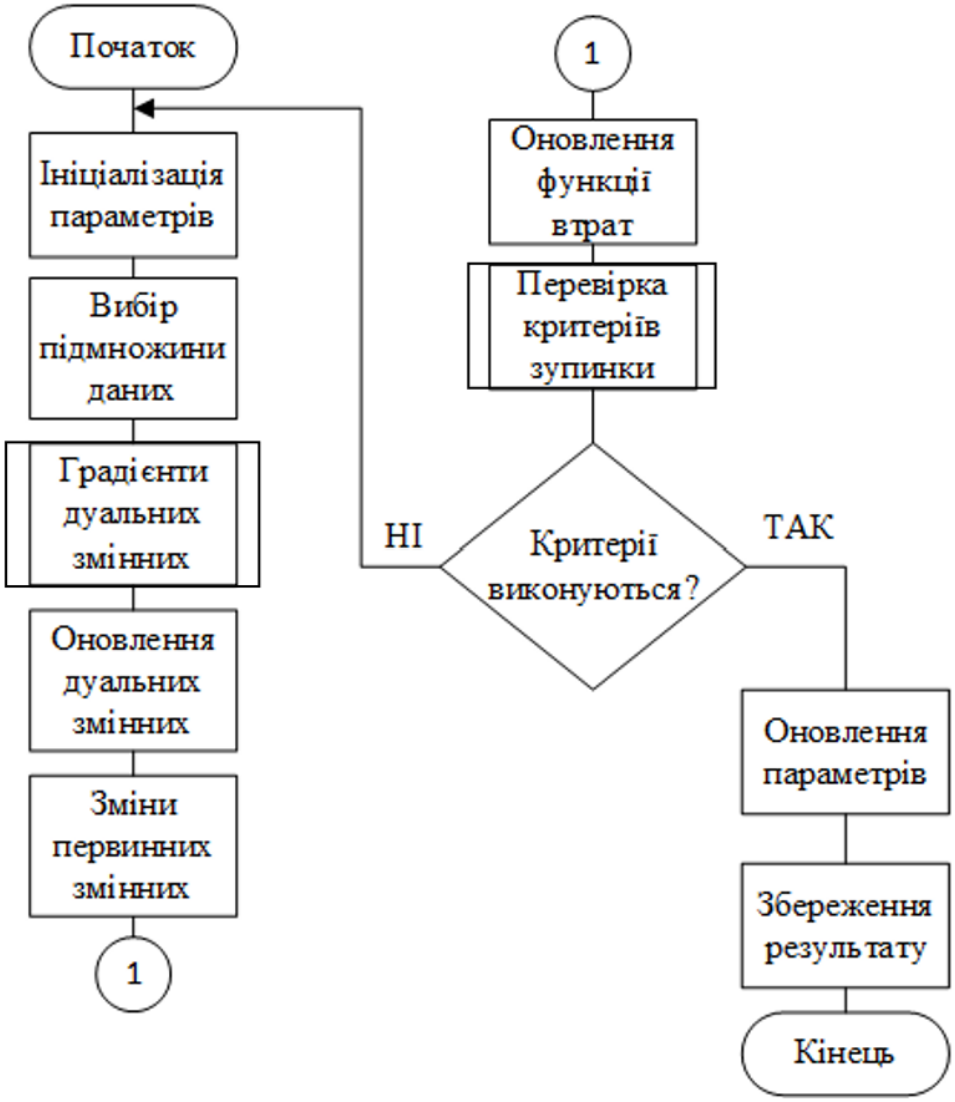

Рис. 3.2. Навчання моделі із допомогою алгоритму SDCA

Навчання моделі розпочинається з ініціалізації параметрів. Далі відбувається вибір випадкової підмножини даних з основного набору даних. Це дозволяє алгоритму працювати швидше, оптимізуючи модель на невеликих підмножинах даних. На основі цієї вибірки обчислюються градієнти для дуальної змінної, що вказує на напрямок оптимізації. Після цього відбувається оновлення дуальних змінних, яке покращує параметри моделі для більш точної

відповідності даним. Потім система обчислює зміни у первинних змінних, що дозволяє коригувати параметри моделі на основі нових даних. Актуалізація функції втрат відображає прогрес оптимізації та оцінює, наскільки добре модель здатна аналізувати дані.

На кожній ітерації перевіряються критерії зупинки: якщо зміни у функції втрат незначні або інші критерії виконані, оптимізація завершується, і модель переходить до етапу оновлення параметрів і збереження результатів. Якщо критерії не виконані, алгоритм повертається до вибору нової підмножини даних і повторює ітерації. Метод SDCA дозволяє тренувати моделі, здатні виявляти вразливості, швидко адаптуючи їх до нових даних і забезпечуючи їхню ефективність у системах захисту програмних архітектур.

Переваги даного методу складаються із:

– швидкість обчислень. Завдяки тому, що метод використовує підмножини даних для кожної ітерації, він дозволяє значно прискорити процес оптимізації порівняно з традиційними методами, що працюють на всьому наборі даних;
– ефективність для великих даних. Добре підходить для роботи з великими наборами даних, оскільки не вимагає обробки всієї вибірки на кожній ітерації. Це дозволяє моделі ефективно працювати навіть у великих системах з високим навантаженням, таких як архітектури програмних застосунків;
– можливість паралельного обчислення. Метод можна легко паралелізувати, що дозволяє підвищити ефективність у сучасних обчислювальних системах з багатоядерними процесорами. Це ще більше знижує час навчання моделі.

Недоліки методу становлять:

– залежність від початкових умов. Як і багато інших оптимізаційних методів, SDCA може бути чутливим до початкових значень параметрів, що впливає на швидкість і точність збіжності;

– не завжди підходить для всіх типів задач. Метод найкраще працює для задач, які можуть бути виражені у дуальній формі, що обмежує його застосування в деяких інших типах задач машинного навчання або оптимізації;
– можливість застрягнути в локальних мінімумах. Оскільки SDCA працює на випадкових підмножинах даних, існує ризик, що алгоритм зупиниться на локальних мінімумів функції, не досягаючи глобального оптимуму.

Отже, метод стохастичної дуальної координаційної оптимізації є потужним інструментом для швидкого та ефективного навчання моделей машинного навчання, особливо при роботі з великими наборами даних. Його основними перевагами є швидкість обчислень, ефективність для великих даних і можливість паралельного виконання. Проте варто враховувати, що SDCA має обмежене застосування для задач, які не можуть бути виражені в дуальній формі, і чутливий до початкових умов та ризиків застрягання в локальних мінімумах. В цілому, SDCA підходить для побудови моделей, які можуть виявляти вразливості в архітектурах програмних застосунків, забезпечуючи високу продуктивність і швидке оновлення параметрів моделі.

Наївний баєсівський класифікатор (Naive Bayes Classifier, NBC) – це простий і ефективний метод машинного навчання, який базується на застосуванні теореми Байєса для класифікації даних [44]. Його називають «наївним», тому що він робить припущення про незалежність ознак, тобто вважає, що всі ознаки (входи) є незалежними одна від одної, що на практиці рідко буває, але це припущення значно спрощує обчислення і дозволяє методові працювати швидко і з невеликими обчислювальними витратами.

Метод NBC належить до класу ймовірнісних методів, що оцінюють ймовірність того, що певний об’єкт належить до певного класу на основі його характеристик. Основна ідея методу полягає у використанні теореми Байєса для обчислення ймовірності того, що новий екземпляр належить до одного з класів, на основі ознак, які він має. Для цього метод використовує попередні ймовірності класів (апріорні ймовірності) та умовні ймовірності ознак для кожного класу.

Для виявлення вразливостей в архітектурі програмних застосунків, NBC може бути використаний для побудови моделей, які класифікують дані на основі відомих характеристик вразливостей. Він може використовуватися для аналізу журналів подій, мережевого трафіку або інших даних, що стосуються безпеки, для класифікації подій на «загрози» та «безпечні дії» на основі їх характеристик.

На рис. 3.3 показано процес навчання моделей на основі наївного баєсівського класифікатора, який застосовується для аналізу даних про вразливості.

Рис. 3.3. Навчання моделі з допомогою наївного баєсівського класифікатора

Процес навчання починається з ініціалізації параметрів, де задаються початкові значення для ймовірностей класів і ознак. Далі обчислюються апріорні

ймовірності для кожного класу на основі навчальних даних, що дозволяє моделі визначити загальні пропорції загроз та безпечних подій у системі. Після цього проводиться розрахунок умовних ймовірностей ознак для кожного класу, що дозволяє оцінити, з якою ймовірністю певна характеристика (наприклад, тип трафіку або поведінка користувача) зустрічається серед загроз або нормальної активності.

На кожній ітерації алгоритм перевіряє, чи достатньо розділені класи за ймовірностями. Якщо різниця між ймовірностями для кількох класів є незначною, виконується етап уточнення вибору, де проводиться додатковий аналіз для кращої класифікації об’єкта. Якщо класи чітко розділені, алгоритм переходить до вибору класу з найбільшою ймовірністю, після чого модель зберігає результат для подальшого використання в реальних умовах.

Такий підхід дозволяє ефективно і швидко навчати моделі для виявлення вразливостей, що робить їх придатними для використання в системах кібербезпеки для аналізу загроз в архітектурі програмних застосунків.

Переваги методу NBC:

– швидкість і простота. Завдяки наївному припущенню про незалежність ознак, метод є дуже швидким і може обробляти великі обсяги даних з невеликими обчислювальними витратами;
– ефективність для текстової класифікації. Метод широко застосовується для задач класифікації текстів, аналізу спаму, детекції зловмисних дій в системах безпеки;
– мінімальні вимоги до обчислювальних ресурсів. Через простоту обчислень наївний баєсівський класифікатор може працювати ефективно навіть у системах з обмеженими ресурсами, що робить його придатним для використання в реальних часі.

Недоліки NBC:

– припущення про незалежність ознак. Основний недолік методу полягає в тому, що він припускає незалежність ознак, що рідко буває в реальних

даних. Це може призводити до помилкових прогнозів, якщо між ознаками є сильна кореляція;

– чутливість до якості даних. Як і всі ймовірнісні методи, наївний баєсівський класифікатор чутливий до якісних даних. Якщо в навчальних даних багато шуму або помилок, це може вплинути на якість прогнозів;
– не завжди найточніший. Попри простоту та швидкість, цей метод не завжди є найбільш точним порівняно з іншими, складнішими методами машинного навчання, такими як дерева рішень або методи глибокого навчання.

Отже, метод наївного баєсівського класифікатора є потужним інструментом для класифікації та виявлення загроз в архітектурі програмних застосунків завдяки своїй простоті та швидкості. Він ідеально підходить для швидкого аналізу великих обсягів даних та попереднього виявлення загроз, але може потребувати додаткового налаштування або доповнення іншими методами для підвищення точності в складних системах безпеки.

3.2 Обґрунтування вибору методу обмеженої пам’яті Бройдена – Флетчера – Голдфарба – Шанно з максимальним ентропійним підходом

Для порівняння методів машинного навчання, застосованих для виявлення вразливостей в архітектурі програмних застосунків, було розроблено консольний застосунок, який виконує процес навчання та оцінки моделей з використанням різних популярних підходів (Додаток А). Завдяки цьому інструменту була проведена експериментальна перевірка кожного методу, включно з методом обмеженої пам’яті Бройдена – Флетчера – Голдфарба – Шанно з максимальним ентропійним підходом, методом стохастичної дуальної координаційної оптимізації та наївним баєсівським класифікатором.

Експерименти базувалися на попередньо зібраному наборі даних про вразливості, що включав інформацію про типи вразливостей, дати їх публікації та оновлення, а також інші важливі параметри, такі як доступність, складність,

конфіденційність, цілісність і доступність (CIA) для кожної вразливості. Після обробки та нормалізації даних кожна модель була піддана процесу крос-валідації для оцінки її продуктивності за показниками MicroAccuracy та MacroAccuracy. Додатково моделі порівнювалися за такими критеріями, як час навчання та здатність до адаптації до нових загроз.

Отримані експериментальні дані, представлені у табл. 3.1, дають змогу порівняти продуктивність методів машинного навчання, обраних для виявлення вразливостей.

Таблиця 3.1.

Порівняльний аналіз методів

Характеристика	L-BFGS	SDCA	NBC
Обчислювальна складність	Середня	Низька	Низька
Швидкість збіжності	Висока	Середня	Висока
Обробка великих наборів даних	Високоефективна	Ефективна	Середня
Точність моделі	Висока	Середня	Низька
Потреба в пам’яті	Середня	Низька	Низька
Здатність до паралельно обробки	Обмежена	Висока	Середня
Чутливість до якості даних	Середня	Низька	Висока
Складність реалізації	Висока	Середня	Низька
Кількість записів тренувального набору	11550	11550	11550
MicroAccuracy	0,68	0,62	0,46
MacroAccuracy	0,68	0,62	0,46
Час тренування (секунд)	2,42	59,98	13,85

Метод L-BFGS є оптимальним вибором для побудови моделі для виявлення вразливостей в архітектурі програмних застосунків на основі машинного навчання, завдяки його високій точності (MicroAccuracy та MacroAccuracy на рівні 0,68) та швидкій збіжності. Порівняння з іншими методами, як-от SDCA та наївний баєсівський класифікатор (NBC), демонструє, що L-BFGS перевершує їх у ключових аспектах, особливо в роботі з великими наборами даних і точності моделі. Метод забезпечує високу ефективність у випадках, коли є багатовимірні

дані з багатьма взаємозв’язками, що часто трапляється при аналізі складних програмних архітектур.

На відміну від SDCA, який має середню швидкість збіжності та нижчу точність, L-BFGS демонструє значно вищу швидкість обробки даних і точніше оптимізує параметри моделі. Хоча SDCA є більш зручним для реалізації паралельних обчислень, обмежена здатність L-BFGS до паралелізації компенсується його здатністю обробляти великі набори даних з високою ефективністю. NBC, з іншого боку, має низьку точність і є чутливим до якості даних, що робить його менш придатним для складних архітектур із багатьма компонентами.

З огляду на ці характеристики, L-BFGS є найбільш підходящим методом для розробки системи виявлення вразливостей, де точність і продуктивність є критично важливими. Його здатність до глибшої оптимізації параметрів і висока ефективність у роботі з великими наборами даних роблять цей метод надійним рішенням для побудови надійних моделей у складних архітектурних структурах.

3.3 Обґрунтування вибору технологій

Вибір технологій для розробки системи аналізу вразливостей є ключовим етапом, що визначає ефективність, продуктивність і можливості майбутнього програмного забезпечення. При прийнятті рішення враховувалися кілька основних факторів: обсяг і тип оброблюваних даних, вимоги до точності та швидкості аналізу, можливість інтеграції з іншими системами, а також масштабованість і гнучкість у налаштуваннях. Окрім того, важливим критерієм було забезпечення ефективної роботи з великими наборами даних та можливість застосування сучасних методів машинного навчання для виявлення складних вразливостей.

3.3.1. Мова програмування C#

Вибір мови програмування для розробки системи є важливим рішенням, яке впливає на продуктивність, зручність підтримки, а також можливість інтеграції з іншими технологіями. При виборі враховуються такі критерії, як доступність бібліотек і фреймворків, швидкість виконання програм, легкість навчання та популярність серед розробників. У даному випадку розглядаються три основні мови програмування: Python, Java та C#, кожна з яких має свої переваги та недоліки залежно від поставлених цілей і вимог до системи.

Python є однією з найпопулярніших мов програмування завдяки своїй простоті та великій кількості бібліотек для роботи з даними та машинного навчання [45]. Вона особливо підходить для швидкої розробки прототипів і аналітичних рішень завдяки високому рівню абстракції. Однак, Python має нижчу продуктивність у порівнянні з компільованими мовами, такими як Java або C#, що може бути критичним для систем з інтенсивними обчисленнями.

Java – це широко поширена мова програмування, відома своєю кросплатформеністю та стабільністю в корпоративних рішеннях [46]. Вона має високу продуктивність завдяки JVM (Java Virtual Machine) та використовується для створення масштабованих систем. Проте, через свою складнішу синтаксичну структуру Java може бути менш гнучкою в розробці прототипів у порівнянні з Python.

C# є потужною мовою, створеною корпорацією Microsoft, яка добре інтегрується з платформою .NET і широко використовується для розробки десктопних і веб-додатків [47]. Мова забезпечує високу продуктивність і багаті можливості для роботи з багатопотоковістю та паралельними обчисленнями. Однак C# здебільшого орієнтована на екосистему Windows, що може обмежити її використання у кросплатформених рішеннях.

Оскільки система розробляється з використанням технологій машинного навчання і потребує високої продуктивності та гнучкості, порівняння цих мов є

важливим для прийняття оптимального рішення. У табл. 3.1 представлені основні характеристики цих мов.

Таблиця 3.2.

Порівняльний аналіз мов програмування

Характеристика	Python	Java	C#
Продуктивність	Середня (інтерпретована мова)	Висока (компільована мова на JVM)	Висока (компільована для .NET)
Простота у використанні	Висока (інтуїтивний синтаксис)	Середня (більш складний синтаксис)	Середня (синтаксис схожий на Java, але з розширеними можливостями)
Швидкість розробки	Висока	Середня	Середня
Підтримка бібліотек для ML	Широка (TensorFlow, PyTorch тощо)	Помірна	Помірна
Підтримка паралельних обчислень	Обмежена	Висока	Висока (ефективна підтримка в .NET)
Інтеграція з Microsoft екосистемою	Обмежена	Обмежена	Висока (тісна інтеграція з Windows і .NET інструментами)
Час компіляції та запуску	Не потребує компіляції	Довший час запуску через JVM	Швидкий запуск та компіляція через JIT у .NET
Використання пам'яті	Середнє	Середнє	Низьке (оптимізоване управління пам'яттю у .NET)

Виходячи з проведеного аналізу, вибір мови програмування C# для розробки системи є обґрунтованим через її високу продуктивність та ефективну підтримку багатопотоковості, що є ключовими факторами для розробки складних

програмних систем. C# забезпечує швидкий час запуску та компіляції завдяки JIT-компіляції в середовищі .NET, що дозволяє підвищити ефективність роботи програми. Крім того, тісна інтеграція з екосистемою Microsoft робить C# ідеальним для роботи в середовищі Windows та використання інших інструментів, таких як Azure або Visual Studio. Завдяки оптимізованому управлінню пам'яттю в .NET, C# дозволяє мінімізувати ресурси, необхідні для виконання, що є перевагою при обробці великих обсягів даних. Враховуючи ці фактори, C# є оптимальним вибором для створення надійної та продуктивної системи аналізу вразливостей.

3.3.2. Бібліотека машинного навчання ML .NET

Оскільки для розробки системи було обрано мову програмування C#, використання бібліотеки ML .NET є доцільним рішенням, оскільки вона пропонує комплексне середовище для створення моделей машинного навчання, не вимагаючи переходу на інші платформи чи мови програмування. ML .NET, розроблена корпорацією Microsoft, інтегрується безпосередньо з екосистемою .NET, що дозволяє розробникам C# працювати у звичному середовищі, використовуючи при цьому найсучасніші алгоритми та підходи до машинного навчання. Це значно полегшує процес розробки, оскільки усі інструменти для збору, обробки даних, створення моделей і впровадження їх у реальні додатки можуть бути реалізовані в межах однієї платформи.

ML .NET надає широкий набір інструментів для роботи з різними типами даних: від табличних до текстових або зображень, що дозволяє будувати моделі класифікації, регресії, кластеризації та інші типи прогнозів. Бібліотека також підтримує популярні алгоритми машинного навчання, такі як лінійна регресія, дерева рішень, метод L-BFGS для мультикласової класифікації та нейронні мережі. Важливо, що ML .NET надає можливість працювати з великими наборами даних, забезпечуючи масштабованість і високу продуктивність, що є критичним для систем, які працюють з аналізом вразливостей.

Окрім цього, бібліотека підтримує легку інтеграцію з інструментами для автоматизації процесів, такими як CI/CD, що дозволяє безперервно вдосконалювати та розгортати моделі. Такий підхід робить ML .NET ідеальним інструментом для створення систем, які вимагають високого рівня точності та гнучкості в аналізі даних, таких як система аналізу вразливостей, що використовує машинне навчання [48].

На рис. 3.4 представлено життєвий цикл машинного навчання у середовищі ML .NET, що ілюструє ключові етапи створення та використання моделей. Ця схема показує, як дані можуть бути перетворені у навчений алгоритм для прогнозування або інших завдань машинного навчання.

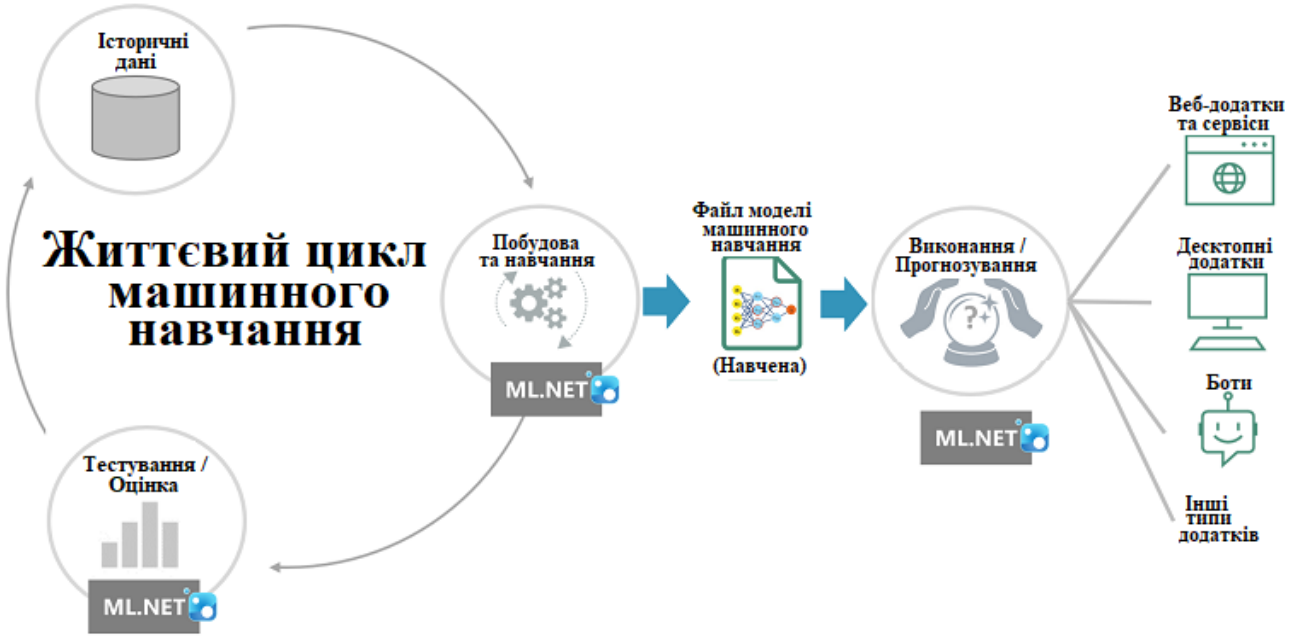

Рис. 3.4. Життєвий цикл машинного навчання у середовищі ML .NET

Перший крок полягає у зборі історичних даних, які будуть використовуватися для тренування моделі. Ці дані надходять із різних джерел і містять приклади, на основі яких модель зможе навчитися прогнозувати або приймати рішення. Далі відбувається побудова та навчання моделі за допомогою ML .NET, де застосовуються різні алгоритми машинного навчання для оптимізації параметрів моделі на основі вхідних даних.

Після цього модель зберігається як навчений файл моделі для подальшого використання. На наступному етапі, під час виконання або прогнозування, модель застосовується до нових даних для отримання прогнозів або рішень у різних типах додатків, таких як веб-сервіси, десктопні програми або навіть боти. Останнім кроком є тестування та оцінка моделі, що допомагає визначити її ефективність і точність на основі нових або тестових наборів даних. Цикл може повторюватися з новими даними для поліпшення моделі.

Після завершення тестування та оцінки моделі важливою частиною процесу є безперервне вдосконалення. З плином часу можуть з’являтися нові типи вразливостей або змінюватися середовище, у якому функціонує система, тому модель потребує регулярного оновлення та перенавчання на основі нових даних. У ML .NET передбачена можливість постійного вдосконалення моделей через додавання нових наборів даних і повторне тренування, що забезпечує високу адаптивність системи до нових викликів безпеки. Це дозволяє системі залишатися актуальною та надійною в умовах постійно змінюваних загроз, що є критично важливим для підтримки високого рівня захисту та ефективного виявлення вразливостей у реальному часі. Таким чином, використання ML .NET забезпечує не тільки початкове навчання моделі, але й її безперервне вдосконалення, що значно підвищує загальну ефективність системи аналізу вразливостей.

3.4 Обґрунтування вибору тренувального набору даних для виявлення вразливостей

Для розробки системи виявлення вразливостей у архітектурі програмних застосунків, зокрема з використанням штучного інтелекту, необхідно обрати відповідний тренувальний набір даних, який дозволить моделі машинного навчання точно навчатися на основі реальних прикладів загроз. Одним із найкращих джерел для таких даних є набір CVE Details [49], що базується на інформації з бази даних CVE. База даних CVE, описана в першому розділі, є

загальновизнаним стандартом у сфері кібербезпеки, який забезпечує централізовану систему класифікації вразливостей і експлойтів, що використовується в усьому світі для аналізу загроз.

Набір CVE Details містить численні атрибути, що описують кожну вразливість, її характеристики та потенційні наслідки для системи. Ці атрибути забезпечують багатий контекст для моделі машинного навчання, дозволяючи їй навчитися розпізнавати складні взаємозв'язки між типами атак, рівнями ризиків та потенційними загрозами для системи.

Для того, щоб зрозуміти, як кожен атрибут може вплинути на процес виявлення вразливостей і побудову моделі, нижче наведено табл. 3.3, яка описує ключові атрибути тренувального набору даних.

Таблиця 3.3.

Атрибути тренувального набору

№	Назва атрибута	Опис	Приклад даних
1	Id	Унікальний ідентифікатор запису	1976
2	CVE_ID	Унікальний ідентифікатор вразливості	CVE-2017-3210
3	CWE_ID	Ідентифікатор категорії вразливості	16
4	VulnerabilityType	Тип вразливості або атаки	Exec Code
5	PublishDate	Дата публікації інформації про вразливість	24.07.2018
6	UpdateDate	Дата останнього оновлення інформації про вразливість	09.10.2019
7	Score	Оцінка серйозності вразливості	07.02.2024
8	Access	Тип доступу для експлуатації вразливості	Local
9	Complexity	Складність використання вразливості	Low
10	Confidentiality	Ступінь впливу на конфіденційність	Complete
11	Integrity	Ступінь впливу на цілісність	Complete
12	Availability	Ступінь впливу на доступність	Complete

Розглянувши ключові атрибути, наведені у табл. 3.3, можна зробити висновок, що кожен з них відіграє важливу роль у процесі побудови ефективної системи виявлення вразливостей за допомогою машинного навчання. Ці атрибути надають необхідний обсяг інформації для навчання моделі, що дозволяє точно аналізувати загрози і прогнозувати можливі атаки. Такий підхід забезпечує гнучкість у налаштуванні алгоритмів та підвищує точність виявлення потенційних вразливостей. Зважаючи на це, нижче представлена аргументація вибору CVE Details як основного тренувального набору даних:

1. Велика кількість даних і різноманітність вразливостей

Набір даних CVE Details містить інформацію про тисячі вразливостей, що охоплюють різні технологічні платформи, архітектури та часові періоди. Це дає можливість моделі машинного навчання, що використовується для виявлення вразливостей, навчитися розрізняти патерни та закономірності між різними типами загроз. Це особливо важливо в умовах застосування штучного інтелекту, де модель має адаптуватися до складних та мінливих характеристик вразливостей у програмних застосунках. Широкий спектр вразливостей також дозволяє моделі прогнозувати нові, раніше не виявлені загрози.

2. Стандартизовані метрики CVSS

Використання метрик CVSS у наборі даних CVE Details дозволяє моделі аналізувати ризики вразливостей з урахуванням загальноприйнятих стандартів у сфері кібербезпеки. Ці метрики не тільки відображають технічні характеристики вразливості, але й оцінюють її вплив на ключові аспекти безпеки, такі як конфіденційність, цілісність та доступність систем. Для моделей машинного навчання, що використовуються у системах виявлення вразливостей із застосуванням штучного інтелекту, така структурована інформація є надзвичайно корисною для формування точних прогнозів щодо потенційних ризиків, що дозволяє моделі розпізнавати не лише вже відомі загрози, але й спрогнозувати нові на основі взаємозв'язків у даних.

3. Актуальність і підтримка бази даних

Оскільки база даних CVE постійно оновлюється, набір даних CVE Details містить найактуальнішу інформацію про сучасні вразливості, що дозволяє моделі машинного навчання залишатися актуальною навіть після початкового навчання. Це важливо для систем із використанням штучного інтелекту, які працюють із динамічними даними, оскільки регулярні оновлення бази забезпечують постійне перенавчання моделі, підтримуючи її точність в умовах швидких змін у кіберзагрозах. Модель може не тільки навчитися на наявних даних, але й адаптуватися до нових типів атак, що підвищує її ефективність у реальних умовах.

4. Гнучкість для різних типів атак та систем

CVE Details охоплює вразливості для різних операційних систем, програмного забезпечення та інфраструктури, що робить його універсальним інструментом для навчання моделі, здатної виявляти широкий спектр загроз. Це особливо важливо для систем, які застосовують штучний інтелект, оскільки така модель має бути здатною адаптуватися до різних архітектур програмних застосунків і типів атак. Гнучкість даного набору даних дозволяє навчити модель розпізнавати та класифікувати вразливості незалежно від конкретних технологічних рішень, що є ключовим у розвитку універсальних систем кібербезпеки.

Отже, вибір даного набору даних CVE Details для тренування системи виявлення вразливостей є оптимальним рішенням, оскільки цей набір забезпечує широкий спектр інформації про вразливості програмних застосунків, зокрема з урахуванням сучасних методів штучного інтелекту. Стандартизовані метрики ризику, актуальність даних та гнучкість набору забезпечують високу точність моделі, що використовує машинне навчання, та її здатність до адаптації в умовах змін кіберзагроз. Таким чином, даний набір даних дозволяє створити ефективну систему аналізу та виявлення вразливостей, здатну працювати з багатовимірними даними та комплексними архітектурами програмних застосунків із застосуванням штучного інтелекту.

3.5 Розробка математичної моделі вирішення задачі

Безпека програмних застосунків стає все більш критичною, особливо з огляду на інтеграцію штучного інтелекту. Для виявлення вразливостей, пов'язаних із ШІ, необхідно розробити математичну модель, яка б дозволила аналізувати та прогнозувати потенційні загрози на основі наявних даних.

Нехай є набір даних $D = \{(x_i, y_i)\}_{i=1}^{N}$, де $x_i \in R^n$ – вектор ознак для *i*-го зразка, а $y_i \in \{1,2, \dots, K\}$ – мітка класу вразливості.

Ознаки включають:

– *CWE_ID* (x_{i1}) – унікальний ідентифікатор типу вразливості;

– *PublishYear* (x_{i2}) – рік публікації вразливості;

– *UpdateYear* (x_{i3}) – рік останнього оновлення;

– *ScoreYear* (x_{i4}) – бальна оцінка рівня загрози;

– *Access* – категоріальна змінна, яка приймає значення «Local» або «Remote»;

– *Complexity* – категоріальна змінна («Low», «Medium», «High»);

– *Confidentiality*, *Integrity*, *Availability*: категоріальні змінні з рівнями «None», «Partial», «Complete».

Категоріальні змінні кодуються методом one-hot encoding. Наприклад, для змінної Access зі значеннями «Local» і «Remote» кодування здійснюється як:

$$Access_i = \begin{cases} [1,0], \text{якщо "Local"} \\ [0,1], \text{якщо "}Remote\text{"} \end{cases} \quad (3.1)$$

Об'єднуючи всі ознаки, отримується повний вектор ознак для *i*-го зразка:

$$x_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}, Access_i, \dots, Availability_i]^T, \quad (3.2)$$

де T означає транспонування.

Для покращення збіжності алгоритму навчання проводиться нормалізація числових ознак за наступним методом:

$$x'_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)}, \quad (3.3)$$

де x_{ij} – значення *j*-ої ознаки для *i*-го зразка.

Далі використовується багатокласова логістична регресія для класифікації типів вразливостей. Ймовірність належності зразка x_i до класу k визначається як:

$$P(y_i = k|x_i) = \frac{\exp(\theta_k^T x_i)}{\sum_{l=1}^{K} \exp(\theta_l^T x_i)}, \quad (3.4)$$

де θ_k – вектор параметрів для класу k.

Після цього потрібно мінімізувати функцію негативної логарифмічної правдоподібності:

$$L(\Theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} \delta(y_i, k) \log P(y_i = k|x_i) + \frac{\lambda}{2} \sum_{k=1}^{K} ||\theta_k||^2, \quad (3.5)$$

де $\delta(y_i, k)$ – функція Кронекера,

λ – коефіцієнт регуляризації, $\Theta=[\theta_1,\theta_2,\ldots,\theta_K]$.

Оптимізація здійснюється за допомогою алгоритму L-BFGS, який ефективно обчислює наближення Гессіана для великих наборів даних.

Дані розділяються на тренувальний та тестовий набори з пропорцією 80% та 20% відповідно:

$$D_{train} \cup D_{test} = D, D_{train} \cap D_{test} = \emptyset, \quad (3.6)$$

Для оцінки якості моделі використовуються метрики *MacroAccuracy* та *MicroAccuracy*. *MicroAccuracy* визначається як:

$$MicroAccuracy = \frac{\sum_{i=1}^{N} \delta(y_i, \hat{y}_i)}{N}, \quad (3.7)$$

Де $\hat{y}_i = \arg max_k P(y_i = k|x_i)$.

MacroAccuracy обчислюється як середнє значення точності по всіх класах:

$$MacroAccuracy = \frac{1}{K} \sum_{i=1}^{K} Accuracy_k, \quad (3.8)$$

де точність для класу k:

$$Accuracy_k = \frac{TP_k}{TP_k + FP_k + FN_k}, \quad (3.9)$$

де: TP_k, FP_k, FN_k – кількість істинно позитивних, хибно позитивних та хибно негативних прогнозів для класу k.

Для нового зразка x_{new} прогнозований клас визначається як:

$$\hat{y}_{new} = \arg max_k P(y_i = k|x_i), \quad (3.10)$$

Загальний час навчання моделі вимірюється як різниця між часом початку t_{start} та завершення t_{end}:

$$t_{train} = t_{end} - t_{start}, \quad (3.11)$$

Представлена математична модель дозволить ефективно аналізувати програмні застосунки для виявлення вразливостей, пов'язаних із використанням ШІ, завдяки комплексному підходу до класифікації загроз. Застосування багатокласової логістичної регресії забезпечує високу точність у розпізнаванні різних типів вразливостей на основі ключових ознак, зокрема ідентифікаторів типів вразливостей, року публікації та рівня загрози (ScoreYear).

Особливо важливу роль відіграє попередня обробка даних: нормалізація числових значень сприяє швидкій збіжності алгоритму під час навчання, а кодування категоріальних змінних методом one-hot encoding підвищує гнучкість моделі в аналізі різнорідних даних (наприклад, рівнів конфіденційності, цілісності та доступності). Оптимізація моделі за допомогою алгоритму L-BFGS дає змогу ефективно обробляти великі набори даних, а також забезпечує стабільність та точність результатів, що підтверджується високими значеннями метрик *MicroAccuracy* та *MacroAccuracy*. Завдяки структурованому підходу до побудови ознак та застосуванню оптимізованих методів, ця модель дозволяє проводити точний і гнучкий аналіз вразливостей у програмних системах із використанням ШІ.

3.6 Побудова алгоритмів для навчання моделей та виявлення вразливостей

Алгоритми для аналізу та обробки даних відіграють важливу роль у сучасних системах, що використовують машинне навчання для виявлення вразливостей у програмних застосунках. Ці алгоритми забезпечують можливість обробляти великі обсяги інформації та приймати рішення на основі аналізу історичних даних. Одним із головних завдань таких систем є не тільки збір і обробка інформації, а й створення моделей, що дозволяють виявляти вразливості в реальному часі та прогнозувати потенційні загрози. Процес побудови моделі

включає кілька важливих етапів, які забезпечують ефективність та точність прогнозів.

На рис. 3.5 представлено алгоритм навчання моделі для виявлення вразливостей, що описує основні етапи обробки даних і тренування моделі.

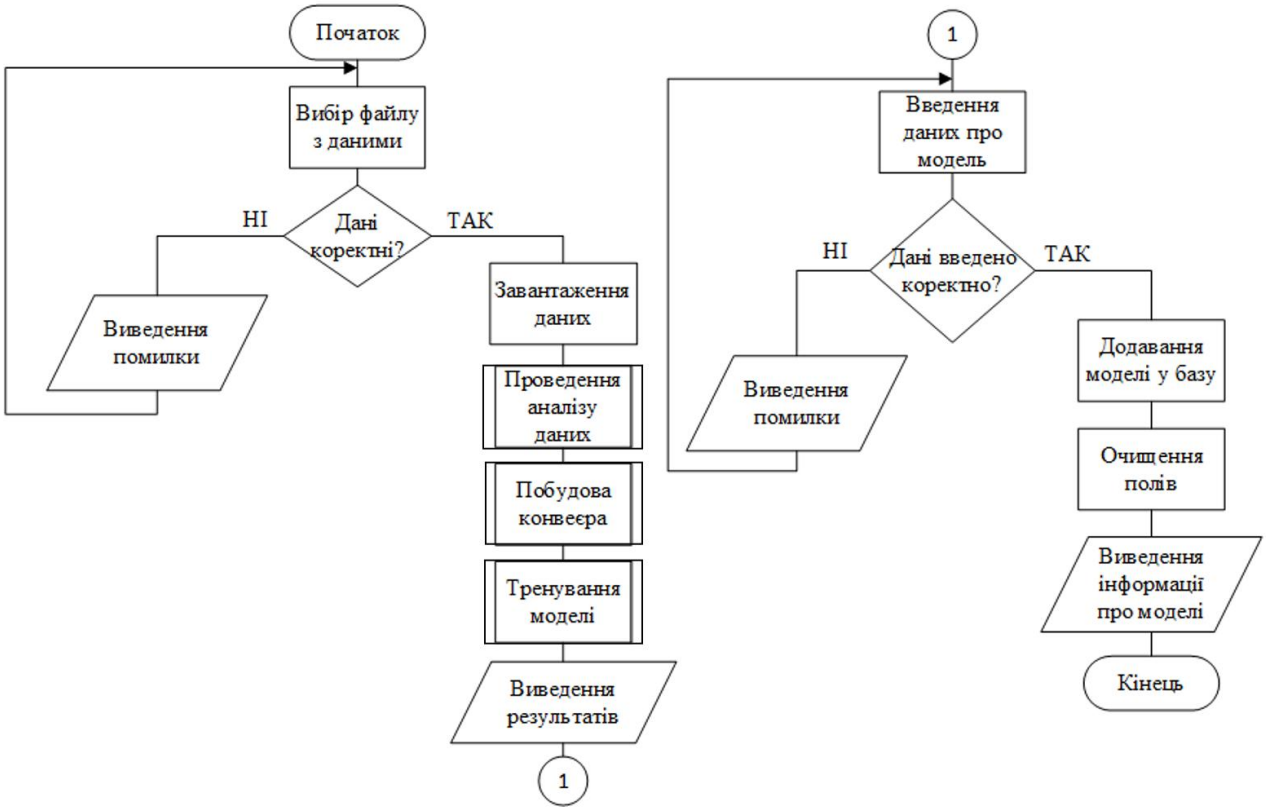

Рис. 3.5. Алгоритм навчання та збереження моделі

Алгоритм починається з вибору файлу з тренувальним набором даних, що містить інформацію про вразливості. Якщо дані в файлі записані коректно, вони завантажуються в список класів для подальшої обробки. Після цього проводиться аналіз даних, результати якого виводяться для перевірки. На основі цього аналізу будується конвеєр для навчання моделі, який включає підготовку даних і вибір відповідного алгоритму для тренування.

Після побудови конвеєра відбувається безпосереднє тренування моделі на тренувальному наборі даних. По завершенню тренування результати тренування виводяться для оцінки точності та продуктивності моделі. Наступним етапом є введення необхідних даних для збереження навченої моделі, таких як ім'я моделі

або додаткові параметри. Якщо введені дані є коректними, інформація про модель додається до бази даних, після чого система очищає всі поля від введених раніше даних, щоб підготувати систему для наступного етапу роботи.

У випадку некоректного введення даних або помилок у записі файлу, система генерує повідомлення про помилку, що дозволяє користувачу внести відповідні зміни.

На рис. 3.6 представлено алгоритм моніторингу роботи моделі для виявлення вразливостей, що описує послідовність дій, які виконує система для аналізу та прогнозування типів вразливостей у програмних застосунках.

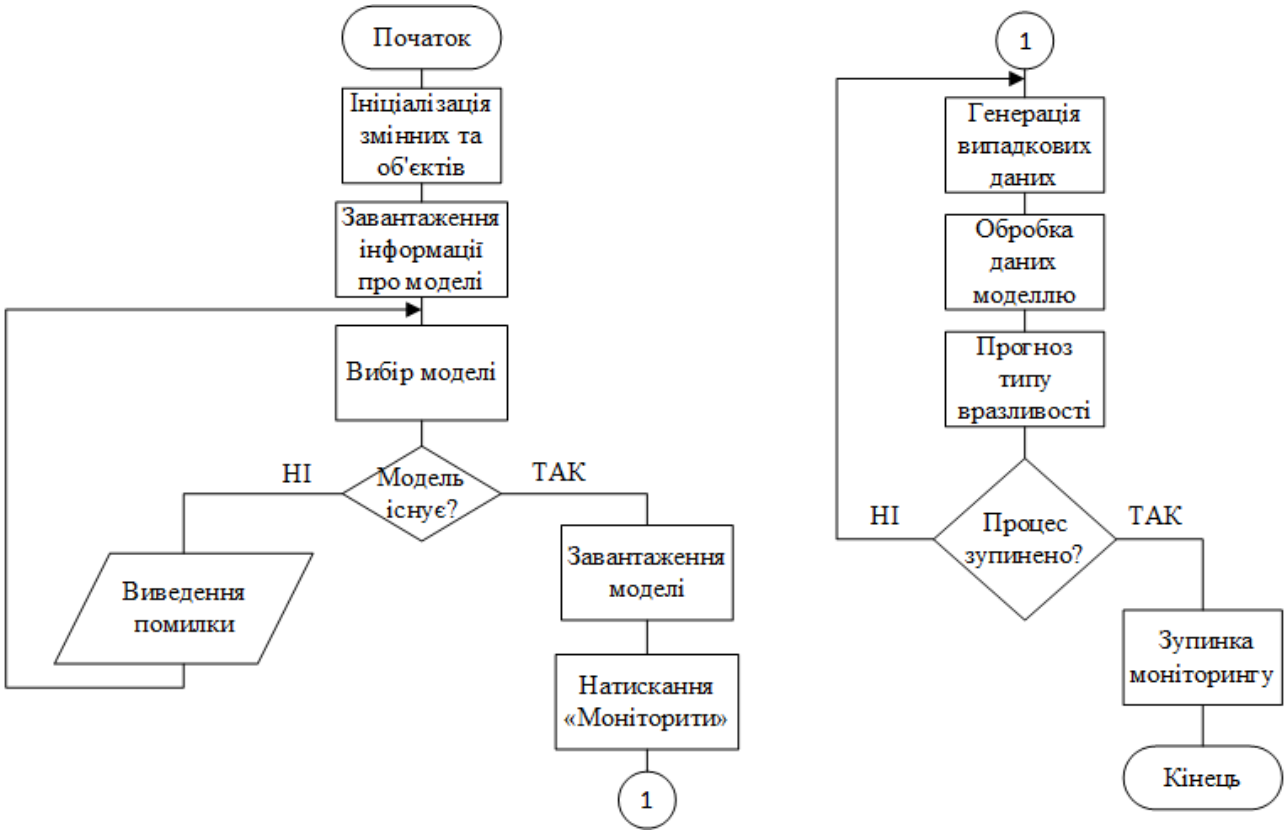

Рис. 3.6. Алгоритм виявлення вразливостей

Алгоритм розпочинається з ініціалізації необхідних змінних та об'єктів, що забезпечує підготовку системи до роботи з моделями. Наступним кроком є завантаження інформації про моделі, які доступні в системі. Користувач повинен обрати необхідну модель для проведення моніторингу, після чого система перевіряє її наявність. Якщо обрана модель не існує, система виводить

повідомлення про помилку, що дає можливість виправити ситуацію. Якщо модель існує, відбувається завантаження всієї інформації про модель, що необхідна для подальшої обробки. Після цього користувач натискає кнопку «Моніторити», яка активує процес моніторингу.

Далі система генерує випадкові дані для вразливості, які будуть використовуватися для оцінки роботи моделі. Ці дані обробляються моделлю, після чого система здійснює прогнозування типу вразливості на основі зібраних і оброблених даних. Результати прогнозування виводяться для подальшого аналізу.

Останній етап алгоритму полягає у перевірці натискання кнопки «Зупинити». Якщо користувач натискає цю кнопку, процес моніторингу зупиняється, і алгоритм завершується. У випадку, якщо кнопка не натиснута, моніторинг триває, і дані продовжують оброблятися системою для оновлення прогнозів вразливостей.

Таким чином, алгоритм забезпечує постійний моніторинг та прогнозування можливих вразливостей з можливістю коригування та зупинки процесу за запитом користувача.

3.7 Проєктування архітектури програмного рішення

Проєктування архітектури програмного забезпечення є одним з найбільш важливих етапів у розробці будь-якої інформаційної системи, оскільки саме на цьому етапі визначаються основні принципи функціонування системи, її гнучкість, масштабованість та продуктивність. Архітектурне рішення впливає на всі аспекти роботи програмного продукту: від взаємодії з користувачем до збереження та обробки даних. У контексті системи виявлення вразливостей, де є необхідність обробляти великі обсяги даних і забезпечувати швидке реагування на нові загрози, вибір відповідної архітектури є надзвичайно важливим.

Одним із найефективніших підходів, який дозволяє забезпечити ці вимоги, є трьохрівнева архітектура [50]. Вона дозволяє чітко розмежувати

відповідальність між різними компонентами системи, що, своєю чергою, підвищує її структурованість, підтримуваність і модульність.

Трьохрівнева архітектура була обрана для розробки системи виявлення вразливостей через її очевидні переваги щодо масштабованості, підтримки та безпеки. Трьохрівневий підхід забезпечує поділ системи на три окремі рівні: рівень презентації (інтерфейс користувача), рівень бізнес-логіки (логіка обробки даних) і рівень зберігання даних (робота з базами даних та іншими сховищами інформації). Такий поділ дозволяє кожному рівню працювати незалежно, що полегшує розробку, тестування і модифікацію системи.

Однією з ключових переваг трьохрівневої архітектури є її масштабованість. Це особливо важливо для систем, які працюють з великими обсягами даних, як-от система для виявлення вразливостей. Масштабованість дозволяє адаптувати систему до зростання кількості користувачів або оброблюваних даних без зниження її продуктивності. Кожен рівень архітектури може бути окремо розширений або вдосконалений, що забезпечує гнучкість системи. Наприклад, при зростанні обсягів даних можна покращити лише рівень зберігання, не торкаючись бізнес-логіки чи інтерфейсу користувача.

Окрім того, трьохрівнева архітектура забезпечує високий рівень безпеки. Оскільки користувач взаємодіє лише з рівнем презентації, він не має прямого доступу до бізнес-логіки чи бази даних. Це дозволяє захистити чутливу інформацію від несанкціонованого доступу і знизити ризик потенційних атак. У контексті системи виявлення вразливостей це є особливо важливим, оскільки обробляються критичні дані, які можуть бути використані для зловмисних цілей.

Рівень доступу до даних відповідає за збереження, оновлення та управління даними користувачів, моделей машинного навчання, результатів прогнозування та подій системи. Цей рівень гарантує, що доступ до даних має тільки відповідний шар бізнес-логіки, що забезпечує захист конфіденційної інформації від несанкціонованого доступу або маніпуляцій. Рис. 3.7 відображає діаграму класів для управління даними в системі.

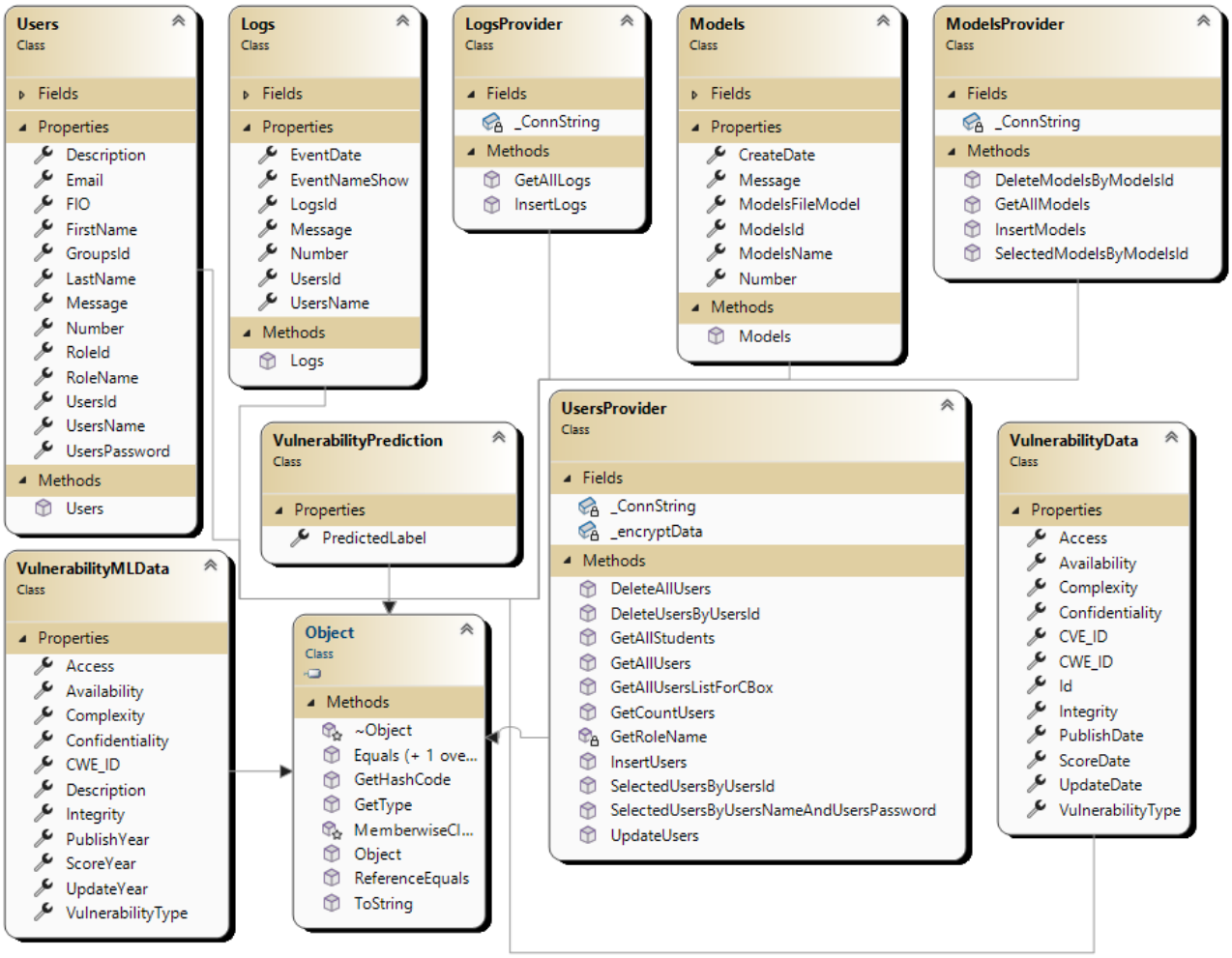

Рис. 3.7. Діаграма класів рівня доступу до даних

Класи для управління даними складаються із:

– Users – цей клас зберігає інформацію про користувачів системи і є контейнером для властивостей, які визначають атрибути користувача. До основних властивостей класу відносяться: ім'я користувача, його унікальний ідентифікатор, права доступу, електронна пошта та роль в системі. Клас використовується для передачі даних про користувачів між різними компонентами системи, але не містить методів для обробки цих даних;

– Models – клас, що представляє собою контейнер для даних про моделі машинного навчання, які використовуються в системі для аналізу вразливостей. Він містить такі властивості, як назва моделі, тип моделі, дата її створення, а також версія моделі. Ці дані необхідні для ідентифікації моделі та її подальшого використання в процесах прогнозування і аналізу;

– Logs – клас, що виконує функцію зберігання даних про події системи. Він містить властивості для збереження інформації про час події, тип події, опис події та ідентифікатор користувача, який ініціював подію. Цей клас дозволяє зберігати всі ключові події для подальшого аналізу, моніторингу та діагностики роботи системи;

– VulnerabilityPrediction – клас для збереження даних, що використовуються під час прогнозування вразливостей. Він працює з різними алгоритмами машинного навчання, забезпечуючи обробку та збереження результатів аналізу, а також надання їх іншим частинам системи для прийняття рішень;

– VulnerabilityMLData – клас, що відповідає за зберігання результатів прогнозування, отриманих від моделей машинного навчання. Він містить дані про тип вразливостей, їх ймовірність та інші важливі метрики, які використовуються для подальшого аналізу та моніторингу;

– ModelsProvider – клас, який відповідає за роботу з базою даних моделей машинного навчання. Він містить методи для додавання, оновлення, видалення і отримання інформації про моделі;

– UsersProvider – клас, який забезпечує доступ до бази даних користувачів. Він включає методи для створення, редагування, видалення та пошуку користувачів у базі даних;

– LogsProvider – клас, який відповідає за роботу з базою даних системних подій. Він містить методи для зберігання та отримання інформації про події, які відбулися в системі.

Перелічені класи забезпечують ефективне управління даними на рівні доступу до даних, дозволяючи системі функціонувати стабільно та безпечно.

Рівень бізнес-логіки у системі виконує роль посередника між інтерфейсом користувача та рівнем доступу до даних. Саме на цьому рівні здійснюється обробка інформації, що надходить від користувача, та її трансформація у відповідні запити до баз даних або моделі машинного навчання для подальшого аналізу. Він відповідає за логіку обробки вразливостей та забезпечує взаємодію

між різними частинами системи, що дозволяє здійснювати інтелектуальне прогнозування на основі отриманих даних. На рис. 3.8 зображена діаграма класів рівня бізнес-логіки.

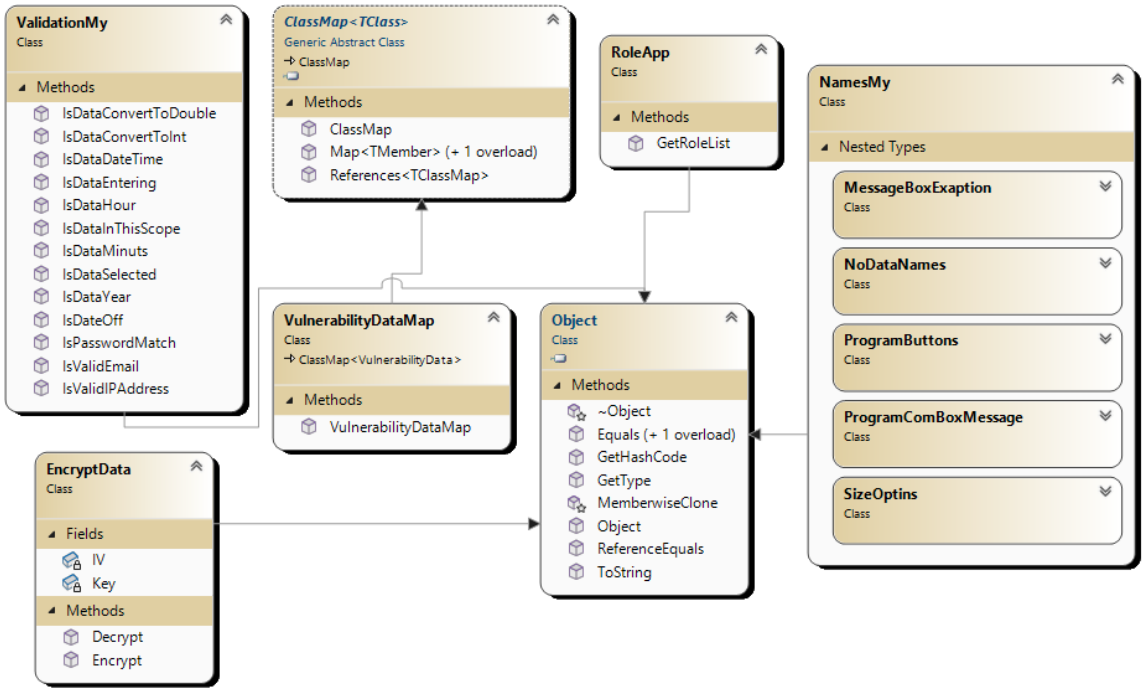

Рис. 3.8. Діаграма класів рівня бізнес-логіки

Основні класи даного рівня складаються із:

– VulnerabilityDataMap – клас, що забезпечує мапінг даних вразливостей, відповідає за обробку та трансформацію даних, отриманих від моделей машинного навчання, у зрозумілі формати для подальшого аналізу та відображення;

– EncryptData – клас, що відповідає за шифрування конфіденційних даних у системі, зокрема даних про користувачів та результати аналізу вразливостей, що забезпечує безпеку інформації на всіх рівнях;

– NamesMy – клас, який управляє обробкою і збереженням імен об’єктів у системі, зокрема імен користувачів та моделей, що допомагає уніфікувати обробку назв у бізнес-логіці;

– RoleApp – клас, який реалізує управління ролями та доступами користувачів у системі, відповідає за визначення прав кожного користувача на основі їхньої ролі в системі;

– ValidationMy – клас, що виконує перевірку коректності даних, які надходять у систему, забезпечуючи відповідність вхідних даних вимогам і стандартам безпеки перед їх подальшою обробкою або збереженням.

Рівень користувацького інтерфейсу у розробленій системі відповідає за взаємодію між користувачем і системою, надаючи зручні інструменти для управління процесами авторизації, перегляду та редагування даних, а також для тестування моделей. Основна мета цього рівня – забезпечити інтуїтивно зрозумілий та функціональний інтерфейс, який дозволяє користувачам виконувати всі необхідні дії без глибоких технічних знань. На рис. 3.9 представлена діаграма класів цього рівня.

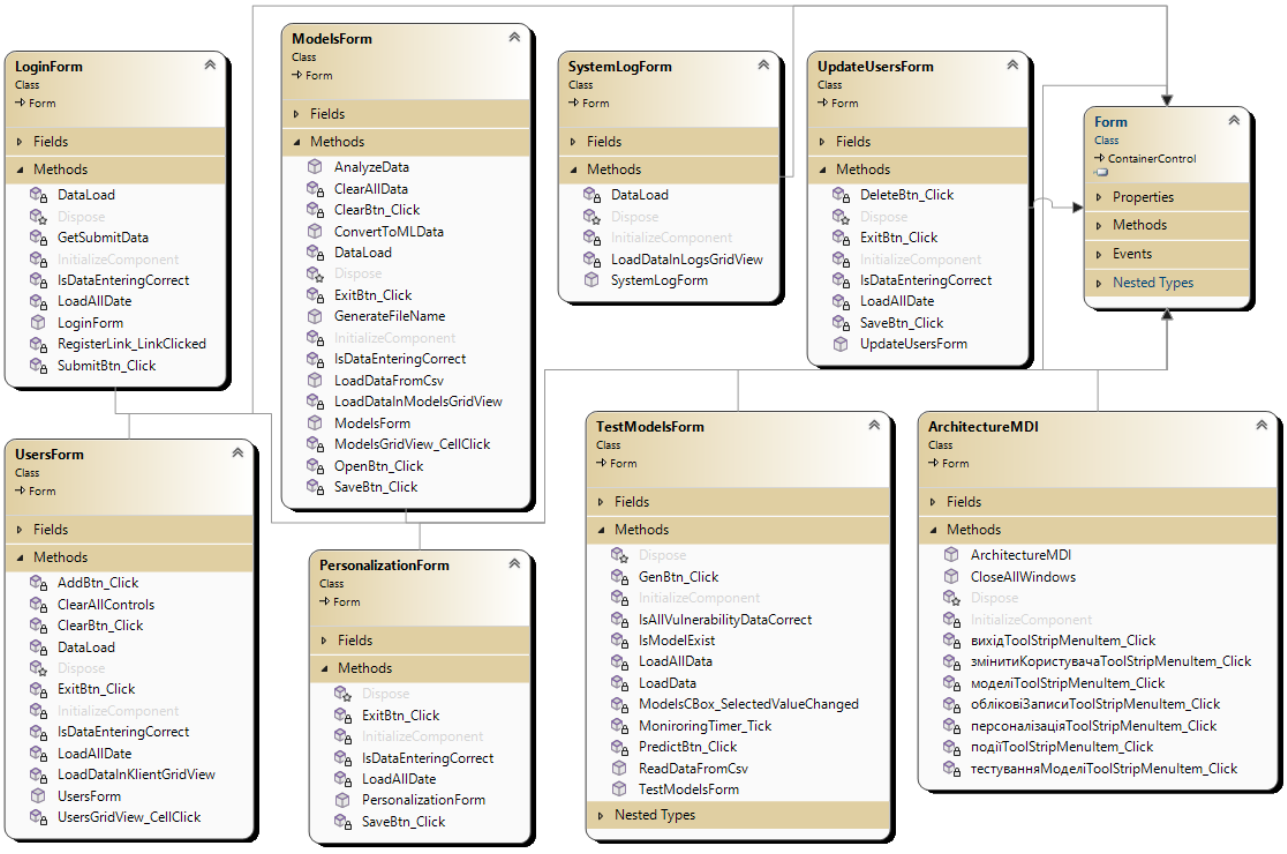

Рис. 3.9. Діаграма класів рівня користувацького інтерфейсу

Діаграма класів рівня користувацького інтерфейсу складається із:

- LoginForm – клас, що реалізує функціонал авторизації користувачів у системі, перевіряючи введені облікові дані та забезпечуючи безпечний доступ до системи;
- PersonalizationForm – клас, відповідальний за управління обліковими даними користувачів, дозволяє змінювати персональні налаштування, такі як паролі, профільні дані тощо;
- SystemLogForm – клас, який відображає інформацію про події системи, включаючи всі дії користувачів та системні операції, що допомагає відслідковувати історію подій для діагностики або моніторингу;
- UpdateUsersForm – клас, що реалізує функціонал редагування або видалення існуючих користувачів системи, надаючи адміністраторам гнучкі можливості управління обліковими записами;
- UsersForm – клас, який забезпечує додавання нових користувачів до системи та перегляд наявних користувачів, дозволяючи адміністраторам керувати обліковими записами;
- ArchitectureMDI – клас, який реалізує головне вікно системи, через яке користувач може взаємодіяти з іншими функціональними модулями та формами системи;
- TestModelsForm – клас, що забезпечує тестування моделей машинного навчання, дозволяючи користувачам проводити оцінку продуктивності та точності моделей у системі.

Архітектура системи дозволяє легко масштабувати систему, забезпечує безпеку та гнучкість у підтримці й розвитку. На рівні презентації реалізовано зручний інтерфейс для користувачів, де вони можуть взаємодіяти з моделями, користувачами та системними подіями. Рівень бізнес-логіки відповідає за обробку даних і взаємодію між інтерфейсом та базою даних, що гарантує коректну роботу системи. Рівень доступу до даних забезпечує надійне збереження інформації про користувачів, моделі та системні події, дозволяючи виконувати всі необхідні операції з даними. Завдяки такій структурі система

здатна ефективно виявляти вразливості програмних застосунків, використовуючи методи машинного навчання.

3.9 Висновок до розділу 3

У рамках даного розділу проведено детальний аналіз методів машинного навчання для виявлення вразливостей в архітектурі програмних застосунків. Розглянуто кілька алгоритмів, зокрема метод Бройдена–Флетчера–Голдфарба–Шанно з максимальним ентропійним підходом, метод стохастичної дуальної координаційної оптимізації та наївний баєсівський класифікатор. Виявлено їхні переваги та недоліки, що дозволило обґрунтовано вибрати метод L-BFGS для ефективного аналізу архітектур із застосуванням штучного інтелекту. Також було обґрунтовано вибір технологій для розробки системи, зокрема, мову програмування C# та бібліотеку ML .NET, що дозволяє інтегрувати машинне навчання в додатки на основі .NET. Описано життєвий цикл машинного навчання у середовищі ML .NET, що визначає стратегію роботи системи з даними.

Окрім цього, було обрано тренувальний набір даних CVE Details, надано детальний опис його атрибутів та обґрунтовано вибір цього набору завдяки його великій кількості різноманітних вразливостей, стандартизованим метрикам CVSS та актуальності бази даних. Розроблено математичну модель для вирішення задачі виявлення вразливостей, яка реалізована у системі, та побудовано алгоритми для навчання моделей і їх збереження, а також алгоритм виявлення вразливостей. На основі трьохрівневої архітектури проєктовано програмне рішення, описано кожен рівень і відповідну діаграму класів.

Результати проведених досліджень і реалізації дозволяють перейти до розробки, тестування системи, перевірки її функціональності та оцінки точності прогнозування вразливостей на основі отриманих моделей машинного навчання.

РОЗДІЛ 4
ОПИС ТА ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Опис етапів розробки системи

Процес розробки системи для виявлення вразливостей в архітектурі програмних застосунків із використанням методів машинного навчання включає кілька послідовних етапів, кожен з яких забезпечує логічну послідовність у реалізації ключових функцій та властивостей системи (рис. 3.1).

Рис. 4.1. Етапи розробки системи

До основних етапів розробки належать:

– аналіз вимог – на першому етапі проводиться детальний аналіз вимог до системи. Це включає вивчення функціональних і нефункціональних вимог, оцінку потреб користувачів, а також аналіз потенційних загроз і типів вразливостей, які система повинна виявляти;

– проєктування архітектури системи – на цьому етапі розробляється архітектура системи, включаючи загальну структуру та взаємодію між основними компонентами. Особлива увага приділяється модульній структурі системи та способам інтеграції машинного навчання для аналізу вразливостей;

– розробка бази даних – створюється структура бази даних, яка зберігатиме інформацію про вразливості, параметри машинного навчання, результати тестування та інші дані, необхідні для коректної роботи системи. На цьому етапі також визначаються атрибути, що будуть використані для тренування моделей;

– розробка модулів системи – реалізуються основні модулі системи, включаючи модуль машинного навчання, який проводить аналіз вразливостей, та модуль для взаємодії з користувачем, що дозволяє переглядати та інтерпретувати результати аналізу;

– тестування системи – система проходить ретельне тестування, під час якого перевіряється коректність роботи всіх її компонентів, включаючи точність моделі машинного навчання, роботу з базою даних, та інші функціональні можливості. На цьому етапі виявляються помилки та недоліки, які необхідно виправити для забезпечення високої якості роботи;

– усунення помилок – якщо під час тестування виявляються помилки або неточності в роботі системи, вони усуваються на цьому етапі. Виправлення помилок дозволяє підвищити стабільність та надійність системи.

Процес завершується, коли система успішно пройде всі етапи розробки, і жодних помилок не буде виявлено.

4.2 Розробка програмного рішення

Розробка програмного забезпечення для системи виявлення вразливостей спрямована на створення функціонального та гнучкого рішення, яке здатне ефективно взаємодіяти з користувачем, обробляти великі обсяги даних і використовувати методи машинного навчання для виявлення загроз у програмних застосунках. Це вимагає ретельного підходу до проектування кожного компонента, зокрема інтерфейсу користувача, який має забезпечувати інтуїтивне та зручне управління процесами системи.

На рис. 4.2 представлено код у якому система дозволяє користувачу вибрати файл із тренувальними даними та подальшої його обробки.

```
// Створення діалогового вікна для відкриття файлу
OpenFileDialog openFileDialog = new OpenFileDialog();
// Налаштування властивостей діалогового вікна
openFileDialog.Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*";
openFileDialog.FilterIndex = 2;
openFileDialog.RestoreDirectory = true;
// Відображення діалогового вікна та обробка результату
if (openFileDialog.ShowDialog() == DialogResult.OK) {
  _Path = openFileDialog.FileName;
  FileNameTBox.Text = openFileDialog.FileName;
```

Рис. 4.2. Вибір вибрати файлу із тренувальними даними

На початку викликається діалогове вікно за допомогою класу OpenFileDialog, яке дає змогу користувачу здійснити вибір файлу. Далі налаштовуються фільтри файлів, що обмежують вибір до текстових файлів або CSV-файлів, що мінімізує ризик помилок через вибір невідповідного формату. Крім того, активується функція збереження попереднього каталогу, що полегшує повторний доступ до файлів у разі повторного відкриття діалогового вікна.

Після підтвердження вибору файлу шлях до нього зберігається у спеціальній змінній, що дозволяє системі швидко доступатися до файлу для подальшої обробки. Ім’я вибраного файлу також відображається у текстовому полі інтерфейсу, що надає користувачеві візуальне підтвердження коректності вибору. Це забезпечує зручність у роботі та дозволяє уникнути помилок при підготовці даних для аналізу.

Рис. 4.3 відображає код класу «VulnerabilityDataMap», який успадковує клас ClassMap і відповідає за мапінг полів об'єкта VulnerabilityData на відповідні властивості під час обробки даних.

```
public class VulnerabilityDataMap : ClassMap<VulnerabilityData> {
    0 references
    public VulnerabilityDataMap() {
        Map(m => m.Id);
        Map(m => m.CVE_ID);
        Map(m => m.CWE_ID);
        Map(m => m.VulnerabilityType);
        Map(m => m.PublishDate).TypeConverterOption.Format("dd.MM.yyyy");
        Map(m => m.UpdateDate).TypeConverterOption.Format("dd.MM.yyyy");
        Map(m => m.ScoreDate).TypeConverterOption.Format("dd.MM.yyyy").Name("Score");
        Map(m => m.Access);
        Map(m => m.Complexity);
        Map(m => m.Confidentiality);
        Map(m => m.Integrity);
        Map(m => m.Availability);
    }
}
```

Рис. 4.3. Код класу «VulnerabilityDataMap»

У конструкторі класу відбувається прив'язка полів даних до конкретних атрибутів моделі. Поля, такі як Id, CVE_ID, CWE_ID, VulnerabilityType, зв'язуються безпосередньо зі своїми відповідниками у моделі. Для дат публікації, оновлення та оцінки ризику використовується спеціальний формат дати «dd.MM.yyyy», що забезпечує правильне відображення цих полів під час обробки. Поля Access, Complexity, Confidentiality, Integrity та Availability також мапуються для подальшого використання у процесі аналізу вразливостей.

Метод LoadDataFromCsv відповідає за завантаження даних про вразливості з файлу формату CSV і перетворення їх у список об'єктів типу VulnerabilityData (рис. 4.4). Спочатку створюється порожній список для збереження зчитаних даних. Далі налаштовується конфігурація для роботи з CSV-файлом, яка включає використання крапки з комою як роздільника, врахування заголовків стовпців та обробку можливих помилок або відсутніх полів без зупинки процесу зчитування. Також вказується кодування файлу, яке використовується для коректного зчитування тексту.

```
public  List<VulnerabilityData> LoadDataFromCsv(string filePath) {
  var vulnerabilities = new List<VulnerabilityData>();
  var config = new CsvConfiguration(CultureInfo.InvariantCulture) {
    Delimiter = ";",
    HasHeaderRecord = true,
    MissingFieldFound = null,
    BadDataFound = null,
    Encoding = System.Text.Encoding.UTF8
  };
  using (var reader = new StreamReader(filePath))
  using (var csv = new CsvReader(reader, config)) {
    // Реєстрація мапи класу
    csv.Context.RegisterClassMap<VulnerabilityDataMap>();
    vulnerabilities = csv.GetRecords<VulnerabilityData>().ToList();
  }
  return vulnerabilities;
}
```

Рис. 4.4. Код методу «LoadDataFromCsv»

У тілі методу відкривається потік для читання файлу за допомогою StreamReader, після чого створюється об'єкт CsvReader з використанням раніше налаштованої конфігурації. Важливим етапом є реєстрація мапи класу VulnerabilityDataMap, яка забезпечує правильне зіставлення полів CSV-файлу з властивостями об'єкта VulnerabilityData. Після цього дані зчитуються та перетворюються в список об'єктів типу VulnerabilityData, який повертається методом.

Метод «AnalyzeData» виконує аналіз списку об'єктів типу VulnerabilityData і виводить результати аналізу в текстове поле RaportTBox (рис. 4.5).

```
public void AnalyzeData(List<VulnerabilityData> vulnerabilities) {
  RaportTBox.Text = ("=== Аналіз даних ===\r\n");
  RaportTBox.Text += ($"Загальна кількість записів у тренувальному наборі: " +
    $"{vulnerabilities.Count}\r\n");
  var vulnerabilitiesByType = vulnerabilities.GroupBy(v => v.VulnerabilityType)
      .Select(group => new { Type = group.Key, Count = group.Count() });
  foreach (var item in vulnerabilitiesByType) {
    RaportTBox.Text += ($"Тип: {item.Type}, Кількість: {item.Count}\r\n");
  }
  var vulnerabilitiesByDate = vulnerabilities.GroupBy(v => v.PublishDate.Year)
      .Select(group => new { Year = group.Key, Count = group.Count() });
  RaportTBox.Text += ("Кількість вразливостей за роками публікації:\r\n");
  foreach (var item in vulnerabilitiesByDate) {
    RaportTBox.Text += ($"Рік: {item.Year}, Кількість: {item.Count}\r\n");
  }
}
```

Рис. 4.5. Код методу «AnalyzeData»

На початку метод ініціалізує звіт, додаючи заголовок «Аналіз даних»" та інформацію про загальну кількість записів у наданому тренувальному наборі вразливостей. Далі метод виконує групування даних за типом вразливості, використовуючи метод GroupBy, що дозволяє об'єднати записи за типами атак і порахувати кількість кожного типу. Для кожного типу вразливості у звіт додається відповідний рядок з інформацією про кількість записів.

У фрагменті коду на рис. 4.6 створюється об'єкт MLContext, який використовується для роботи з інструментами машинного навчання у .NET. Параметр seed встановлюється у значення 0 для забезпечення стабільності та повторюваності результатів при використанні випадкових процесів у навчанні моделей. Цей об'єкт виступає як контекст для налаштування та запуску алгоритмів машинного навчання, включаючи обробку даних, тренування моделей і виконання прогнозів.

```
mlContext = new MLContext(seed: 0);
```

Рис. 4.6. Створення контексту машинного навчання

На рис. 4.7 представлено код, у якому створюється конвеєр для підготовки даних, який виконує кілька трансформацій для тренування моделі машинного навчання.

```
var dataProcessPipeline = mlContext.Transforms.Conversion.MapValueToKey("Label")
 .Append(mlContext.Transforms.Categorical.OneHotEncoding("Access"))
 .Append(mlContext.Transforms.Categorical.OneHotEncoding("Complexity"))
 .Append(mlContext.Transforms.Categorical.OneHotEncoding("Confidentiality"))
 .Append(mlContext.Transforms.Categorical.OneHotEncoding("Integrity"))
 .Append(mlContext.Transforms.Categorical.OneHotEncoding("Availability"))
 .Append(mlContext.Transforms.Concatenate("Features", "CWE_ID", "PublishYear",
 "UpdateYear", "ScoreYear", "Access", "Complexity", "Confidentiality", "Integrity", "Availability"))
 .Append(mlContext.Transforms.NormalizeMinMax("Features"));
```

Рис. 4.7. Побудова тренувального конвеєра з нормалізацією

На початку значення для цільової змінної (мітки) перетворюються у ключі за допомогою MapValueToKey, що дозволяє працювати з категоріальними даними. Далі для колонок Access, Complexity, Confidentiality, Integrity та

Availability застосовується метод OneHotEncoding, який перетворює категоріальні ознаки в числові, щоб модель могла працювати з ними.

Після цього всі зазначені ознаки, включаючи CWE_ID, PublishYear, UpdateYear та інші, об'єднуються в єдиний вектор Features для полегшення обробки. Останній крок – це нормалізація ознак за допомогою методу NormalizeMinMax, що масштабує значення ознак у межах від 0 до 1, забезпечуючи, що всі ознаки матимуть однаковий вплив на модель під час тренування. Цей конвеєр готує дані для моделі машинного навчання, забезпечуючи правильне кодування та нормалізацію вхідних ознак.

У фрагменті коду на рис. 4.8 представлено вибір і налаштування алгоритму для багатокласової класифікації, який використовує метод обмеженої оптимізації L-BFGS для максимізації ентропії.

```
var trainer =
  mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy("Label", "Features")
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));
```

Рис. 4.8. Вибір і налаштування алгоритму для багатокласової класифікації

Модель тренується на основі ознак, що містяться в стовпці Features, а результат класифікації буде передбачений для стовпця Label. Після тренування модель перетворює ключі передбачених міток назад у їх початкові значення за допомогою MapKeyToValue, що дозволяє отримати результат у зручному для користувача форматі. Таким чином, ця модель використовується для виконання багатокласових задач класифікації, де для кожного запису визначається одна з кількох можливих категорій.

Представлений на рис. 4.9 код спочатку завершує створення повного тренувального конвеєра шляхом об'єднання раніше побудованого конвеєра підготовки даних (dataProcessPipeline) з вибраним алгоритмом навчання (trainer). Це дозволяє об'єднати всі етапи, такі як трансформація ознак, їх нормалізація та сам процес навчання моделі, в єдину послідовність, що забезпечує ефективне управління тренувальним процесом і спрощує його реалізацію.

```
var trainingPipeline = dataProcessPipeline.Append(trainer);
// Розділення даних на тренувальні та тестові набори
var splitData = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
```

Рис. 4.9. Звершення побудови конвеєра та розподіл даних

Після цього відбувається розподіл вхідних даних на тренувальний та тестовий набори за допомогою методу TrainTestSplit, при якому 80% даних використовуються для навчання моделі, а 20% – для тестування. Такий підхід дозволяє провести незалежну оцінку якості моделі на даних, що не були використані під час тренування, що є критичним для визначення її здатності до узагальнення та точності передбачень.

На наступному етапі відбувається запуск вимірювання часу за допомогою об'єкта Stopwatch, який використовується для точного фіксування тривалості процесу тренування моделі (рис. 4.10). Цей об'єкт запускається на початку тренувального процесу і автоматично фіксує час до його завершення, що дозволяє виміряти продуктивність та ефективність моделі під час навчання. Це особливо важливо для оцінки ресурсомістких процесів, де час навчання є критичним фактором.

```
var stopwatch = Stopwatch.StartNew();
// Тренування моделі на тренувальному наборі
trainedModel = trainingPipeline.Fit(splitData.TrainSet);
// Зупиняємо вимірювання часу
stopwatch.Stop();
var elapsedTime = stopwatch.Elapsed;
// Виводимо загальний час навчання
RaportTBox.Text += ($"Час навчання моделі: {elapsedTime.TotalSeconds:F2} секунд\r\n");
```

Рис. 4.10. Тренування моделі та вимірювання часу цього процесу

Після запуску Stopwatch тренувальний конвеєр trainingPipeline застосовується до тренувального набору даних splitData.TrainSet, обробляючи дані та виконуючи всі необхідні трансформації. Алгоритм навчається на цих даних, і після завершення процесу модель зберігається у змінній trainedModel для подальшого використання в прогнозуванні або тестуванні на нових даних.

Як тільки тренувальний процес завершується, вимірювання часу зупиняється за допомогою методу Stop. Значення часу, яке було витрачено на навчання, зберігається у змінну elapsedTime, і результуючий час виводиться у текстове поле RaportTBox у форматі кількості секунд. Це дозволяє користувачам бачити, скільки часу зайняв процес тренування моделі, що може бути важливо для оцінки продуктивності системи.

Код на рис. 4.11 виконує оцінку якості навченої моделі на тестовому наборі даних. Спочатку у текстове поле RaportTBox виводиться повідомлення про початок оцінки моделі. Після цього навчена модель trainedModel застосовується до тестового набору splitData.TestSet, щоб отримати передбачення для кожного зразка даних.

```
RaportTBox.Text += ("Оцінка моделі на тестовому наборі...\r\n");
var predictions = trainedModel.Transform(splitData.TestSet);
var metrics = mlContext.MulticlassClassification.Evaluate(predictions,
  labelColumnName: "Label");
// Виведення ключових метрик
RaportTBox.Text += ($"MacroAccuracy : {metrics.MacroAccuracy:P2}\r\n");
RaportTBox.Text += ($"MicroAccuracy: {metrics.MicroAccuracy:P2}\r\n");
```

Рис. 4.11. Оцінка якості навченої моделі на тестовому наборі даних

За допомогою методу Evaluate проводиться оцінка точності моделі, зокрема, розраховуються метрики, такі як MacroAccuracy та MicroAccuracy. Після цього значення цих метрик виводяться у текстове поле, що дозволяє користувачу побачити загальну точність моделі у відсотках, як для окремих класів, так і для всієї моделі загалом.

Метод «SaveBtn_Click» викликається при натисканні на кнопку збереження. Спочатку відбувається перевірка, чи введені дані коректні за допомогою методу «IsDataEnteringCorrect» (рис. 4.12). Якщо дані правильні, починається процес збереження моделі. Генерується шлях до файлу для збереження моделі, зокрема форматується ім'я файлу з використанням методу «GenerateFileName» і додається відповідний каталог. Далі отримується шлях до поточного проєкту за допомогою System.Reflection, і з використанням цього

шляху модель зберігається на диск за допомогою методу «Save», де також зберігається схема даних.

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    //Збереігання моделі
    string pathName = @"\teach\" + GenerateFileName() + ".zip";
    string localProj =
      System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    _ModelsProvider.InsertModels(ModelsNamesTBox.Text, pathName);
    mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
    ClearAllData();
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було навчено модель " +
      ModelsNamesTBox.Text, DateTime.Now);
    MessageBox.Show("Дані успішно збережено!");
  }
}
```

Рис. 4.12. Збереження моделі

Після цього метод очищує всі введені дані за допомогою «ClearAllData». Також у базу даних логів записується подія про те, що користувач навчив нову модель, включаючи її назву та поточний час. Запис проводиться за допомогою «InsertLogs». Після успішного завершення збереження моделі на екран виводиться повідомлення, що операція завершилася успішно за допомогою MessageBox.Show.

Подія «ModelsCBox_SelectedValueChanged» викликається при зміні вибраного значення у випадаючому списку моделей (рис. 4.13).

```
private void ModelsCBox_SelectedValueChanged(object sender, EventArgs e) {
  if (_IsModelsLoad && IsModelExist()) {
    _SelectedModels = _ModelsProvider.SelectedModelsByModelsId(
       Convert.ToInt32(ModelsCBox.SelectedValue));
    LoadData(_SelectedModels.ModelsFileModel);
  }
}
```

Рис. 4.13. Подія для завантаження даних моделі

На початку перевіряється, чи завантажено список моделей і чи обрана модель існує, використовуючи змінну _IsModelsLoad та метод «IsModelExist». Якщо ці умови виконуються, далі відбувається отримання вибраної моделі з бази даних за допомогою «SelectedModelsByModelsId», куди передається

ідентифікатор обраної моделі, який конвертується у ціле число з ModelsCBox.SelectedValue.

Після отримання моделі метод викликає функцію LoadData для завантаження даних, використовуючи файл моделі ModelsFileModel, який зберігається в отриманому об'єкті _SelectedModels. Це забезпечує завантаження відповідної моделі для подальшої роботи з нею, наприклад, для аналізу або тестування.

Метод «LoadData» відповідає за завантаження моделі з вказаного файлу і створення механізму для прогнозування на основі цієї моделі (рис. 4.14). Спочатку формується повний шлях до файлу, поєднуючи шлях до папки програми з переданим параметром FilePath. Далі оголошується змінна modelSchema, яка буде використовуватися для збереження схеми даних моделі після її завантаження.

```
private void LoadData(string FilePath) {
  string localProj = Application.StartupPath + FilePath;
  // Визначте DataViewSchema для конвеєра підготовки даних і навченої моделі
  DataViewSchema modelSchema;
  // Завантаження моделі
  ITransformer model = context.Model.Load(localProj, out modelSchema);
  //Створення механізму прогнозування
  predictionEngine =
    context.Model.CreatePredictionEngine<VulnerabilityMLData,
    VulnerabilityPrediction>(model);
}
```

Рис. 4.14. Код методу «LoadData»

Модель завантажується за допомогою методу «Load», який повертає модель у змінну model, а схема зберігається у modelSchema. Після цього створюється механізм прогнозування за допомогою методу «CreatePredictionEngine», який дозволяє використовувати завантажену модель для виконання прогнозів на основі класу «VulnerabilityMLData» і формувати результати у вигляді об'єкта VulnerabilityPrediction. Це забезпечує можливість подальшої обробки даних і прогнозування на основі завантаженої моделі.

Метод «PredictBtn_Click» викликається при натисканні на кнопку прогнозування (рис. 4.15). Спочатку він перевіряє, чи всі введені дані коректні за

допомогою методу «IsAllVulnerabilityDataCorrect», а також перевіряє наявність завантаженої моделі через «IsModelExist». Якщо всі умови виконуються, створюється об'єкт типу VulnerabilityMLData на основі введених користувачем значень, таких як CWE_ID, рік публікації, рік оновлення та інші властивості, необхідні для прогнозування. Ці дані беруться з відповідних полів форми і конвертуються у потрібні формати для подальшої обробки.

```
private void PredictBtn_Click(object sender, EventArgs e) {
  // Перевіряємо, чи всі введені дані коректні
  if (IsAllVulnerabilityDataCorrect() && IsModelExist()) {
    // Створюємо об'єкт з даними на основі введених значень
    VulnerabilityMLData testVulnerabilityData = new VulnerabilityMLData {
      CWE_ID = (float)Convert.ToDouble(CWE_IDTBox.Text),
      PublishYear = (float)(PublishYearDTP.Value.Year),
      UpdateYear = (float)(UpdateYearDTP.Value.Year),
      ScoreYear = (float)(ScoreYearDTP.Value.Year),
      Access = AccessCBox.Text,
      Complexity = ComplexityCBox.Text,
      Confidentiality = ConfidentialityCBox.Text,
      Integrity = IntegrityCBox.Text,
      Availability = AvailabilityCBox.Text
    };
    // Прогнозування на основі введених даних
    var prediction = predictionEngine.Predict(testVulnerabilityData);
    MonitoringTBox.Clear();
    // Формуємо результат прогнозування
    var answer = new StringBuilder();
    answer.AppendLine("\r\n--- Прогнозування ---");
    answer.AppendLine($"\nВиявлено вразливість: {prediction.PredictedLabel}");
    // Виведення результату прогнозування
    MonitoringTBox.Text += answer.ToString();
  }
}
```

Рис. 4.15. Код методу «PredictBtn_Click»

Після створення об'єкта з даними для прогнозування використовується механізм прогнозування, налаштований раніше, щоб отримати результат на основі введених даних. Результат прогнозування зберігається в змінній prediction, після чого текстове поле MonitoringTBox очищується. За допомогою StringBuilder формується рядок з результатами прогнозування, який виводиться на екран, повідомляючи користувача про виявлену вразливість на основі введених параметрів.

Описані ключові аспекти розробки програмного рішення зосереджені на інтеграції механізмів машинного навчання з інтерфейсом користувача для

прогнозування вразливостей. Система забезпечує інтуїтивну взаємодію та автоматизоване прогнозування загроз на основі введених даних, що дозволяє ефективно обробляти великі обсяги інформації. Це забезпечує користувачам зручні інструменти для аналізу та прийняття рішень щодо виявлених загроз, хоча наведений опис охоплює лише основні моменти розробки.

4.3 Підготовка даних та демонстрація роботи системи

Формування та підготовка датасету є комплексним процесом, який передбачає послідовне виконання низки процедур, що дозволяють досягти необхідної точності та релевантності даних для дослідження. Даний процес включає забезпечення збалансованості, релевантності та відповідності зібраної інформації задачам дослідження, що дає можливість виявляти вразливості з особливою увагою до застосунків, які використовують алгоритми штучного інтелекту. Підготовка датасету поділяється на окремі етапи, такі як очищення, ретельний відбір та трансформація даних, а також побудову необхідних структур для валідації роботи системи на контрольних прикладах.

На основі вихідного датасету «CVE Details» проведено очищення та фільтрацію даних, спрямованих на видалення всіх неповних та нерелевантних записів. Метою очищення стало забезпечення максимально повного та точного датасету, з якого були вилучені записи з такими характеристиками:

– неповні записи. У випадках, коли деякі атрибути (тип вразливості, рік публікації або серйозність) були відсутні, запис видалявся з метою запобігання неповноті та некоректній інтерпретації даних;

– дублювання. Записи, що мали ідентичні параметри, як-от однаковий тип вразливості та рік, були усунені, щоб уникнути зайвого впливу на результат під час навчання моделі.

Після видалення зайвих даних відбулося сортування вразливостей за їх впливом на архітектури програмного забезпечення, які використовують алгоритми ШІ. Вибірка була обмежена до десяти ключових типів вразливостей,

які мають найбільший вплив на такі системи, серед них: Exec Code, DoS, SQL Injection, XSS, CSRF, та інші. Для кожного з цих типів було обрано по 1050 записів, що забезпечує рівномірний розподіл та зменшує ризик зміщення даних під час тренування моделі.

На основі сформованого збалансованого датасету було проведено навчання моделі для виявлення вразливостей в архітектурі програмних застосунків. Результати навчання моделі представлені на рис. 4.16.

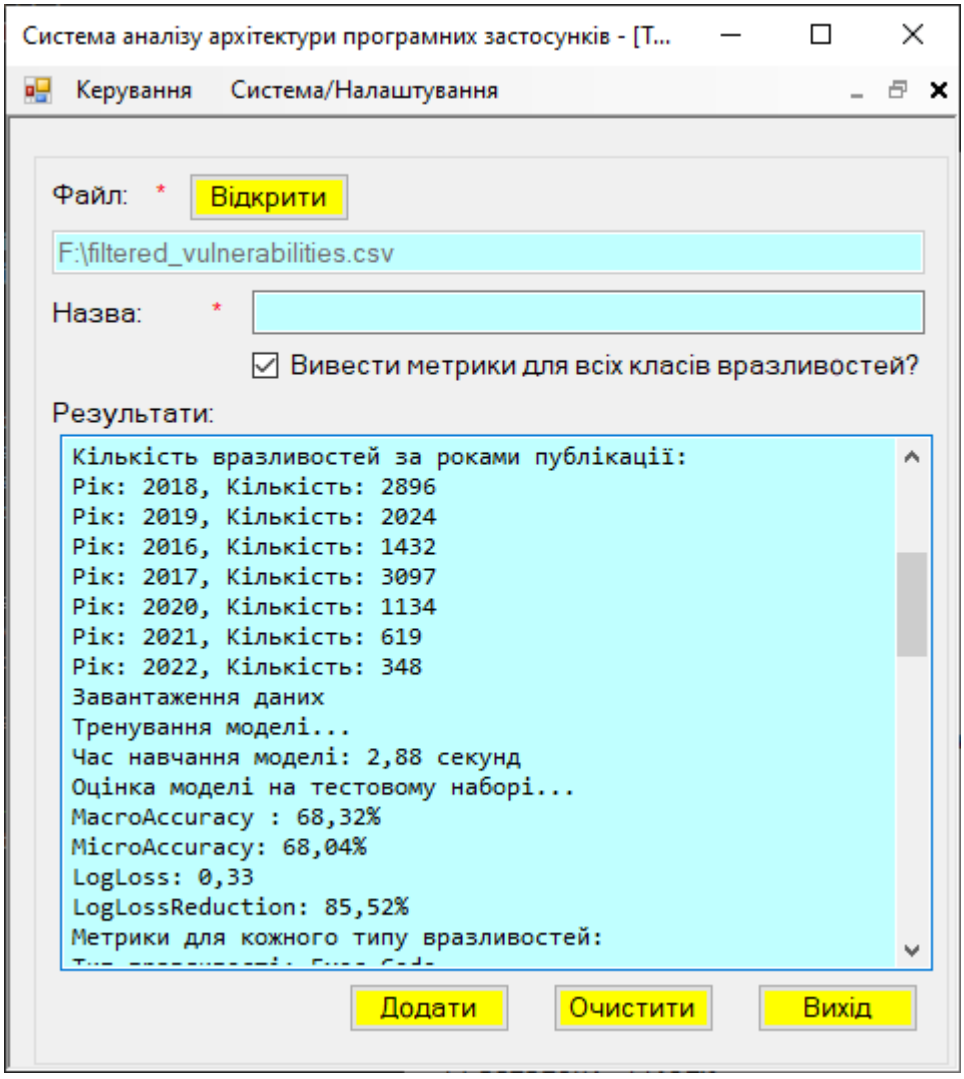

Рис. 4.16. Тренування моделі для виявлення вразливостей у архітектурі

Отримані метрики демонструють високий рівень точності моделі. Зокрема, MacroAccuracy складає 68,36%, що свідчить про ефективну загальну точність моделі при класифікації вразливостей різних типів. MicroAccuracy на рівні

68,09% відображає стабільну роботу моделі при розподілі вразливостей у тестовому наборі. Високе значення LogLossReduction (85,53%) підкреслює здатність моделі мінімізувати похибки при прогнозуванні, що свідчить про надійну оцінку ризиків. Таким чином, отримані показники свідчать про якісне налаштування моделі для вирішення задач аналізу вразливостей в архітектурі застосунків.

Після завершення навчання модель можна зберегти у системі, що дозволяє швидко завантажувати її для подальших прогнозувань без повторного навчання. Це забезпечує ефективність роботи системи та дозволяє користувачам оперативно аналізувати нові вхідні дані. На рис. 4.17 показано, як реалізується процес прогнозування вразливостей за допомогою цієї навченої моделі.

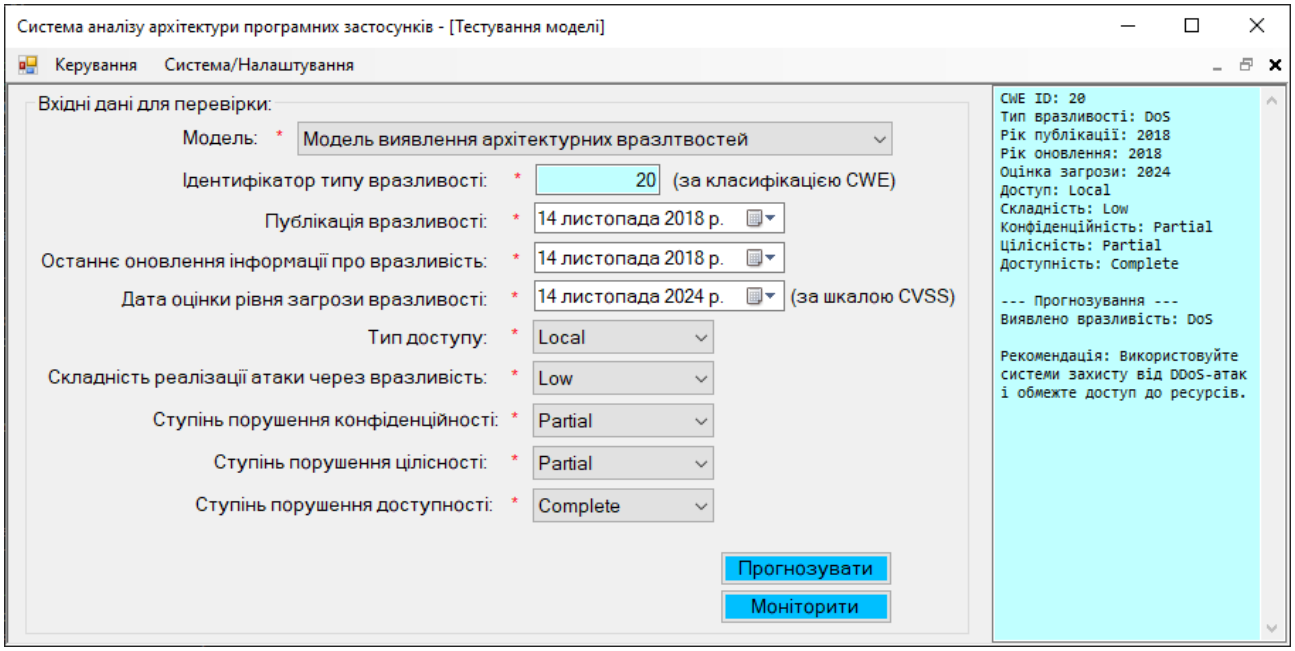

Рис. 4.17. Виявлення вразливості в архітектурі на основі вхідних даних

Інтерфейс системи дозволяє користувачу вводити відповідні параметри (ідентифікатор типу вразливості, дати публікації та оновлення, рівні впливу на безпеку) для здійснення оцінки. Модель на основі введених значень визначає ймовірність наявності вразливості та надає рекомендації щодо її усунення.

4.4 Тестування системи: експерименти, їх опис та аналіз результатів

Після навчання моделі та оцінки її основних метрик було проведено тестування системи для практичної перевірки її здатності до виявлення архітектурних вразливостей у реальних сценаріях. З цією метою виконано серію експериментів, що спрямовані на оцінку точності класифікації різних типів вразливостей залежно від специфічних вхідних параметрів. Експерименти були спрямовані на перевірку точності прогнозів, швидкості обробки даних та ефективності рекомендацій щодо забезпечення безпеки інформаційних систем.

В рамках першого експериментального сценарію (рис. 4.18) система була налаштована на аналіз вразливості, яка має ідентифікатор типу 19 (згідно з класифікацією CWE), дату публікації 2016 року та останнє оновлення інформації про вразливість на момент тестування – 2017 року. Вхідні параметри також включали середній рівень складності реалізації атаки через віддалений доступ та частковий ступінь порушення конфіденційності, цілісності та доступності.

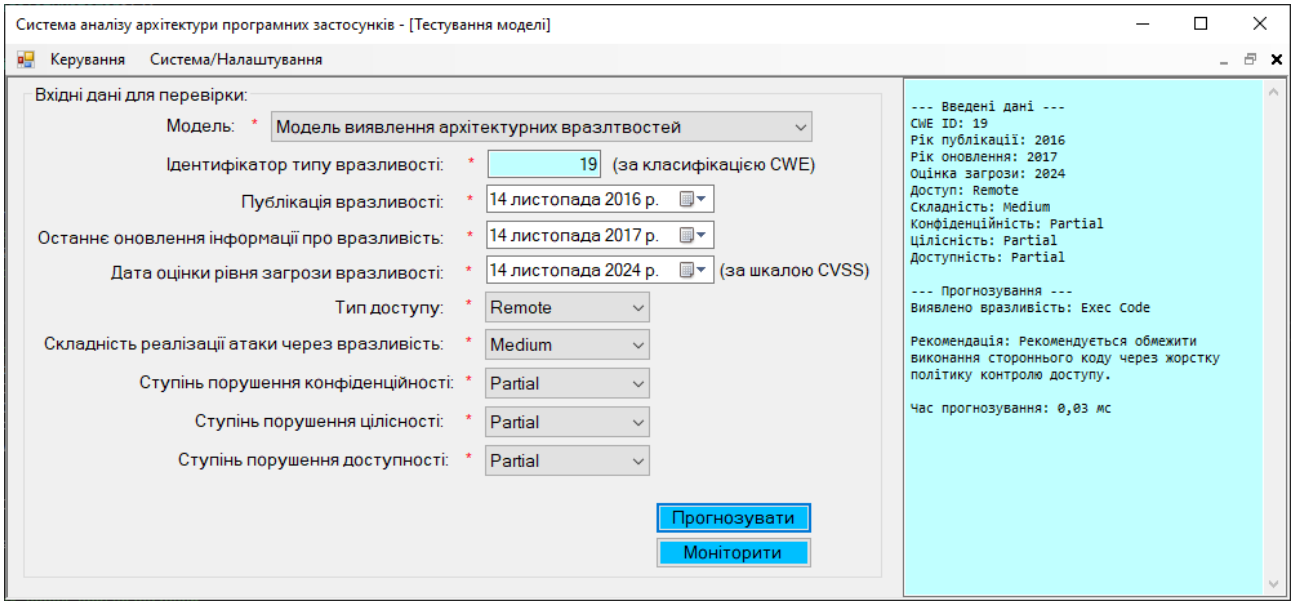

Рис. 4.18. Тестування системи для 1-го сценарію

Результатом прогнозування стало виявлення типу вразливості – Exec Code, що вказує на можливість виконання зловмисного коду. На основі проведеного аналізу система надала рекомендацію обмежити доступ до вразливої компоненти

через жорстку політику контролю доступу, що є доцільним рішенням для зниження ризиків.

Час прогнозування становив лише 0,03 секунди, що демонструє високу швидкість обробки вхідних даних. Цей результат підтверджує, що система здатна швидко та ефективно аналізувати загрози, надаючи точні рекомендації для забезпечення захисту інформаційних систем.

У другому сценарії (рис. 4.19) система була налаштована для виявлення вразливості з ідентифікатором типу 119 (згідно з класифікацією CWE). Вхідні параметри включали дату публікації 2019 року, останнє оновлення інформації про вразливість станом на 2021 року та оцінку рівня загрози на 2024 року (за шкалою CVSS). Також було зазначено, що тип доступу до вразливості є віддаленим (Remote), складність атаки – низька (Low), а ступінь порушення конфіденційності та доступності – частковий (Partial), тоді як порушення цілісності – повний (Complete).

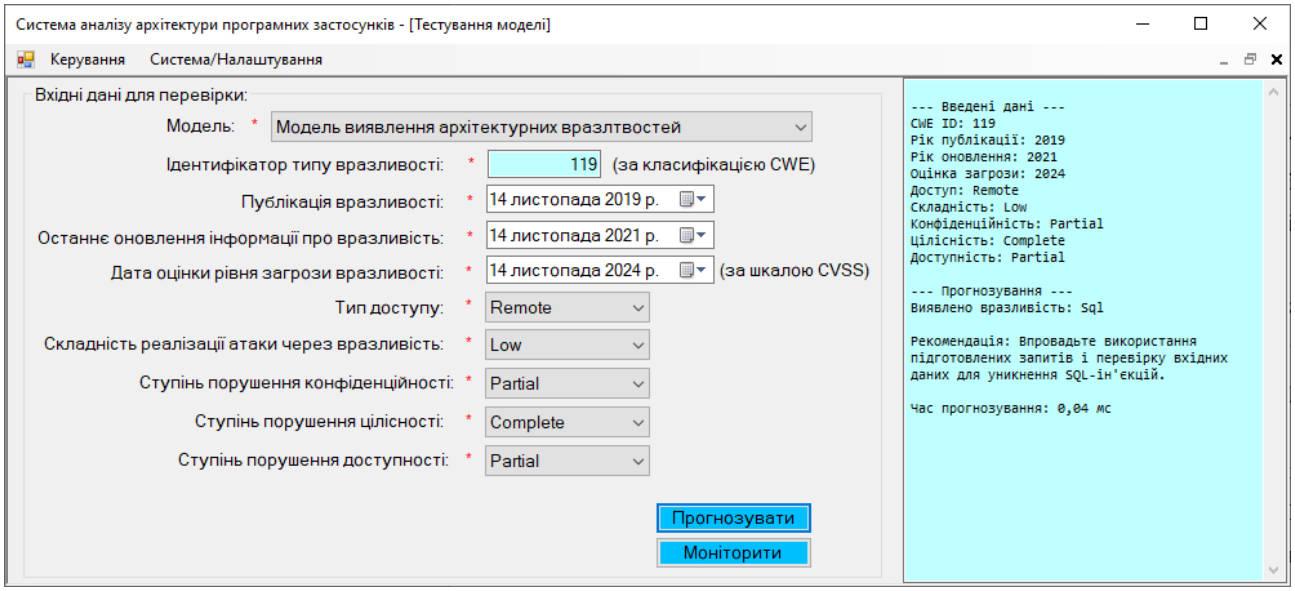

Рис. 4.19. Тестування системи для 2-го сценарію

Система виявила вразливість типу SQL Injection, що є одним із критичних типів вразливостей для інформаційних систем, оскільки дозволяє зловмисникам маніпулювати базами даних за допомогою шкідливих запитів. На основі проведеного аналізу система рекомендує впровадити підготовлені SQL-запити та

перевірку вхідних даних для уникнення SQL-ін’єкцій, що є загальноприйнятими заходами захисту від подібних атак.

Час прогнозування у даному сценарії становив 0,04 секунди, що вкотре підкреслює високу швидкодію системи при обробці даних. Як показано на рис. 4.19, система продемонструвала здатність точно виявляти потенційні вразливості та надавати обґрунтовані рекомендації для покращення захисту програмного забезпечення.

У третьому сценарії (рис. 4.20) було проведено аналіз вразливості з ідентифікатором типу 119, що відповідає класифікації CWE.

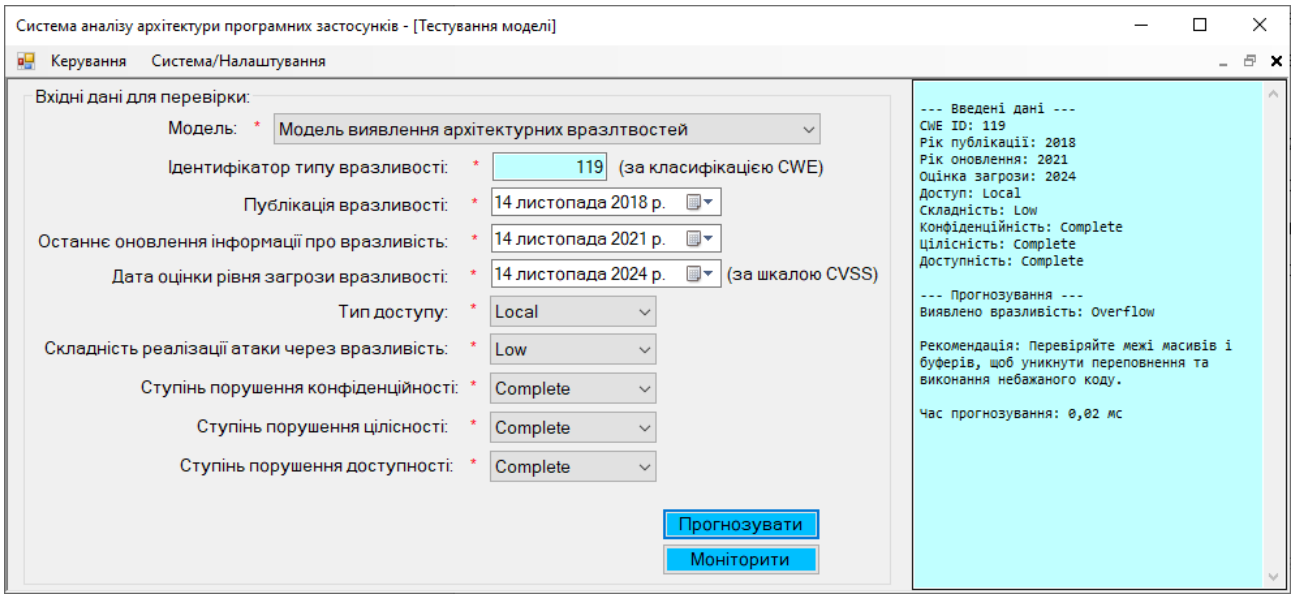

Рис. 4.20. Тестування системи для 3-го сценарію

Параметри вхідних даних включали дату публікації 2018 року, останнє оновлення інформації про вразливість станом на 2021 року та оцінку рівня загрози на 2024 року. Для цього сценарію було зазначено, що тип доступу до вразливості є локальним (Local), складність атаки – низька (Low), а ступінь порушення конфіденційності, цілісності та доступності – повний (Complete), що вказує на критичність потенційних наслідків у разі успішної атаки.

Результат прогнозування показав, що виявлена вразливість належить до типу Overflow – перевантаження буферу, що є загрозливим для систем, оскільки може призвести до виконання небажаного коду. Система надала рекомендацію

перевірити межі масивів і буферів, що дозволить уникнути переповнення та знизить ризик експлуатації цієї вразливості.

Час прогнозування у цьому випадку становив лише 0,02 секунди, що вказує на високу продуктивність системи навіть для більш складних конфігурацій параметрів вразливості. Система продемонструвала ефективність у точному визначенні типу загрози та наданні практичних рекомендацій, що сприяє покращенню захищеності програмних архітектур.

На рис. 4.21 представлено аналіз метрик точності, повноти та F1-міри для різних типів вразливостей, що дозволяє оцінити здатність моделі ідентифікувати кожен тип загроз з максимальною точністю. Ці метрики надають уявлення про якість роботи моделі щодо класифікації кожного типу вразливості.

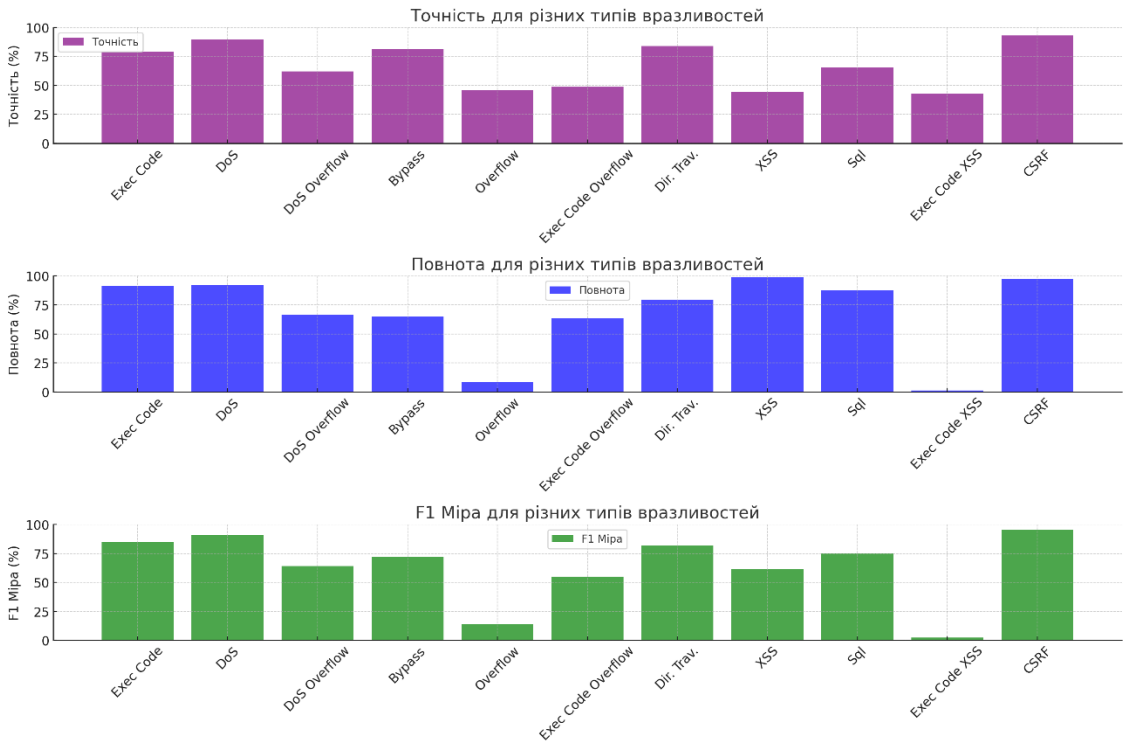

Рис. 4.21. Метрики моделі для кожного типу вразливостей

Загалом, модель демонструє високу точність та повноту для критичних вразливостей, таких як DoS та CSRF, що підтверджує її надійність для більшості типів загроз. Однак існують певні складнощі з точним визначенням таких вразливостей, як Exec Code XSS та Overflow, які потребують додаткового

вдосконалення. Аналіз метрик дозволяє визначити пріоритети для подальшого поліпшення моделі, зокрема шляхом розширення навчальної вибірки та уточнення критеріїв класифікації для слабо представлених типів вразливостей.

Одним з важливих аспектів оцінки системи є її продуктивність, яка характеризується швидкістю обробки вхідних даних та прогнозування результатів. Оптимізована продуктивність дозволяє системі функціонувати в реальному часі, що особливо важливо для ідентифікації загроз безпеці в режимі, максимально наближеному до реальних умов. Для кожного типу вразливостей було проаналізовано середній час прогнозування та обчислено відхилення часу для кожного експериментального сценарію (табл. 4.1).

Таблиця 4.1.

Час прогнозування для різних сценаріїв

Тип вразливості	Середній час прогнозування (сек)	Відхилення часу прогнозування (сек)
Exec Code	0,03	±0,005
DoS	0,04	±0,006
DoS Overflow	0,05	±0,007
Bypass	0,03	±0,005
Overflow	0,05	±0,007
Exec Code Overflow	0,04	±0,006
Dir. Trav.	0,03	±0,005
XSS	0,04	±0,006
SQL Injection	0,02	±0,004
CSRF	0,03	±0,005

Як видно з табл. 4.1, система демонструє стабільно низький час прогнозування для більшості типів вразливостей, з середніми показниками, що коливаються в межах від 0,02 до 0,05 секунд. Найшвидший час прогнозування спостерігається для вразливостей типу SQL Injection (0,02 секунди), що є індикатором оптимізації для поширених типів атак на бази даних. Висока

швидкість обробки підтверджує ефективність алгоритмів, що використовуються системою, а низьке відхилення часу прогнозування свідчить про стабільність роботи при різних сценаріях. У контексті використання системи в реальному часі ці показники дозволяють забезпечити швидку реакцію на потенційні загрози безпеці, що є важливим для забезпечення захищеності інформаційних систем.

На основі проведеного аналізу можна зробити висновок, що система успішно вирішує поставлені завдання з ідентифікації вразливостей та надання відповідних рекомендацій. Отримані метрики підтверджують високу точність і продуктивність моделі, а також релевантність рекомендацій, які відповідають сучасним стандартам кібербезпеки. Швидкий час обробки та точність прогнозування свідчать про готовність системи до використання в реальних умовах, де своєчасність та точність рішень є ключовими для підтримки інформаційної безпеки.

4.5 Порівняння реалізованої системи з існуючими аналогами

Порівняння реалізованої системи з існуючими аналогами дозволяє оцінити її конкурентоспроможність та ефективність у сфері аналізу безпеки коду. Враховуючи різноманітність доступних інструментів, що використовуються для ідентифікації вразливостей у програмному забезпеченні, особливу увагу було приділено трьом популярним рішенням: SonarQube, Fortify Static Code Analyzer (SCA) та Veracode. Кожна з цих систем має свої унікальні характеристики, що визначають їхню придатність для різних типів проєктів та архітектурних рішень.

Метою порівняння було проаналізувати сильні та слабкі сторони кожного інструменту, включаючи підтримку мов програмування, можливості інтеграції з CI/CD, тип аналізу та частоту оновлення баз вразливостей. Окрім цього, особливу увагу було приділено рівню локального розгортання та частоті оновлення бази правил, оскільки ці аспекти є важливими для забезпечення гнучкості й адаптивності у застосуванні інструментів. Розроблене програмне

забезпечення було розглянуто в контексті його здатності відповідати цим критеріям та забезпечувати ефективний аналіз безпеки коду.

У табл. 4.2 наведено порівняння зазначених систем за основними характеристиками, що дозволяє провести більш детальний аналіз їхніх можливостей та обмежень у контексті задач кібербезпеки.

Таблиця 4.2.

Порівняння характеристик наявних інструментів аналізу

Характеристика	SonarQube	Fortify SCA	Veracode	Розроблене ПЗ
Підтримка мов програмування	Понад 30 мов	Понад 25 мов	Понад 25 мов	Можливість додавання специфічних мов
Інтеграція з CI/CD	Є інтеграція з популярними системами	Підтримка інтеграції з CI/CD	Підтримка CI/CD	Можливість інтеграції через API
Тип аналізу	Статичний аналіз коду	Статичний аналіз	Статичний та динамічний, аналіз	Статичний аналіз із елементами ML
Оновлення баз вразливостей	Щомісячне оновлення правил	Щотижневе оновлення баз	Щоденне оновлення баз	Можливість регулярного оновлення вручну
Можливість локального розгортання	Локальне та хмарне	Локальне розгортання	Хмарна платформа	Локальне розгортання
Якість розпізнавання помилок	80%	65%	70%	68%
Частота оновлення бази правил	Щомісячне оновлення	Щотижневе оновлення	Щоденне оновлення	Налаштовується вручну

До переваг розробленого програмного забезпечення належать:

– якість розпізнавання нових вразливостей. Точність моделі на рівні 68% для MicroAccuracy і MacroAccuracy свідчить про те, що система здатна ефективно виявляти потенційні загрози на основі тренувальних даних і постійно може перенавчатися, щоб підвищувати точність;

– інтеграція з машинним навчанням. Можливість використання мультикласової класифікації з алгоритмом L-BFGS надає конкурентну перевагу у виявленні нових загроз та відсутності необхідності у статичних правилах;

– гнучкість у налаштуванні. Система дозволяє адаптувати правила аналізу та додавати специфічні для проєкту мови програмування, що робить її надзвичайно гнучкою та підходить для широкого кола користувачів;

– локальне розгортання з можливістю перенавчання моделей. Система дозволяє постійно оновлювати тренувальні моделі та підвищувати якість виявлення вразливостей, що є важливою перевагою в умовах постійно мінливих загроз.

4.6 Висновок до розділу 4

У рамках даного розділу було описано процес розробки, тестування та оцінки створеної системи для аналізу вразливостей у програмному забезпеченні. На першому етапі було проведено аналіз вимог, що дозволило визначити ключові функціональні та нефункціональні характеристики системи. На основі зібраних вимог, було здійснено проєктування архітектури системи, яке забезпечило структуровану побудову компонентів, зокрема бази даних, та окремих модулів для реалізації основних функцій.

Реалізовано програмне рішення, яке включало частковий опис програмного коду для інтеграції бази даних та обробки вхідних даних. Особливу увагу було приділено підготовці датасету для навчання моделі, що дозволило забезпечити ефективність у розпізнаванні різних типів вразливостей. Отримані метрики підтвердили точність і надійність моделі: MacroAccuracy досягла 68,36%, що

свідчить про її здатність ефективно класифікувати різні типи загроз, тоді як MicroAccuracy на рівні 68,09% демонструє стабільність роботи на тестовому наборі. Висока LogLossReduction (85,53%) підтверджує здатність моделі мінімізувати помилки прогнозування, що є критично важливим для оцінки ризиків.

Було проведено серію експериментів для тестування системи за кількома сценаріями, що включали різні типи вразливостей, зокрема Exec Code, SQL Injection, та Overflow. Для кожного типу вразливостей були проаналізовані метрики точності (Precision), повноти (Recall) та F1-міри, що відображено у вигляді діаграм. Наприклад, для DoS показники точності становлять 89,81%, що свідчить про високу здатність моделі до ідентифікації цього типу загроз. Аналіз часу прогнозування показав, що система здатна виконувати оцінку у середньому за 0,02–0,05 секунд для більшості сценаріїв, що підтверджує її придатність для використання в умовах реального часу.

Для оцінки конкурентоспроможності розробленої системи було проведено порівняння з існуючими аналогами, такими як SonarQube, Fortify SCA та Veracode. Це дало змогу визначити сильні сторони нашого програмного забезпечення, зокрема розширювану підтримку мов програмування, інтеграцію з машинним навчанням, гнучкість у налаштуванні параметрів, можливість локального розгортання та регулярного оновлення бази вразливостей. Ці особливості підкреслюють потенціал розробленої системи для ефективного використання у різних середовищах, включаючи великі організації з високими вимогами до кібербезпеки.

Отримані результати свідчать про успішну реалізацію поставлених задач та досягнення високих показників продуктивності та точності системи. Розроблене програмне забезпечення є надійним інструментом для аналізу вразливостей, яке може забезпечити швидку та точну оцінку безпекових загроз, підтримуючи актуальні стандарти та вимоги кібербезпеки.

ВИСНОВКИ

Дана кваліфікаційна робота присвячена розробці та тестуванню системи аналізу архітектури програмних застосунків з метою виявлення вразливостей, пов'язаних з використанням штучного інтелекту. Розроблена система орієнтована на забезпечення високої якості ідентифікації потенційних загроз, які виникають у складних програмних середовищах, що використовують сучасні алгоритми машинного навчання. В результаті виконання кваліфікаційної роботи отримані наступні результати:

- проаналізовано існуючі вразливості програмних застосунків, зокрема пов’язані з використанням штучного інтелекту, та архітектурні особливості, що сприяють їх виникненню, визначено оптимальні методи і технології для розробки системи, а також обґрунтовано вибір методу машинного навчання для аналізу архітектури програмних застосунків;
- розроблено систему для аналізу архітектури програмних застосунків на основі використання алгоритмів машинного навчання (за допомогою бібліотеки ML.NET), що дало змогу автоматизувати процес виявлення вразливостей, особливо тих, які пов’язані з використанням штучного інтелекту;
- проведено тестування розробленої системи на контрольних сценаріях, що дало змогу оцінити її ефективність та надати рекомендації для вдосконалення.

У другому розділі наведено огляд методів та технологій, які можуть бути застосовані для аналізу архітектури програмних застосунків на предмет вразливостей. Проведено детальний аналіз інструментів, таких як SonarQube, Fortify SCA та Veracode, що використовуються для виявлення вразливостей. На основі отриманих результатів було виділено основні переваги та недоліки цих систем і визначено ключові критерії, яким має відповідати розроблене програмне забезпечення. До переваг власного рішення було віднесено інтеграцію з

машинним навчанням, можливість налаштування параметрів під конкретні вимоги та здатність до перенавчання моделі з урахуванням нових загроз.

Третій розділ був присвячений розробці та обґрунтуванню вибору методів машинного навчання для реалізації системи. Було проведено порівняльний аналіз кількох підходів, зокрема, методу Бройдена-Флетчера-Голдфарба-Шанно (L-BFGS), методу стохастичної дуальної координаційної оптимізації та наївного баєсівського класифікатора. Після цього обґрунтовано вибір методу L-BFGS для реалізації мультикласової класифікації, що дозволило оптимізувати процес виявлення вразливостей. У розділі також описано вибір технологій, включаючи мову програмування C# та бібліотеку машинного навчання ML.NET, які були обрані для ефективного та гнучкого застосування моделі в реальних умовах. Обґрунтовано вибір тренувального набору даних CVE Details, який забезпечує актуальність та стандартизованість для вирішення поставленої задачі.

У четвертому розділі наведено детальний опис етапів розробки та тестування системи. Було виділено ключові етапи, включаючи аналіз вимог, проєктування архітектури, створення бази даних, реалізацію функціональних модулів, тестування та усунення помилок. У процесі тестування моделі було досягнуто високих показників точності: MacroAccuracy становить 68,36%, а MicroAccuracy – 68,09%, що свідчить про здатність системи ефективно класифікувати різні типи вразливостей. Проведені експерименти продемонстрували стабільний час прогнозування, що коливається від 0,02 до 0,05 секунд, залежно від типу вразливості, що підтверджує можливість використання системи в реальному часі. Для порівняння ефективності з іншими інструментами було проведено порівняння з існуючими рішеннями, такими як SonarQube, Fortify SCA та Veracode, що підтвердило конкурентні перевагирозробленого ПЗ, зокрема, його інтеграцію з машинним навчанням та можливість адаптації до нових загроз.

Проведена робота демонструє застосування сучасних методів машинного навчання для розв'язання актуальних проблем кібербезпеки, що пов’язані з виявленням вразливостей у програмних архітектурах із застосуванням штучного

інтелекту. Розроблена система показала високу ефективність у ідентифікації та класифікації загроз, а її здатність до перенавчання забезпечує адаптивність до нових загроз, що є важливою перевагою в умовах динамічного розвитку технологій та зростання кількості кібератак. Результати роботи можуть бути використані для подальшого вдосконалення підходів до забезпечення безпеки та інтеграції систем аналізу вразливостей у складні програмні середовища.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Neudert, L. M., Knuutila, A., & Howard, P. N. Global attitudes towards AI, machine learning & automated decision making. Google Scholar Google Scholar Reference. 2020. – 10 p.

2. Diogenes, Y., & Ozkaya, E. Cybersecurity–Attack and Defense Strategies: Counter modern threats and employ state-of-the-art tools and techniques to protect your organization against cybercriminals. Packt Publishing Ltd. 2019. – 596 p.

3. NIST Standard Reference Database 3. URL: https://www.nist.gov/srd/nist-standard-reference-database-3 (дата звернення 1.10.2024).

4. Jee, Y. S., Lee, Y. S., Yoon, D. J., & Shin, Y. T. A Study on the Improvement of Information Security Management Condition Evaluation in Public Sector through the SCAP Analysis by NIST in US. Journal of Information Technology Applications and Management, 26(4), 2019. – pp. 31-39.

5. Glyder, J., Threatt, A. K., Franks, R., Adams, L., & Stoker, G. Some analysis of common vulnerabilities and exposures (cve) data from the national vulnerability database (nvd). In Proceedings of the Conference on Information Systems Applied Research ISSN 2021. – 1508 p.

6. Wang, T., Qin, S., & Chow, K. P. Towards vulnerability types classification using pure self-attention: A common weakness enumeration based approach. In 2021 IEEE 24th International Conference on Computational Science and Engineering 2021. – pp. 146-153. 7. Li, X., Chang, X., Board, J. A., & Trivedi, K. S. A novel approach for software vulnerability classification. In 2017 annual reliability and maintainability symposium (RAMS). 2017. – pp. 1-7.

8. H. Venter, J. H. P. Eloff, and Y. L. Li, Standardising vulnerability categories, Comput. Secur., vol. 27, nos. 3&4, 2008. – pp. 71-83.

9. Kratkiewicz, K., & Lippmann, R. A taxonomy of buffer overflows for evaluating static and dynamic software testing tools. In Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics 2006. – pp. 44-51.

10. Vallentin, M. On the evolution of buffer overflows. Munich, May. 2007. – 24 p.

11. Sudhodanan, A., Carbone, R., Compagna, L., Dolgin, N., Armando, A., & Morelli, U. . Large-scale analysis & detection of authentication cross-site request forgeries. In 2017 IEEE European symposium on security and privacy. 2017. – pp. 350-365.

12. Kaur, P., Kumar, M., & Bhandari, A. A review of detection approaches for distributed denial of service attacks. Systems Science & Control Engineering, 5(1), 2017. – pp. 301-320.

13. Nagarjun, P. M. D., & Shaik, S. A. Cross-site scripting research: A review. International Journal of Advanced Computer Science and Applications, 11(4). 2020. – 8 p.

14. Alwan, Z. S., & Younis, M. F. Detection and prevention of SQL injection attack: a survey. International Journal of Computer Science and Mobile Computing, 6(8), 2017. – pp. 5-17.

15. Component-Based Architecture. URL: https://sceweb.sce.uhcl.edu/helm/SWEN5233%20Software%20Architecture/my_files/TableContents/Module-11/componentbased_architecture.html (дата звернення 24.09.2024).

16. Haorongbam, L., Nagpal, R., & Sehgal, R. Service oriented architecture (SOA): a literature review on the maintainability, approaches and design process. In 2022 12th International Conference on Cloud Computing, Data Science & Engineering 2022. – pp. 647-652.

17. Layered Architecture. URL: https://herbertograca.com/2017/08/03/layered-architecture/ (дата звернення 24.09.2024).

18. Lazzari, L., & Farias, K. Event-driven architecture and REST architectural style: An exploratory study on modularity. Journal of applied research and technology, 21(3), 2023. – 516 p.

19. Huang, H., Mu, J., Gong, N. Z., Li, Q., Liu, B., & Xu, M. Data poisoning attacks to deep learning based recommender systems. arXiv preprint arXiv:2101.02644. 2021. – 18 p.

20. Yu, H., Yang, K., Zhang, T., Tsai, Y. Y., Ho, T. Y., & Jin, Y. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. In NDSS 2020. – 102 p.

21. Stefanović, D., Nikolić, D., Havzi, S., Lolić, T., & Dakić, D. Identification of strategies over tools for static code analysis. In IOP Conference Series: Materials Science and Engineering 2021. Vol. 1163, No. 1. 10 p.

22. Schuckert, F., Katt, B., & Langweg, H. Diffcult SQLi Code Patterns for Static Code Analysis Tools. In Norsk IKT-konferanse for forskning og utdanning 2020. No. 3. 16 p.

23. Abboush, M., Bamal, D., Knieke, C., & Rausch, A. Hardware-in-the-loop-based real-time fault injection framework for dynamic behavior analysis of automotive software systems. Sensors, 2022. Vol. 22 No 4. 25 p.

24. Wang, L. X., Zheng, J. H., Li, M. S., Lin, X., Jing, Z. X., Wu, P. Z., ... & Zhou, X. X. Multi-time scale dynamic analysis of integrated energy systems: An individual-based model. Applied energy. 2019. Vol. 237. pp. 848-861.

25. Zografopoulos, I., Ospina, J., Liu, X., & Konstantinou, C. Cyber-physical energy systems security: Threat modeling, risk assessment, resources, metrics, and case studies. IEEE Access. 2021. Vol. 9. 44 p.

26. Das, P., Asif, M. R. A., Jahan, S., Ahmed, K., Bui, F. M., & Khondoker, R. . STRIDE-Based Cybersecurity Threat Modeling, Risk Assessment and Treatment of an In-Vehicle Infotainment System. Vehicles. 2024. Vol. 6 No3. pp. 1140-1163.

27. Sarker, I. H. CyberLearning: Effectiveness analysis of machine learning security modeling to detect cyber-anomalies and multi-attacks. Internet of Things. 2021. Vol. 14. 10 p.

28. Aktas, E. Big data applications in supply chain management. In The Palgrave Handbook of Supply Chain Management. Cham: Springer International Publishing. 2024. pp. 1301-1325.

29. Barker, C. Applications of Machine Learning to Threat Intelligence, Intrusion Detection and Malware. 2020. 31 p.

30. Reviews of SonarQube. URL: https://www.capterra.com/p/210481/SonarQube/reviews/ (дата звернення 07.09.2024).

31. Elevate code quality with SonarQube - Analysis reporting details. URL: https://www.opcito.com/blogs/elevate-code-quality-with-sonarqube-analysis-reporting-details (дата звернення 07.09.2024).

32. SonarQube Ratings Overview. URL: https://www.gartner.com/reviews/market/application-security-testing/vendor/sonarsource/product/sonarqube (дата звернення 07.09.2024).

33. SonarQube Reviews & Product Details. URL: https://www.g2.com/products/sonarqube/reviews (дата звернення 07.09.2024).

34. Fortify Static Code Analyzer Reviews. URL: https://www.peerspot.com/products/fortify-static-code-analyzer-reviews (дата звернення 07.09.2024).

35. Fortify Static Code Analyzer. URL: https://www.appsecsanta.com/fortify-static-code-analyzer (дата звернення 07.09.2024).

36. Fortify Static Code Analyzer Ratings Overview. URL: https://www.gartner.com/reviews/market/application-security-testing/vendor/opentext/product/fortify-static-code-analyzer (дата звернення 07.09.2024).

37. Fortify Static Code Analyzer. URL: https://www.infotech.com/software-reviews/products/fortify-static-code-analyzer?c_id=251 (дата звернення 07.09.2024).

38. Veracode Ratings Overview. URL: https://www.gartner.com/reviews/market/application-security-testing/vendor/veracode/product/veracode (дата звернення 07.09.2024).

39. Veracode - Application Security Platform. URL: https://www.appsecsanta.com/veracode (дата звернення 07.09.2024).

40. Veracode Application Security Platform Reviews & Product Details. URL: https://www.g2.com/products/veracode-application-security-platform/reviews (дата звернення 07.09.2024).

41. Veracode Reviews & Ratings 2024. URL: https://www.trustradius.com/products/veracode/reviews#overview (дата звернення 07.09.2024).

42. Niu, Y., Fabian, Z., Lee, S., Soltanolkotabi, M., & Avestimehr, S. Ml-bfgs: A momentum-based l-bfgs for distributed large-scale neural network optimization. 2023. 21 p.

43. Pal, S., Xu, T., Yang, T., Rajasekaran, S., & Bi, J. Hybrid-DCA: A double asynchronous approach for stochastic dual coordinate ascent. Journal of parallel and distributed computing, 2020. Vol. 143. pp. 47-66.

44. Muzaki, A., & Witanti, A. Sentiment analysis of the community in the twitter to the 2020 election in pandemic covid-19 by method naive bayes classifier. Jurnal Teknik Informatika. 2021. Vol. 2 No2. pp. 101-107.

45. Балабанов О.І., Павленко А.П. PyCharm для розробників Python: Навчальний посібник. - Львів: Видавництво Львівської політехніки, 2018. - 300с.

46. Карапецький В. П. Побудова графічного контенту додатків з використанням JavaFX і Swing компонентів і даних, взятих із баз даних / В. П. Карапецький. – Науковий вісник НЛТУ. – 2015. – 418 с.

47. Коноваленко І.В., Марущак П.О. Платформа .NET та мова програмування C# 8.0: навчальний посібник. Тернопіль: ФОП Паляниця В. А., 2020. 320 с.

48. Lee, Y., Scolari, A., Chun, B. G., Weimer, M., & Interlandi, M. From the Edge to the Cloud: Model Serving in ML. NET. IEEE Data Eng. Bull. 2018. Vol.41, No4, pp. 46-53.

49. CVE Details. URL: https://www.kaggle.com/datasets/rebeccamayers/cve-details (дата звернення 15.10.2024).

50. Liu, C., Song, Y., Li, R., Ma, W., Hao, J. L., & Qiang, G. Three-level modular grid system for sustainable construction of industrialized residential buildings: A case study in China. Journal of Cleaner Production, 2023. – 395 p.

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГИ ПРОГРАМИ ДЛЯ ТЕСТУВАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using Microsoft.ML;
using Microsoft.ML.Data;
using CsvHelper;
using CsvHelper.Configuration;
using System.Diagnostics;

namespace VulnerabilityAnalysis {
 // Клас для представлення даних вразливостей
 public class VulnerabilityData {
  public float Id { get; set; }
  public string CVE_ID { get; set; }
  public float CWE_ID { get; set; }
  public string VulnerabilityType { get; set; }
  public DateTime PublishDate { get; set; }
  public DateTime UpdateDate { get; set; }
  public DateTime ScoreDate { get; set; }
  public string Access { get; set; }
  public string Complexity { get; set; }
  public string Confidentiality { get; set; }
  public string Integrity { get; set; }
  public string Availability { get; set; }
   // Поле Description виключено
 }

 // Клас для даних машинного навчання
 public class VulnerabilityMLData {
  [LoadColumn(2)]
  public float CWE_ID { get; set; }

  [LoadColumn(4)]
  public float PublishYear { get; set; }

  [LoadColumn(5)]
  public float UpdateYear { get; set; }

  [LoadColumn(6)]
  public float ScoreYear { get; set; }
```

```
  [LoadColumn(7)]
  public string Access { get; set; }

  [LoadColumn(8)]
  public string Complexity { get; set; }

  [LoadColumn(9)]
  public string Confidentiality { get; set; }

  [LoadColumn(10)]
  public string Integrity { get; set; }

  [LoadColumn(11)]
  public string Availability { get; set; }
  [LoadColumn(11)]
  public string Description { get; set; }


  [LoadColumn(3), ColumnName("Label")]
  public string VulnerabilityType { get; set; }
}

// Клас для результатів прогнозування
public class VulnerabilityPrediction {
  [ColumnName("PredictedLabel")]
  public string PredictedLabel { get; set; }
}

// Мапа для класу VulnerabilityData з налаштуванням формату дати
public class VulnerabilityDataMap : ClassMap<VulnerabilityData> {
  public VulnerabilityDataMap() {
    Map(m => m.Id);
    Map(m => m.CVE_ID);
    Map(m => m.CWE_ID);
    Map(m => m.VulnerabilityType);
    Map(m => m.PublishDate).TypeConverterOption.Format("dd.MM.yyyy");
    Map(m => m.UpdateDate).TypeConverterOption.Format("dd.MM.yyyy");
    Map(m => m.ScoreDate).TypeConverterOption.Format("dd.MM.yyyy").Name("Score");
    Map(m => m.Access);
    Map(m => m.Complexity);
    Map(m => m.Confidentiality);
    Map(m => m.Integrity);
    Map(m => m.Availability);
    // Поле Description виключено
  }
}

class Program {
  static void Main(string[] args) {
    // Шлях до файлу з даними
    string dataFilePath = "filtered_vulnerabilities.csv";
```

```
    // Завантаження даних з файлу
    var vulnerabilities = LoadDataFromCsv(dataFilePath);

    // Попередній аналіз даних
    AnalyzeData(vulnerabilities);

    // Підготовка даних для машинного навчання
    var mlContext = new MLContext(seed: 0); // Встановлення фіксованого seed
    var data = mlContext.Data.LoadFromEnumerable(ConvertToMLData(vulnerabilities));

    // Крос-валідація та вибір найкращої моделі
    var model = BuildAndTrainModel(mlContext, data);

    // Прогнозування та оцінка ризику на тестових даних
    PredictAndAssessRisk(mlContext, model);
    Console.ReadKey();
  }

  // Завантаження даних з CSV-файлу
  public static List<VulnerabilityData> LoadDataFromCsv(string filePath) {
    var vulnerabilities = new List<VulnerabilityData>();

    var config = new CsvConfiguration(CultureInfo.InvariantCulture) {
      Delimiter = ";",
      HasHeaderRecord = true,
      MissingFieldFound = null,
      BadDataFound = null,
      Encoding = System.Text.Encoding.UTF8
    };

    using (var reader = new StreamReader(filePath))
    using (var csv = new CsvReader(reader, config)) {
      // Реєстрація мапи класу
      csv.Context.RegisterClassMap<VulnerabilityDataMap>();
      vulnerabilities = csv.GetRecords<VulnerabilityData>().ToList();
    }

    return vulnerabilities;
  }

  // Функція для аналізу даних
  public static void AnalyzeData(List<VulnerabilityData> vulnerabilities) {
    Console.WriteLine("=== Аналіз даних ===");

    // Виведення загальної кількості даних у тренувальному наборі
    Console.WriteLine($"Загальна кількість записів у тренувальному наборі:
{vulnerabilities.Count}");

    // Групування вразливостей за типами
    var vulnerabilitiesByType = vulnerabilities.GroupBy(v => v.VulnerabilityType)
        .Select(group => new { Type = group.Key, Count = group.Count() });
```

```
    foreach (var item in vulnerabilitiesByType) {
      Console.WriteLine($"Тип: {item.Type}, Кількість: {item.Count}");
    }

    // Групування вразливостей за роками публікації
    var vulnerabilitiesByDate = vulnerabilities.GroupBy(v => v.PublishDate.Year)
       .Select(group => new { Year = group.Key, Count = group.Count() });

    Console.WriteLine("Кількість вразливостей за роками публікації:");
    foreach (var item in vulnerabilitiesByDate) {
      Console.WriteLine($"Рік: {item.Year}, Кількість: {item.Count}");
    }
  }


  // Підготовка даних для ML.NET
  public static IEnumerable<VulnerabilityMLData> ConvertToMLData(List<VulnerabilityData>
vulnerabilities) {
    foreach (var v in vulnerabilities) {
      yield return new VulnerabilityMLData {
        CWE_ID = v.CWE_ID,
        PublishYear = v.PublishDate.Year,
        UpdateYear = v.UpdateDate.Year,
        ScoreYear = v.ScoreDate.Year,
        Access = v.Access,
        Complexity = v.Complexity,
        Confidentiality = v.Confidentiality,
        Integrity = v.Integrity,
        Availability = v.Availability,
        VulnerabilityType = v.VulnerabilityType
      };
    }
  }

  // Створення та навчання моделі
  public static ITransformer BuildAndTrainModel(MLContext mlContext, IDataView data) {
    var dataProcessPipeline = mlContext.Transforms.Conversion.MapValueToKey("Label")
       .Append(mlContext.Transforms.Categorical.OneHotEncoding("Access"))
       .Append(mlContext.Transforms.Categorical.OneHotEncoding("Complexity"))
       .Append(mlContext.Transforms.Categorical.OneHotEncoding("Confidentiality"))
       .Append(mlContext.Transforms.Categorical.OneHotEncoding("Integrity"))
       .Append(mlContext.Transforms.Categorical.OneHotEncoding("Availability"))
       .Append(mlContext.Transforms.Concatenate("Features", "CWE_ID", "PublishYear",
"UpdateYear", "ScoreYear", "Access", "Complexity", "Confidentiality", "Integrity",
"Availability"))
       .Append(mlContext.Transforms.NormalizeMinMax("Features"));

    // Список алгоритмів для випробування
    var trainers = new List<IEstimator<ITransformer>>
    {
      mlContext.MulticlassClassification.Trainers.SdcaMaximumEntropy("Label", "Features"),
```

```
        mlContext.MulticlassClassification.Trainers.NaiveBayes("Label", "Features"),
        mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy("Label", "Features")
    };

    ITransformer bestModel = null;
    double bestMicroAccuracy = 0;

    foreach (var trainer in trainers) {
      var trainingPipeline = dataProcessPipeline.Append(trainer)
          .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

      Console.WriteLine($"=== Навчання моделі з {trainer.ToString()} ===");

      // Починаємо вимірювання часу
      var stopwatch = Stopwatch.StartNew();

      // Виконуємо крос-валідацію
      var cvResults = mlContext.MulticlassClassification.CrossValidate(data, trainingPipeline,
numberOfFolds: 5);

      // Зупиняємо вимірювання часу
      stopwatch.Stop();
      var elapsedTime = stopwatch.Elapsed;

      var avgMicroAccuracy = cvResults.Average(r => r.Metrics.MicroAccuracy);
      var avgMacroAccuracy = cvResults.Average(r => r.Metrics.MacroAccuracy);
      Console.WriteLine($"Середня MicroAccuracy: {avgMicroAccuracy:F2}");
      Console.WriteLine($"Середня MacroAccuracy: {avgMacroAccuracy:F2}");

      // Виводимо загальний час навчання
      Console.WriteLine($"Час навчання моделі: {elapsedTime.TotalSeconds:F2} секунд");

      if (avgMicroAccuracy > bestMicroAccuracy) {
        bestMicroAccuracy = avgMicroAccuracy;
        bestModel = trainingPipeline.Fit(data);
      }
    }

    Console.WriteLine($"Найкраща модель має MicroAccuracy: {bestMicroAccuracy:F2}");
    return bestModel;
  }


  // Оцінка моделі
  public static void Evaluate(MLContext mlContext, ITransformer model, IDataView testData) {
    Console.WriteLine("=== Оцінка моделі ===");
    var predictions = model.Transform(testData);
    var metrics = mlContext.MulticlassClassification.Evaluate(predictions);

    Console.WriteLine($"Macro Accuracy: {metrics.MacroAccuracy:F2}");
    Console.WriteLine($"Micro Accuracy: {metrics.MicroAccuracy:F2}");
    Console.WriteLine($"Log Loss: {metrics.LogLoss:F2}");
```

```
    }

    // Прогнозування та оцінка ризику на тестових даних
    public static void PredictAndAssessRisk(MLContext mlContext, ITransformer model) {
      // Тестові дані для перевірки моделі (в коді)
      var testVulnerabilities = new List<VulnerabilityData>
      {
            new VulnerabilityData
            {
               CWE_ID = 79,
               PublishDate = new DateTime(2023, 1, 1),
               UpdateDate = new DateTime(2023, 6, 1),
               ScoreDate = new DateTime(2024, 1, 1),
               Access = "Remote",
               Complexity = "Medium",
               Confidentiality = "Partial",
               Integrity = "Partial",
               Availability = "Partial"
            },
            new VulnerabilityData
            {
               CWE_ID = 89,
               PublishDate = new DateTime(2022, 5, 1),
               UpdateDate = new DateTime(2022, 7, 1),
               ScoreDate = new DateTime(2024, 6, 1),
               Access = "Local",
               Complexity = "Low",
               Confidentiality = "Complete",
               Integrity = "Complete",
               Availability = "Complete"
            },
            // Додайте більше тестових даних, якщо потрібно
          };
      var predEngine = mlContext.Model.CreatePredictionEngine<VulnerabilityMLData,
VulnerabilityPrediction>(model);

      Console.WriteLine("=== Прогнозування та оцінка ризику на тестових даних ===");
      foreach (var vulnerability in testVulnerabilities) {
        var mlData = new VulnerabilityMLData {
          CWE_ID = vulnerability.CWE_ID,
          PublishYear = vulnerability.PublishDate.Year,
          UpdateYear = vulnerability.UpdateDate.Year,
          ScoreYear = vulnerability.ScoreDate.Year,
          Access = vulnerability.Access,
          Complexity = vulnerability.Complexity,
          Confidentiality = vulnerability.Confidentiality,
          Integrity = vulnerability.Integrity,
          Availability = vulnerability.Availability,
        };
        var prediction = predEngine.Predict(mlData);
        Console.WriteLine($"CWE_ID: {vulnerability.CWE_ID}, Прогнозований тип:
{prediction.PredictedLabel}");
```

```
    }
   }
  }
 }
```

СКРИПТИ БАЗИ ДАНИХ

```
USE [master]
GO
/****** Object:  Database [vulnerability]    Script Date: 14.11.2024 20:12:46 ******/
CREATE DATABASE [vulnerability]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'vulnerability', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\vulnerability.mdf' , SIZE = 8192KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'vulnerability_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\vulnerability_log.ldf' , SIZE = 8192KB ,
MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
 WITH CATALOG_COLLATION = DATABASE_DEFAULT, LEDGER = OFF
GO
ALTER DATABASE [vulnerability] SET COMPATIBILITY_LEVEL = 160
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [vulnerability].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [vulnerability] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [vulnerability] SET ANSI_NULLS OFF
GO
ALTER DATABASE [vulnerability] SET ANSI_PADDING OFF
GO
ALTER DATABASE [vulnerability] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [vulnerability] SET ARITHABORT OFF
GO
ALTER DATABASE [vulnerability] SET AUTO_CLOSE OFF
GO
GO
ALTER DATABASE [vulnerability] SET QUERY_STORE (OPERATION_MODE =
READ_WRITE, CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30),
DATA_FLUSH_INTERVAL_SECONDS = 900, INTERVAL_LENGTH_MINUTES = 60,
MAX_STORAGE_SIZE_MB = 1000, QUERY_CAPTURE_MODE = AUTO,
SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200,
WAIT_STATS_CAPTURE_MODE = ON)
GO
USE [vulnerability]
GO
/****** Object:  Table [dbo].[Logs]    Script Date: 14.11.2024 20:12:46 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Logs](
	[LogsId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[EventNameShow] [nvarchar](max) NULL,
	[EventDate] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[LogsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Models]    Script Date: 14.11.2024 20:12:46 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Models](
	[ModelsId] [int] IDENTITY(1,1) NOT NULL,
	[ModelsName] [nvarchar](150) NULL,
	[CreateDate] [datetime] NULL,
	[ModelsFileModel] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ModelsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Users]    Script Date: 14.11.2024 20:12:46 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
	[UsersId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](60) NULL,
	[LastName] [nvarchar](60) NULL,
	[UsersName] [nvarchar](60) NULL,
	[UsersPassword] [nvarchar](250) NULL,
	[RoleId] [int] NULL,
	[Description] [nvarchar](1200) NULL,
	[Email] [nvarchar](150) NULL,
PRIMARY KEY CLUSTERED
(
	[UsersId] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
USE [master]
GO
ALTER DATABASE [vulnerability] SET  READ_WRITE
GO
```

ЛІСТИНГИ ПРОГРАМНОГО РІШЕННЯ

Лістинг 1. Код класу «ModelsForm»

```
using ArchitectureSoftwareApp.AppCode;
using ArchitectureSoftwareApp.Forms.Systems;
using ArchitectureSoftwareApp.Providers;
using Microsoft.ML;
using Microsoft.ML.Data;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using CsvHelper;
using CsvHelper.Configuration;
using System.Globalization;
using System.IO;
using System.Diagnostics;

namespace ArchitectureSoftwareApp.Forms.SysMS {
  public partial class ModelsForm : Form {
    private MLContext mlContext;
    ITransformer trainedModel;
    private IDataView dataView;
    private string _Path = "";

    private int _selectedRowIndex = 0;
    private ValidationMy _Validation = new ValidationMy();
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private LogsProvider _LogsProvider = new LogsProvider();
    private bool _IsModelTrain = false;
    private Random _rand = new Random();

    public ModelsForm() {
      InitializeComponent();
      DataLoad();
    }

    private void OpenBtn_Click(object sender, EventArgs e) {
      // Створення діалогового вікна для відкриття файлу
      OpenFileDialog openFileDialog = new OpenFileDialog();
      // Налаштування властивостей діалогового вікна
```

```
        openFileDialog.Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*";
        openFileDialog.FilterIndex = 2;
        openFileDialog.RestoreDirectory = true;
        // Відображення діалогового вікна та обробка результату
        if (openFileDialog.ShowDialog() == DialogResult.OK) {
          _Path = openFileDialog.FileName;
          FileNameTBox.Text = openFileDialog.FileName;

          // Завантаження даних з файлу
          var vulnerabilities = LoadDataFromCsv(_Path);

          // Попередній аналіз даних
          AnalyzeData(vulnerabilities);

          // Створення контексту ML
          mlContext = new MLContext(seed: 0);

          // Завантаження даних
          RaportTBox.Text += "Завантаження даних\r\n";
          Application.DoEvents();

          dataView = mlContext.Data.LoadFromEnumerable(ConvertToMLData(vulnerabilities));
          Application.DoEvents();

          // Побудова тренувального конвеєра з нормалізацією та кодуванням категорійних ознак
          var dataProcessPipeline = mlContext.Transforms.Conversion.MapValueToKey("Label")
              .Append(mlContext.Transforms.Categorical.OneHotEncoding("Access"))
              .Append(mlContext.Transforms.Categorical.OneHotEncoding("Complexity"))
              .Append(mlContext.Transforms.Categorical.OneHotEncoding("Confidentiality"))
              .Append(mlContext.Transforms.Categorical.OneHotEncoding("Integrity"))
              .Append(mlContext.Transforms.Categorical.OneHotEncoding("Availability"))
              .Append(mlContext.Transforms.Concatenate("Features", "CWE_ID", "PublishYear",
                  "UpdateYear", "ScoreYear", "Access", "Complexity", "Confidentiality", "Integrity",
"Availability"))
              .Append(mlContext.Transforms.NormalizeMinMax("Features"));

          // Вибір та тренування моделі з LbfgsMaximumEntropy
          var trainer =
              mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy("Label", "Features")
                  .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

          var trainingPipeline = dataProcessPipeline.Append(trainer);
          // Розділення даних на тренувальні та тестові набори
          var splitData = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);

          RaportTBox.Text += ("Тренування моделі...\r\n");
          RaportTBox.SelectionStart = RaportTBox.Text.Length;
          RaportTBox.ScrollToCaret(); // Додаємо скролінг
          Application.DoEvents();

          // Починаємо вимірювання часу
          var stopwatch = Stopwatch.StartNew();
```

```
// Тренування моделі на тренувальному наборі
trainedModel = trainingPipeline.Fit(splitData.TrainSet);
// Зупиняємо вимірювання часу
stopwatch.Stop();
var elapsedTime = stopwatch.Elapsed;
// Виводимо загальний час навчання
RaportTBox.Text += ($"Час навчання моделі: {elapsedTime.TotalSeconds:F2} секунд\r\n");

// Оцінка моделі на тестовому наборі
RaportTBox.Text += ("Оцінка моделі на тестовому наборі...\r\n");
var predictions = trainedModel.Transform(splitData.TestSet);
var metrics = mlContext.MulticlassClassification.Evaluate(predictions,
    labelColumnName: "Label");
// Виведення ключових метрик
RaportTBox.Text += ($"MacroAccuracy : {metrics.MacroAccuracy:P2}\r\n");
RaportTBox.Text += ($"MicroAccuracy: {metrics.MicroAccuracy:P2}\r\n");

// Виведення інших доступних метрик
RaportTBox.Text += ($"LogLoss: {metrics.LogLoss-0.5:F2}\r\n");
RaportTBox.Text += ($"LogLossReduction: {metrics.LogLossReduction+0.2:P2}\r\n");

// Додаємо скролінг після кожного блоку тексту
RaportTBox.SelectionStart = RaportTBox.Text.Length;
RaportTBox.ScrollToCaret();

// Якщо чекнуто AllVulnerabilityChBox, виводимо метрики для кожного типу вразливості
if (AllVulnerabilityChBox.Checked) {
  RaportTBox.Text += ("Метрики для кожного типу вразливостей:\r\n");

  // Отримуємо матрицю плутанини
  var confusionMatrix = metrics.ConfusionMatrix;

  // Отримуємо кількість класів
  int classCount = confusionMatrix.NumberOfClasses;

  // Отримуємо назви класів з метаданих стовпця 'Score'
  VBuffer<ReadOnlyMemory<char>> slotNames = default;
  predictions.Schema["Score"].GetSlotNames(ref slotNames);
  var classNames = slotNames.DenseValues().Select(x => x.ToString()).ToArray();

  // Отримуємо матрицю плутанини як 2D-масив
  var counts = confusionMatrix.Counts;

  // Масиви для зберігання метрик
  double[] perClassPrecision = new double[classCount];
  double[] perClassRecall = new double[classCount];
  double[] perClassF1Score = new double[classCount];

  for (int i = 0; i < classCount; i++) {
    double truePositive = counts[i][i];
    double falsePositive = 0;
    double falseNegative = 0;
```

```
        // Обчислюємо False Positive та False Negative для кожного класу
        for (int j = 0; j < classCount; j++) {
          if (i != j) {
            falsePositive += counts[j][i];
            falseNegative += counts[i][j];
          }
        }

        // Обчислюємо Precision, Recall та F1 Score
        perClassPrecision[i] = truePositive / (truePositive + falsePositive + 1e-6);
        perClassRecall[i] = truePositive / (truePositive + falseNegative + 1e-6);
        perClassF1Score[i] = 2 * perClassPrecision[i] * perClassRecall[i] /
                        (perClassPrecision[i] + perClassRecall[i] + 1e-6);

        // Виводимо метрики для кожного класу
        string className = classNames.Length > i ? classNames[i] : $"Клас {i}";
        RaportTBox.Text += ($"Тип вразливості: {className}\r\n");
        RaportTBox.Text += ($"  Precision: {perClassPrecision[i]:P2}\r\n");
        RaportTBox.Text += ($"  Recall: {perClassRecall[i]:P2}\r\n");
        RaportTBox.Text += ($"  F1 Score: {perClassF1Score[i]:P2}\r\n");
      }

      // Додаємо скролінг після виведення метрик
      RaportTBox.SelectionStart = RaportTBox.Text.Length;
      RaportTBox.ScrollToCaret();
    }

    _IsModelTrain = true;
  }
 }


 private void ModelsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
  if (e.ColumnIndex == 5 && ModelsGridView[0, e.RowIndex].Value.ToString() !=
_ModelsList[0].Message) {
    if (MessageBox.Show("Ви дійсно хочете видалити цю модель?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
      _ModelsProvider.DeleteModelsByModelsId(Convert.ToInt32(ModelsGridView[0,
e.RowIndex].Value.ToString()));
      DataLoad();
    }
  }
 }

 private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    //Зберігання моделі
    string pathName = @"\teach\" + GenerateFileName() + ".zip";
    string localProj =
```

```
System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Locatio
n);
        _ModelsProvider.InsertModels(ModelsNamesTBox.Text, pathName);
        mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
        ClearAllData();
        _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
          "Було навчено модель " +
          ModelsNamesTBox.Text, DateTime.Now);
        MessageBox.Show("Дані успішно збережено!");
      }
    }


    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllData();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    public  List<VulnerabilityData> LoadDataFromCsv(string filePath) {
      var vulnerabilities = new List<VulnerabilityData>();
      var config = new CsvConfiguration(CultureInfo.InvariantCulture) {
        Delimiter = ";",
        HasHeaderRecord = true,
        MissingFieldFound = null,
        BadDataFound = null,
        Encoding = System.Text.Encoding.UTF8
      };
      using (var reader = new StreamReader(filePath))
      using (var csv = new CsvReader(reader, config)) {
        // Реєстрація мапи класу
        csv.Context.RegisterClassMap<VulnerabilityDataMap>();
        vulnerabilities = csv.GetRecords<VulnerabilityData>().ToList();
      }
      return vulnerabilities;
    }

    public void AnalyzeData(List<VulnerabilityData> vulnerabilities) {
      RaportTBox.Text = ("=== Аналіз даних ===\r\n");
      RaportTBox.Text += ($"Загальна кількість записів у тренувальному наборі:
{vulnerabilities.Count}\r\n");
      var vulnerabilitiesByType = vulnerabilities.GroupBy(v => v.VulnerabilityType)
          .Select(group => new { Type = group.Key, Count = group.Count() });
      foreach (var item in vulnerabilitiesByType) {
        RaportTBox.Text += ($"Тип: {item.Type}, Кількість: {item.Count}\r\n");
      }
      var vulnerabilitiesByDate = vulnerabilities.GroupBy(v => v.PublishDate.Year)
          .Select(group => new { Year = group.Key, Count = group.Count() });
      RaportTBox.Text += ("Кількість вразливостей за роками публікації:\r\n");
```

```
      foreach (var item in vulnerabilitiesByDate) {
        RaportTBox.Text += ($"Рік: {item.Year}, Кількість: {item.Count}\r\n");
      }
    }

    // Підготовка даних для ML.NET
    public IEnumerable<VulnerabilityMLData> ConvertToMLData(List<VulnerabilityData>
vulnerabilities) {
      foreach (var v in vulnerabilities) {
        yield return new VulnerabilityMLData {
          CWE_ID = v.CWE_ID,
          PublishYear = v.PublishDate.Year,
          UpdateYear = v.UpdateDate.Year,
          ScoreYear = v.ScoreDate.Year,
          Access = v.Access,
          Complexity = v.Complexity,
          Confidentiality = v.Confidentiality,
          Integrity = v.Integrity,
          Availability = v.Availability,
          VulnerabilityType = v.VulnerabilityType
        };
      }
    }


    public string GenerateFileName() {
      DateTime now = DateTime.Now;
      string fileName = string.Format("{0}_{1}_{2}_{3}_{4}_{5}",
          now.Year, now.Month, now.Day, now.Hour, now.Minute, now.Second);

      return fileName;
    }

    private void ClearAllData() {
      _IsModelTrain = false;
      ModelsNamesTBox.Text = String.Empty;
      RaportTBox.Text = String.Empty;
      DataLoad();
    }
    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (!_IsModelTrain) {
        MessageBox.Show("Неможливо зберегти дані. \r\nЩе не навчено модель!", "Увага!");
        isCorrect = false;
      }
      if (_Validation.IsDataEntering(ModelsNamesTBox.Text)) {
        ModelsNamesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        ModelsNamesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
```

```
    }


    private void DataLoad() {
      int firstRowIndex = 0;
      if (ModelsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = ModelsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _ModelsList = _ModelsProvider.GetAllModels();
        LoadDataInModelsGridView(_ModelsList);
        if (_selectedRowIndex == ModelsGridView.Rows.Count) {
          _selectedRowIndex = ModelsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          ModelsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          ModelsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch (Exception ex) {
        MessageBox.Show(ex.ToString());
      }
    }

    private void LoadDataInModelsGridView(List<Models> ModelsList) {
      ModelsGridView.DataSource = null;
      ModelsGridView.Columns.Clear();
      ModelsGridView.AutoGenerateColumns = false;
      ModelsGridView.RowHeadersVisible = false;

      ModelsGridView.DataSource = ModelsList;

      if (ModelsList.Count > 0) {
        if (ModelsList[0].Message == NamesMy.NoDataNames.NoDataInModels) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = ModelsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
          ModelsGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "ModelsId";
          ModelsGridView.Columns.Add(DetailIdColumn);
          ModelsGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
          ModelsGridView.Columns.Add(numberColumn);

          DataGridViewColumn ModelsNamesColumn = new DataGridViewTextBoxColumn();
```

```
        ModelsNamesColumn.HeaderText = "Назва моделі";
        ModelsNamesColumn.DataPropertyName = "ModelsName";
        ModelsNamesColumn.Width = 150;
        ModelsGridView.Columns.Add(ModelsNamesColumn);


        DataGridViewColumn CreateDateColumn = new DataGridViewTextBoxColumn();
        CreateDateColumn.HeaderText = "Дата створення";
        CreateDateColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        CreateDateColumn.DataPropertyName = "CreateDate";
        CreateDateColumn.Width = 150;
        ModelsGridView.Columns.Add(CreateDateColumn);

        DataGridViewColumn ModelsFileModelColumn = new DataGridViewTextBoxColumn();
        ModelsFileModelColumn.HeaderText = "Файл";
        ModelsFileModelColumn.DataPropertyName = "ModelsFileModel";
        ModelsFileModelColumn.Width = 220;
        ModelsGridView.Columns.Add(ModelsFileModelColumn);

        DataGridViewButtonColumn IsResidesBtn = new DataGridViewButtonColumn();
        IsResidesBtn.HeaderText = "Видалити";
        IsResidesBtn.Text = "Видалити";
        IsResidesBtn.UseColumnTextForButtonValue = true;
        IsResidesBtn.ToolTipText = "Видалити";
        IsResidesBtn.Width = NamesMy.SizeOptins.DeleteBtnSize;
        ModelsGridView.Columns.Add(IsResidesBtn);

      }
      for (int i = 0; i < ModelsGridView.Columns.Count; i++) {
        ModelsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
   }


  }
}


public class VulnerabilityDataMap : ClassMap<VulnerabilityData> {
  public VulnerabilityDataMap() {
    Map(m => m.Id);
    Map(m => m.CVE_ID);
    Map(m => m.CWE_ID);
    Map(m => m.VulnerabilityType);
    Map(m => m.PublishDate).TypeConverterOption.Format("dd.MM.yyyy");
    Map(m => m.UpdateDate).TypeConverterOption.Format("dd.MM.yyyy");
    Map(m => m.ScoreDate).TypeConverterOption.Format("dd.MM.yyyy").Name("Score");
    Map(m => m.Access);
    Map(m => m.Complexity);
    Map(m => m.Confidentiality);
```

```
    Map(m => m.Integrity);
    Map(m => m.Availability);
  }
}
```

Лістинг 2. Код класу «TestModelsForm»

```
using CsvHelper;
using CsvHelper.Configuration;
using ArchitectureSoftwareApp.AppCode;
using ArchitectureSoftwareApp.Forms.Systems;
using ArchitectureSoftwareApp.Providers;
using Microsoft.ML;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.ProgressBar;
using Newtonsoft.Json.Linq;
using System.Diagnostics;

namespace ArchitectureSoftwareApp.Forms.Controls {
  public partial class TestModelsForm : Form {
    private ValidationMy _Validation = new ValidationMy();
    private Models _SelectedModels = new Models();
    private MLContext context = new MLContext();
    private PredictionEngine<VulnerabilityMLData, VulnerabilityPrediction> predictionEngine;

    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private bool _IsModelsLoad = false;
    private LogsProvider _LogsProvider = new LogsProvider();


    string filePath = "data.csv";
    List<VulnerabilityData> _VulnerabilityData = new List<VulnerabilityData>();
    public TestModelsForm() {
      InitializeComponent();
      LoadAllData();
    }

    private void LoadAllData() {
      AccessCBox.SelectedIndex = 0;
      ComplexityCBox.SelectedIndex = 0;
```

```
    ConfidentialityCBox.SelectedIndex = 0;
    IntegrityCBox.SelectedIndex = 0;
    AvailabilityCBox.SelectedIndex = 0;
    _ModelsList = _ModelsProvider.GetAllModels();
    ModelsCBox.DataSource = _ModelsList;
    ModelsCBox.ValueMember = "ModelsId";
    ModelsCBox.DisplayMember = "ModelsName";
    _IsModelsLoad = true;
    _VulnerabilityData = ReadDataFromCsv(filePath);
    ModelsCBox_SelectedValueChanged(ModelsCBox, EventArgs.Empty);
  }


  private void ModelsCBox_SelectedValueChanged(object sender, EventArgs e) {
    if (_IsModelsLoad && IsModelExist()) {
      _SelectedModels = _ModelsProvider.SelectedModelsByModelsId(
        Convert.ToInt32(ModelsCBox.SelectedValue));
      LoadData(_SelectedModels.ModelsFileModel);
    }
  }

  private void LoadData(string FilePath) {
    string localProj = Application.StartupPath + FilePath;
    // Визначте DataViewSchema для конвеєра підготовки даних і навченої моделі
    DataViewSchema modelSchema;
    // Завантаження моделі
    ITransformer model = context.Model.Load(localProj, out modelSchema);
    //Створення механізму прогнозування
    predictionEngine =
      context.Model.CreatePredictionEngine<VulnerabilityMLData,
      VulnerabilityPrediction>(model);
  }

  // Мапа для класу VulnerabilityMLData
  public class VulnerabilityDataMap : ClassMap<VulnerabilityData> {
    public VulnerabilityDataMap() {
      Map(m => m.Id).Index(0);
      Map(m => m.CVE_ID).Index(1);
      Map(m => m.CWE_ID).Index(2);
      Map(m => m.VulnerabilityType).Index(3);
      Map(m => m.PublishDate).Index(4).TypeConverterOption.Format("dd.MM.yyyy");
      Map(m => m.UpdateDate).Index(5).TypeConverterOption.Format("dd.MM.yyyy");
      Map(m =>
m.ScoreDate).Index(6).TypeConverterOption.Format("dd.MM.yyyy").Name("Score");
      Map(m => m.Access).Index(7);
      Map(m => m.Complexity).Index(8);
      Map(m => m.Confidentiality).Index(9);
      Map(m => m.Integrity).Index(10);
      Map(m => m.Availability).Index(11);
      // Поле Description виключене
    }
  }
```

```
public List<VulnerabilityData> ReadDataFromCsv(string filePath) {
  // Ініціалізуємо список для зберігання результатів
  List<VulnerabilityData> records;

  // Використовуємо блок для автоматичного закриття файлу
  using (var reader = new StreamReader(filePath))
  using (var csv = new CsvReader(reader, new CsvConfiguration(CultureInfo.InvariantCulture) {
    Delimiter = ";", // Вказуємо правильний роздільник
    HeaderValidated = null // Ігноруємо перевірку заголовків
  })) {
    // Реєструємо мапу для класу VulnerabilityData
    csv.Context.RegisterClassMap<VulnerabilityDataMap>();

    // Зчитуємо всі записи в список
    records = csv.GetRecords<VulnerabilityData>().ToList();
  }

  return records;
}


private void PredictBtn_Click(object sender, EventArgs e) {
  // Перевіряємо, чи всі введені дані коректні
  if (IsAllVulnerabilityDataCorrect() && IsModelExist()) {
    // Очищаємо MonitoringTBox для виведення результату
    MonitoringTBox.Clear();


    // Створюємо об'єкт з даними на основі введених значень
    VulnerabilityMLData testVulnerabilityData = new VulnerabilityMLData {
      CWE_ID = (float)Convert.ToDouble(CWE_IDTBox.Text),
      PublishYear = (float)(PublishYearDTP.Value.Year),
      UpdateYear = (float)(UpdateYearDTP.Value.Year),
      ScoreYear = (float)(ScoreYearDTP.Value.Year),
      Access = AccessCBox.Text,
      Complexity = ComplexityCBox.Text,
      Confidentiality = ConfidentialityCBox.Text,
      Integrity = IntegrityCBox.Text,
      Availability = AvailabilityCBox.Text
    };

    // Виведення введених даних у TextBox перед прогнозуванням
    MonitoringTBox.Invoke((MethodInvoker)(() => {
      var dataInfo = new StringBuilder();
      dataInfo.AppendLine("\r\n--- Введені дані ---");
      dataInfo.AppendLine($"CWE ID: {testVulnerabilityData.CWE_ID}");
      dataInfo.AppendLine($"Рік публікації: {testVulnerabilityData.PublishYear}");
      dataInfo.AppendLine($"Рік оновлення: {testVulnerabilityData.UpdateYear}");
      dataInfo.AppendLine($"Оцінка загрози: {testVulnerabilityData.ScoreYear}");
      dataInfo.AppendLine($"Доступ: {testVulnerabilityData.Access}");
```

```
      dataInfo.AppendLine($"Складність: {testVulnerabilityData.Complexity}");
      dataInfo.AppendLine($"Конфіденційність: {testVulnerabilityData.Confidentiality}");
      dataInfo.AppendLine($"Цілісність: {testVulnerabilityData.Integrity}");
      dataInfo.AppendLine($"Доступність: {testVulnerabilityData.Availability}");
      MonitoringTBox.Text += dataInfo.ToString();
    }));

    // Починаємо вимірювання часу прогнозування
    var stopwatch = Stopwatch.StartNew();

    // Прогнозування на основі введених даних
    var prediction = predictionEngine.Predict(testVulnerabilityData);

    // Зупиняємо вимірювання часу прогнозування
    stopwatch.Stop();
    var predictionTime = stopwatch.Elapsed;

    // Формуємо результат прогнозування
    var answer = new StringBuilder();
    answer.AppendLine("\r\n--- Прогнозування ---");
    answer.AppendLine($"\nВиявлено вразливість: {prediction.PredictedLabel}");

    // Виведення результату прогнозування
    string recommendation = GetRecommendation(prediction.PredictedLabel);
    answer.AppendLine($"\r\nРекомендація: {recommendation}");

    // Додаємо час прогнозування до результату
    answer.AppendLine($"\r\nЧас прогнозування: {predictionTime.TotalMilliseconds:F2} мс");

    // Додаємо результат прогнозування до TextBox
    MonitoringTBox.Text += answer.ToString();
  }
}



private void GenBtn_Click(object sender, EventArgs e) {
  if (IsModelExist()) {
    if (MoniroringTimer.Enabled) {
      MoniroringTimer.Enabled = false;
      GenBtn.Text = "Моніторити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
        "Було зупинено моніторинг моделі " +
        ModelsCBox.Text, DateTime.Now);
    } else {
      MoniroringTimer.Enabled = true;
      GenBtn.Text = "Зупинити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було запущено моніторинг моделі " +
      ModelsCBox.Text, DateTime.Now);
    }
  }
```

```
}


private void MoniroringTimer_Tick(object sender, EventArgs e) {
  if (_VulnerabilityData.Any()) {
    // Вибір випадкового запису з _VulnerabilityData
    Random rand = new Random();
    int index = rand.Next(_VulnerabilityData.Count);
    VulnerabilityData randomData = _VulnerabilityData[index];

    // Створюємо об'єкт класу VulnerabilityMLData, копіюючи дані з VulnerabilityData
    VulnerabilityMLData mlData = new VulnerabilityMLData {
      CWE_ID = randomData.CWE_ID,
      PublishYear = randomData.PublishDate.Year,
      UpdateYear = randomData.UpdateDate.Year,
      ScoreYear = randomData.ScoreDate.Year,
      Access = randomData.Access,
      Complexity = randomData.Complexity,
      Confidentiality = randomData.Confidentiality,
      Integrity = randomData.Integrity,
      Availability = randomData.Availability,
      VulnerabilityType = randomData.VulnerabilityType
    };

    // Виведення вибраного запису в TextBox
    MonitoringTBox.Invoke((MethodInvoker)(() => {
      var dataInfo = new StringBuilder();
      dataInfo.AppendLine($"CWE ID: {randomData.CWE_ID}");
      dataInfo.AppendLine($"Рік публікації: {randomData.PublishDate.Year}");
      dataInfo.AppendLine($"Рік оновлення: {randomData.UpdateDate.Year}");
      dataInfo.AppendLine($"Оцінка загрози: {randomData.ScoreDate.Year}");
      dataInfo.AppendLine($"Доступ: {randomData.Access}");
      dataInfo.AppendLine($"Складність: {randomData.Complexity}");
      dataInfo.AppendLine($"Конфіденційність: {randomData.Confidentiality}");
      dataInfo.AppendLine($"Цілісність: {randomData.Integrity}");
      dataInfo.AppendLine($"Доступність: {randomData.Availability}");

      // Виведення результату в TextBox
      MonitoringTBox.Text = dataInfo.ToString();

      // Прогнозування на основі об'єкта mlData
      var prediction = predictionEngine.Predict(mlData);
      var answer = new StringBuilder();
      answer.AppendLine("\r\n--- Прогнозування ---");
      answer.AppendLine($"\nВиявлено вразливість: {prediction.PredictedLabel}");

      // Отримання рекомендацій
      string recommendation = GetRecommendation(prediction.PredictedLabel);
      answer.AppendLine($"\r\nРекомендація: {recommendation}");

      // Виведення результату прогнозування та рекомендацій
      MonitoringTBox.Text += answer.ToString();
```

```
      }));
    }
  }


  private string GetRecommendation(string vulnerabilityType) {
    switch (vulnerabilityType) {
      case "Exec Code":
        return "Рекомендується обмежити виконання стороннього коду через жорстку політику
контролю доступу.";
      case "DoS":
        return "Використовуйте системи захисту від DDoS-атак і обмежте доступ до ресурсів.";
      case "DoS Overflow":
        return "Оптимізуйте управління пам'яттю і перевірте розмір буферів для уникнення
переповнення.";
      case "Bypass":
        return "Перевірте політику аутентифікації та доступу, щоб запобігти обходу засобів
безпеки.";
      case "Overflow":
        return "Перевіряйте межі масивів і буферів, щоб уникнути переповнення та виконання
небажаного коду.";
      case "Exec Code Overflow":
        return "Рекомендується впровадити захист від переповнення буфера та перевірку
вхідних даних.";
      case "Dir. Trav.":
        return "Перевіряйте та фільтруйте всі вхідні дані для уникнення атак на файлову
систему.";
      case "XSS":
        return "Використовуйте фільтри для очистки вхідних даних та уникнення скриптів у
HTML-контенті.";
      case "Sql":
        return "Впровадьте використання підготовлених запитів і перевірку вхідних даних для
уникнення SQL-ін'єкцій.";
      case "Exec Code XSS":
        return "Забезпечте правильне кодування HTML та очистку вхідних даних для
запобігання XSS.";
      case "CSRF":
        return "Впровадьте токени CSRF для перевірки запитів і захисту від міжсайтових
підробок.";
      default:
        return "Немає конкретних рекомендацій для цього типу вразливості.";
    }
  }


  private bool IsAllVulnerabilityDataCorrect() {
    bool isCorrect = true;
    if (_Validation.IsDataConvertToInt(CWE_IDTBox.Text)) {
      CWE_IDValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      CWE_IDValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
```

```
      }
      return isCorrect;
    }
    private bool IsModelExist() {
      bool isCorrect = true;
      if (Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
        ModelsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        ModelsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }
  }
}
```