Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

студента *Мущака Артема Олександровича*

академічної групи *125-20-2*

спеціальності *125 Кібербезпека*

спеціалізації[1]

за освітньо-професійною програмою *Кібербезпека*

на тему ***Дослідження методів виявлення та аналізу вразливостей у мобільних додатках з метою підвищення їх безпеки***

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Проф. Гусєв О.Ю.			
розділів:				
спеціальний	Проф. Гусєв О.Ю.			
економічний	к. е. н., доц. Пілова Д.П.	90		

Рецензент				

Нормоконтролер	ст. викл. Мєшков В.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
безпеки інформації та телекомунікацій
_____ д.т.н., проф. Корнієнко В.І.

«_____»_____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавра

студенту	***Мущаку А.О.***	**академічної групи**	***125-20-2***
	(прізвище ім'я по-батькові)		(шифр)

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

на тему	***Дослідження методів виявлення та аналізу вразливостей у мобільних додатках з метою підвищення їх безпеки***

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 № 469-с

Розділ	**Зміст**	**Термін виконання**
Розділ 1	Аналіз та визначення проблем сучасного стану кібербезпеки мобільних додатків	22.05.2024
Розділ 2	Огляд методів виявлення вразливостей у мобільних додатках	18.06.2024
Розділ 3	ЕКОНОМІЧНИЙ РОЗДІЛ	25.06.2024

Завдання видано _____ Олександр Гусєв
(підпис керівника) (ім'я, прізвище)

Дата видачі: 15.01.2024р.

Дата подання до екзаменаційної комісії: 28.06.2024р.

Прийнято до виконання _____ Артем Мущак
(підпис студента) (ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 85 с., 10 рис., 1 табл., 5 додатка, 20 джерел.

Об'єкт дослідження: процеси виявлення вразливостей у мобільних додатках.

Мета роботи: розробка та аналіз методів виявлення вразливостей у мобільних додатках з використанням штучного інтелекту,його практичне застосування і інтеграція в існуючі системи.

Методи розробки: аналіз, моделювання, машинне навчання, тестування.

У першому розділі було проаналізовано сучасний стан кібербезпеки мобільних додатків, існуючі методи виявлення вразливостей, та визначено актуальність розробки нових підходів. У спеціальній частині було розроблено новий метод виявлення вразливостей на базі алгоритмів машинного навчання, описано технологічний стек та проведено порівняльний аналіз з існуючими методами.

В економічному розділі визначено економічну ефективність впровадження розробленого методу, проведено розрахунок витрат на розробку та потенційну економію від зменшення кіберзагроз.

Практичне значення роботи полягає у підвищенні рівня кібербезпеки мобільних додатків за рахунок впровадження ефективного методу виявлення вразливостей. Наукова новизна роботи полягає у розробці методу виявлення вразливостей, що базується на штучному інтелекті, з використанням новітніх алгоритмів машинного навчання.

КІБЕРБЕЗПЕКА, МОБІЛЬНІ ДОДАТКИ, ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ, АВТОМАТИЗАЦІЯ, АЛГОРИТМИ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Explanatory note: 85 p., 10 figures, 1 table, 4 appendices, 20 sources.

The object of research: vulnerability detection processes in mobile applications.

The purpose of the work: development and analysis of methods for detecting vulnerabilities in mobile applications using artificial intelligence, its practical application and integration into existing systems.Development methods: analysis, modeling, machine learning, testing.

The first chapter analyzed the current state of cyber security of mobile applications, existing methods of detecting vulnerabilities, and determined the relevance of developing new approaches. In the special part, a new method of detecting vulnerabilities based on machine learning algorithms was developed, the technological stack was described, and a comparative analysis with existing methods was carried out.

In the economic section, the economic efficiency of the implementation of the developed method is determined, the development costs and potential savings from reducing cyber threats are calculated.

The practical significance of the work consists in increasing the level of cyber security of mobile applications due to the implementation of an effective method of detecting vulnerabilities. The scientific novelty of the work consists in the development of a method of detecting vulnerabilities based on artificial intelligence, using the latest machine learning algorithms.

CYBER SECURITY, MOBILE APPLICATIONS, VULNERABILITY DETECTION, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, COST EFFICIENCY, AUTOMATION, ALGORITHMS, SOFTWARE TESTING.

СПИСОК УМОВНИХ СКОРОЧЕНЬ

ОС - Операційна система;

ШІ - Штучний інтелект;

API - Application Programming Interface;

CVE - Common List Of Vulnerabilities And Exposures;

DAST - Dynamic Code Analysis;

IAST - Interactive Security Analysis;

SAST - Static Code Analysis;

SDK - Software Development Kit;

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ 9
1.1 Аналіз сучасного стану кібербезпеки мобільних додатків 9
1.2 Визначення проблем виявлення та аналізу вразливостей 14
1.3 Постановка основних задач дослідження 20
1.4 Висновки до розділу 21
РОЗДІЛ 2. СПЕЦІАЛЬНА ЧАСТИНА 23
2.1 Дослідження методів виявлення вразливостей у мобільних додатках 23
2.1.1 Статичний аналіз коду 23
2.1.2 Динамічний аналіз 27
2.1.3 Аналіз трафіку 29
2.1.4 Фаззінг 32
2.1.5 Комбіновані методи 34
2.1.6 Порівняльний аналіз всих методів 36
2.1.7 Огляд недавніх досліджень та наукових робіт щодо ефективності методів виявлення вразливостей 36
2.2 Аналіз інструментів для аналізу безпеки мобільних додатків 39
2.3 Розробка власного методу виявлення та аналізу вразливостей на базі ШІ 42
2.3.1Вибір підходящих моделей ШІ для розробки методу 43
2.3.2 Розробка модулів 43
2.3.3 Процес роботи системи 46
2.3.4 Детальний опис технологічного стеку при побудові системи виявлення вразливостей 48
2.4 Практичне застосування методу 49
2.5 Висновки до розділу 53
РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ 54
3.1 Оцінка ефективності та вигідності використання запропонованих методів 54
3.2 Розрахунок можливого збитку від атаки на вузол або сегмент корпоративної мережі 62
3.3 Рекомендації щодо впровадження отриманих результатів в практику 64
3.4 Висновки до розділу 66

ВИСНОВКИ 68
ПЕРЕЛІК ПОСИЛАНЬ 70
ДОДАТОК А –Відомість матеріалів кваліфікаційної роботи 72
ДОДАТОК Б – Перелік документів на оптичному носії 73
ДОДАТОК В – Вихідний код моделі 74
ДОДАТОК Г – Відгук керівника економічного розділу 84
ДОДАТОК Д – Відгук керівника кваліфікаціоної роботи 85

ВСТУП

У сучасному інформаційному суспільстві зростає значущість кібербезпеки, особливо в контексті поширення мобільних технологій та збільшення кількості мобільних додатків. Забезпечення безпеки цих додатків є надзвичайно важливим завданням, оскільки вони використовуються для обробки та зберігання конфіденційної інформації та мають доступ до різноманітних сервісів та ресурсів.

Обрана тема дослідження про кібербезпеку мобільних додатків є актуальною і важливою у зв'язку зі зростанням кількості кіберзагроз та кількістю користувачів мобільних пристроїв. Фахівці з кібербезпеки мають великий інтерес до цієї проблеми, оскільки вона пов'язана з їхнім об'єктом діяльності та спеціальністю.

На сьогоднішній день проблема кібербезпеки мобільних додатків залишається актуальною через постійне зростання кількості загроз та недоліків у цій області. Аналіз сучасного стану показує, що існує низка проблем, таких як недостатня захищеність даних, небезпечні дії користувачів та вразливості програмного забезпечення. Існуючі методи виявлення вразливостей також мають свої обмеження, що підкреслює потребу в розробці нових підходів.

Метою даної кваліфікаційної роботи є дослідження методів виявлення та аналізу вразливостей у мобільних додатках з метою підвищення їхньої безпеки. Це дозволить не лише розробити новий метод, але й застосувати його для практичного використання з метою запобігання кібератакам та забезпечення безпеки користувачів.

Отримані результати матимуть як теоретичну, так і практичну значущість. Теоретично вони сприятимуть розширенню знань у галузі кібербезпеки мобільних додатків, а практично - допоможуть вдосконалити методи захисту та забезпечити безпеку користувачів мобільних пристроїв.

РОЗДІЛ 1 СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз сучасного стану кібербезпеки мобільних додатків

У зв'язку з постійним розвитком технологій та зростанням використання мобільних додатків, питання кібербезпеки стає дедалі більш актуальним. Мобільні додатки, незважаючи на їхню популярність та корисність, часто стають об'єктом кібератак та зловмисних дій.

На сьогоднішній день існує ряд загроз та вразливостей, які ставлять під загрозу безпеку мобільних додатків. Серед найбільш поширених можна виділити:

a) недостатня захищеність даних - багато додатків не належним чином захищають конфіденційні дані користувачів, що може призвести до їхнього витоку [1];

b) небезпечні дії користувачів - деякі додатки можуть змушувати користувачів виконувати небезпечні дії, такі як встановлення шкідливих додатків або надання доступу до приватних даних;

c) вразливості безпеки програмного забезпечення - помилки в програмному забезпеченні додатків можуть призвести до вразливостей, через які зловмисники можуть отримати доступ до системи. На рис. 1.1. зображена ілюстрація, що відображає статистику вразливостей програмного забезпечення мобільних додатків протягом останніх років в різних ОС [2-3]. Цей графік підкреслює зростання загроз і також ілюструє порівння того що згідно з даними CVE, у 2023 році Android мав найбільше вразливостей серед усіх операційних систем. У цьому випадку вразливості визначаються як помилки в програмному забезпеченні, які можуть бути безпосередньо використані хакером для отримання доступу до системи або мережі. У 2023 році в Android було виявлено 523 таких

вразливості, що значно випереджає 161 в iOS. Хоча попередній рік мав іншу статистику. Тоді в Android було лише 125 вразливостей, а в iOS – 387;

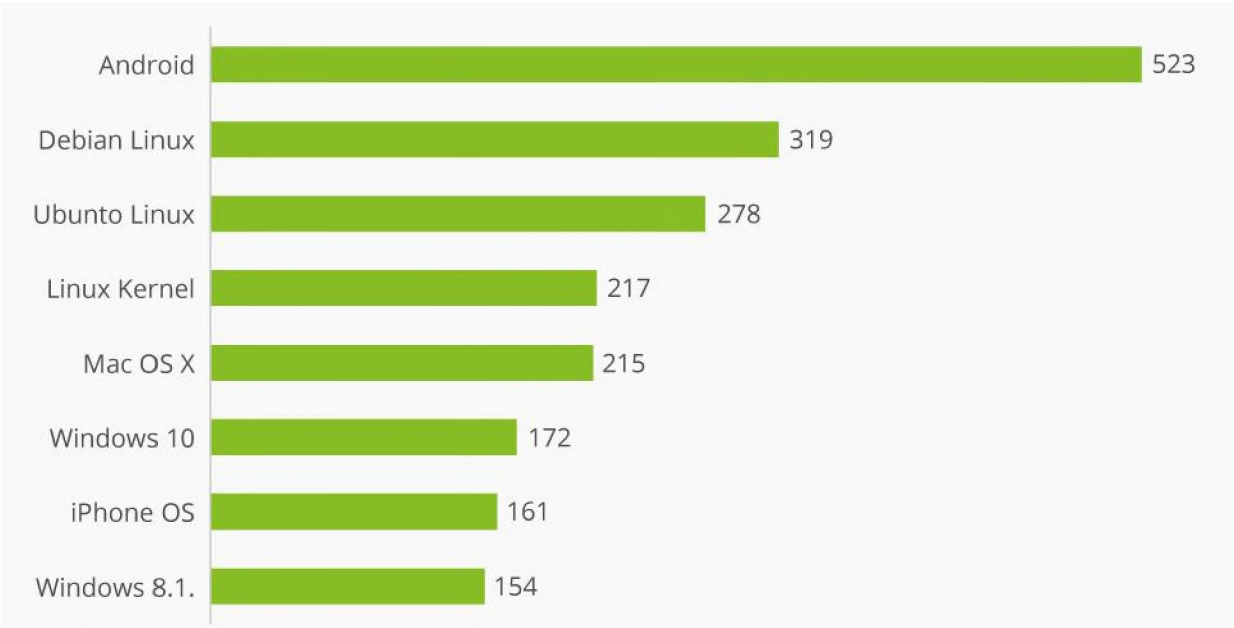

Рисунок 1.1 – Графік вразливостей операційних систем

d) зламані або підроблені додатки - зловмисники можуть створювати підроблені версії популярних додатків або інтегрувати шкідливий код у легітимні додатки. Користувачі, які завантажують ці додатки, можуть стати жертвами крадіжок даних або інших шкідливих атак. Одним із таких прикладів у 2023 році це було виявлено підроблену версію додатка TikTok для Android, яка містила шкідливий код, призначений для крадіжки особистих даних користувачів. Ця підроблена версія поширилася через сторонні додатки і загрожувала безпеці користувачів;

e) використання незахищених API - мобільні додатки можуть використовувати API (інтерфейси програмування застосунків), які не захищені від несанкціонованого доступу або атак. Якщо API не належним чином захищені, зловмисники можуть отримати доступ до конфіденційних даних або виконати шкідливі операції через мобільні додатки;

є) недостатня аутентифікація та авторизація - деякі мобільні додатки можуть мати недоліки в механізмах аутентифікації та авторизації, що може

призвести до можливості несанкціонованого доступу до облікових записів користувачів або обмежених функцій додатка. Наприклад, у 2019 році було виявлено, що мобільний додаток Facebook для iOS не вимагав повної перевірки автентичності під час авторизації, що дозволяло зловмисникам отримувати доступ до облікових записів користувачів без їхнього відома;

ж) використання шкідливих рекламних SDK - деякі рекламні SDK (набори розробки програм) можуть містити шкідливий код, який може виконувати шкідливі дії на пристрої користувача, такі як крадіжка даних або виведення на обман;

з) соціальна інженерія та фішингові атаки - зловмисники можуть використовувати соціальну інженерію для обману користувачів та отримання їхніх облікових даних або особистої інформації через мобільні додатки;

е) несанкціоновані точки доступу Wi-Fi - це коли хакери маскуються під безкоштовну публічну мережу Wi-Fi і заманюють користувачів підключатися до неї, щоб викрасти їхні особисті дані. Однак, після підключення до цих фальшивих мереж, зловмисники можуть зламати пристрій користувача або використовувати спеціальні програми для перехоплення трафіку та викрадення особистих даних. Детальний процес дії може бути наступним:

1) Створення фальшивої точки доступу - хакер створює фальшиву Wi-Fi мережу, яка маскується під легітимну публічну мережу, наприклад, «Free Cafe Wi-Fi» або «Hotel Guest Wi-Fi»;

2) Заманювання користувачів - хакери розміщуються в публічних місцях, де очікуються підключення до Wi-Fi, таких як кафе, аеропорти або готелі. Вони намагаються привернути увагу користувачів, пропонуючи безкоштовний доступ до інтернету;

3) Перехоплення трафіку і викрадення даних - після підключення користувача до фальшивої мережі, хакери можуть перехоплювати весь трафік,

що передається через цю мережу. Це дозволяє їм отримати доступ до особистих даних, таких як паролі, номери кредитних карток і особисті повідомлення.

4) Злам пристрою - деякі атаки можуть включати в себе спроби злому пристрою користувача після підключення до фальшивої мережі. Наприклад, хакери можуть спробувати використати вразливості в операційній системі або додатках для отримання додаткового доступу.

Згідно з даними досліджень, більше половини мобільних додатків мають проблеми з безпекою. Наприклад, у 2023 році було виявлено, що 62,7% мобільних додатків містили принаймні одну серйозну вразливість. Прикладами таких вразливостей можуть бути недостатня аутентифікація користувачів, недостатня захист від атак переповнення буфера або недостатня перевірка даних, що надходять від користувачів.

Обов'язково при аналізі вразливостей треба розглянути конкретні ситуації які сталися протягом останніх років, а саме можна розглянути наступні:

a) Атаки через SMS або месенджери - розсилка шкідливих посилань через SMS або месенджери, що привели до інсталяції шкідливого ПЗ або фішингових сайтів які відбулися у 2023 році шляхом надсилання таких через оператора мобільного зв’язку LifeCell;

b) Витік даних у Clubhouse. У квітні 2021 року виявлено вразливість у популярному аудіо-додатку Clubhouse, яка полягала у витоку конфіденційних даних користувачів через недолік у системі захисту. Аналіз показав, що зловмисники могли використати цю вразливість для отримання доступу до аудіо-потоків без відома користувачів. Це викликало серйозні обурення серед користувачів та підкреслило необхідність посилення кібербезпеки в мобільних додатках;

c) Атака на Microsoft Exchange Server, вплив на мобільний додаток Outlook. У березні 2021 року виявлено серйозну вразливість у сервері електронної пошти Microsoft Exchange, яка дозволяла зловмисникам віддалено

виконувати код на сервері та отримувати доступ до конфіденційних даних. Ця атака стала причиною великого числа компрометацій та витоку даних у різних організаціях. Подальші аналізи показали, що вразливість також вплинула на мобільні додатки Outlook для iOS та Android, що збільшило ризик для користувачів, які використовують ці додатки для робочої електронної пошти. Схема хакерської атаки цього випадку зображена на рискунку 1.2;

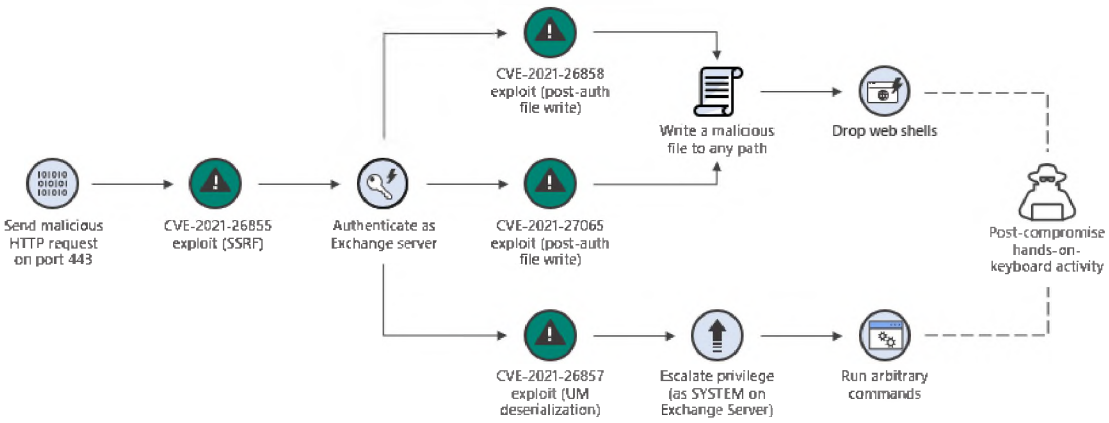

Рисунок 1.2 – Ланцюжок експлойтів на сервері Exchange Server

d) Вразливість у WhatsApp. У травні 2022 року було виявлено вразливість у месенджері WhatsApp, яка дозволяла зловмисникам віддалено відкривати спеціально сформовані відомості та виконувати код на вразливому пристрої. Ця вразливість могла бути використана для викрадення особистих даних користувачів, включаючи повідомлення та медіафайли. Цей інцидент підкреслив необхідність посилення захисту особистих даних у мобільних додатках та постійного моніторингу наявних вразливостей. Також можна зазначити що у період воєнних дій цей додаток не одноразово використовується країною терористом для хакерських дій, в наслідок чого отримують важливі дані для своїх потреб [4-5].

Ці приклади демонструють, які реальні наслідки можуть мати вразливості мобільних додатків та як вони можуть впливати на безпеку та конфіденційність користувачів.

Аналіз сучасного стану кібербезпеки мобільних додатків показує, що існує необхідність в розробці нових методів та інструментів для виявлення та аналізу вразливостей з метою підвищення рівня безпеки цих додатків. У наступних підрозділах будуть розглянуті проблеми виявлення та аналізу вразливостей у мобільних додатках та поставлені основні завдання дослідження.

1.2 Визначення проблем виявлення та аналізу вразливостей

Аналіз сучасних методів виявлення вразливостей у мобільних додатках є важливим кроком у розробці нових підходів до забезпечення безпеки. Для цього потрібно ретельно дослідити існуючі методи та ідентифікувати їхні недоліки та обмеження.

Існуючі методи виявлення вразливостей у мобільних додатках:

a) Статичний аналіз коду - цей метод включає в себе аналіз вихідного коду мобільного додатка без його виконання [6-7]. Під час статичного аналізу програмного коду розглядаються самі файли програмного коду для виявлення потенційних вразливостей. Це може включати перевірку на використання небезпечних функцій або недостатню обробку введених даних. Наприклад, аналізатори програмного коду можуть виявити в коді додатка використання ненадійних функцій для обробки паролів або виконання системних команд, що може призвести до вразливостей;

b) Динамічний аналіз - під час динамічного аналізу додаток виконується у контрольованому середовищі, де здійснюється моніторинг його поведінки. Цей метод дозволяє виявити вразливості, які можуть бути активовані лише під час виконання програми. Наприклад, під час динамічного аналізу можуть виявитися вразливості, пов'язані з взаємодією з вищим рівнем привілеїв, які можуть бути використані для отримання несанкціонованого доступу до даних або ресурсів;

c) Аналіз вразливостей на рівні мережі - цей метод полягає в моніторингу трафіку між мобільним додатком і сервером для виявлення потенційно шкідливих дій або витоку конфіденційної інформації. Аналізатори можуть виявити спроби передачі конфіденційної інформації у відкритому вигляді, відправку даних до ненадійних джерел або інші підозрілі дії під час мережевої взаємодії. Наприклад, можуть виявитися спроби надсилання паролів чи інших конфіденційних даних у незахищеному вигляді через мережу;

d) Пентестінг (тестування на проникнення) - цей метод включає запуск реальних атак на мобільний додаток для виявлення вразливостей та оцінки рівня його безпеки. Тестери можуть використовувати різноманітні інструменти та техніки, які використовуються зловмисниками, для проведення атак на додаток. Пентестінг може бути ручним або автоматизованим;

e) Аналіз внутрішнього предметного середовища (SAST) - цей метод включає в себе аналіз вихідного коду програми для виявлення вразливостей та помилок безпеки на ранніх етапах розробки. Аналізатори програмного коду використовуються для пошуку можливих проблем в коді, таких як недостатнє використання криптографічних алгоритмів, вразливості SQL-ін'єкцій, використання ненадійних функцій тощо;

f) Аналіз поведінки (моніторинг поведінки додатка) - цей метод включає в себе відстеження та аналіз поведінки мобільного додатка в реальному часі для виявлення незвичайних або підозрілих дій. Зазвичай цей аналіз використовується для виявлення шкідливих або несанкціонованих дій додатка під час його виконання;

g) Реверс-інжиніринг - цей метод включає декомпіляцію додатків для аналізу байт-коду (наприклад, Java байт-коду для Android додатків). Це дозволяє виявити приховані вразливості, небезпечні функції та потенційно шкідливий код. Схема якого можна побачити на рис. 1.3.

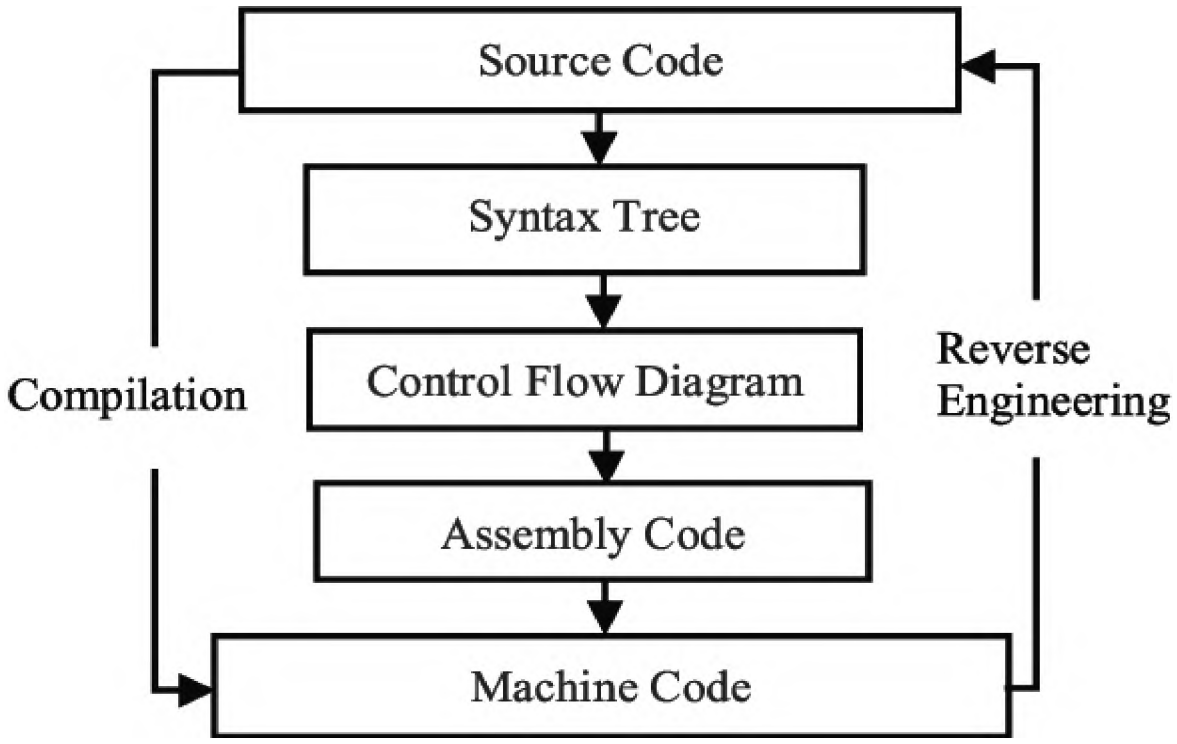

Рисунок 1.3 - Блок-схема процесу реінжинірингу

Ці методи аналізу використовуються для виявлення та виправлення вразливостей у мобільних додатках з метою забезпечення їх безпеки та надійності. Кожен з них має свої переваги і недоліки, тому їх комбінація може бути ефективним підходом до забезпечення безпеки програмного забезпечення.

Комбінація цих методів дозволяє забезпечити більш повну та ефективну безпеку мобільних додатків, сприяючи вчасному виявленню та усуненню вразливостей.

Але крім методів також є інструменти та платформм для виявлення вразливостей у мобільних додатках. Деякі з них включають:

a) «OWASP ZAP (Zed Attack Proxy)» - відкритий інструмент для автоматичного тестування веб-додатків на наявність вразливостей;

b) «MobSF (Mobile Security Framework)» - всебічний інструмент для безпеки мобільних додатків, який підтримує як статичний, так і динамічний аналіз Android та iOS додатків;

c) «Drozer» - інструмент для тестування безпеки Android додатків, який дозволяє шукати вразливості в додатках та операційній системі.

d) «Frida» - динамічний інструментарій для інжектування коду, який дозволяє дослідникам з безпеки взаємодіяти з виконанням програм на рівні машинного коду;

e) - «Appium» - автоматизований інструмент для тестування мобільних додатків, який може використовуватися для виявлення проблем з безпекою через автоматизовані тестові сценарії.

Можна підсумувати що, регулярне тестування та аналіз безпеки мобільних додатків є критично важливими для виявлення та усунення вразливостей перед тим, як зловмисники зможуть їх використати. Це включає в себе не тільки використання автоматизованих інструментів, але й мануальне рев'ю коду та тестування, щоб забезпечити всебічний підхід до безпеки мобільних додатків.

Аналіз методів виявлення вразливостей у програмному забезпеченні виявляє ряд недоліків та обмежень, які можуть впливати на ефективність та точність процесу виявлення. Детальний розгляд цих обмежень дозволяє краще зрозуміти потенційні виклики, з якими можуть зіткнутися дослідники та розробники при забезпеченні безпеки програмного забезпечення, тому основними недоліками та обмеженнями можна визначити наступні:

a) Неявність повного покриття - одним з основних обмежень існуючих методів виявлення вразливостей є неможливість забезпечення повного покриття всіх потенційних вразливостей. Це обмеження особливо актуальне для статичного аналізу коду, який базується на перевірці вихідного коду програми без її виконання. Хоча статичний аналіз може ефективно ідентифікувати широкий спектр вразливостей, він може не виявити ті, що проявляються лише під

час динамічного виконання програми, наприклад, через складні взаємодії між компонентами або через специфічні умови виконання;

b) Високий рівень фальс-позитивів - іншою значною проблемою є високий рівень фальс-позитивних результатів, які генерують деякі методи виявлення вразливостей. Фальс-позитивні результати виникають, коли метод помилково ідентифікує безпечний код як вразливий. Це може призвести до значних витрат часу та ресурсів на перевірку та аналіз цих помилкових спрацьовувань, відволікаючи увагу від реальних загроз. Високий рівень фальс-позитивів може також знизити довіру до інструментів виявлення вразливостей, обтяжуючи процес виправлення та управління вразливостями;

c) Велика кількість фальш-спрацювань - динамічний аналіз, який включає виконання програми в контрольованому середовищі для ідентифікації вразливостей, також схильний до генерації значної кількості ложних спрацьовувань. Це особливо актуально для складних додатків або додатків, що виконують велику кількість динамічних дій.

d) Фальш спрацьовування - виникають, коли динамічний аналіз інтерпретує легітимну поведінку програми як потенційно шкідливу або вразливу. Це може бути пов'язано зі складністю аналізу контексту виконання програми, де динамічні особливості, такі як взаємодія з користувачем, зовнішніми системами або мережевими запитами, можуть значно варіюватися. В результаті, динамічний аналіз може помилково класифікувати нормальну поведінку як підозрілу або небезпечну, що призводить до необхідності додаткового аналізу та перевірки з боку спеціалістів з безпеки.

Аналізуючі ці методи можна сказати що є шляхи вдосконалення. Обмеження існуючих методів виявлення вразливостей підкреслюють важливість розробки більш точних та ефективних інструментів та методик. Для подолання цих викликів, можуть бути використані наступні підходи:

a) Поєднання статичного та динамічного аналізу - інтеграція результатів статичного та динамічного аналізу може допомогти забезпечити більш повне покриття потенційних вразливостей, знижуючи при цьому кількість фальс-позитивних та ложних спрацьовувань;

b) Розвиток технологій машинного навчання - використання алгоритмів машинного навчання для аналізу великих обсягів даних про вразливості може допомогти покращити точність виявлення, зменшуючи кількість фальс-позитивних результатів;

c) Розробка спеціалізованих інструментів для конкретних мов програмування та платформ - створення інструментів, оптимізованих для специфіки конкретних мов програмування або платформ, може підвищити ефективність виявлення вразливостей, зменшуючи кількість невідповідностей та помилок;

d) Активне залучення спільноти - співпраця з дослідницькою спільнотою та використання відкритих баз даних про вразливості може допомогти в ідентифікації нових та невідомих вразливостей, покращуючи загальну ефективність процесів виявлення та виправлення.

Враховуючи ці обмеження, важливо розуміти, що жоден інструмент або методика не може гарантувати 100% виявлення всіх вразливостей. Тому, інтегрований підхід, який включає різноманітні методи та інструменти, є ключовим для підвищення рівня безпеки програмного забезпечення. Крім того, постійне оновлення знань та інструментів, а також адаптація до нових загроз і технологій, є необхідними для ефективного захисту від потенційних вразливостей.

Освітні програми та тренінги для розробників та спеціалістів з безпеки також відіграють важливу роль у підвищенні загального рівня безпеки, оскільки вони допомагають краще розуміти потенційні ризики та методи їх усунення. В кінцевому підсумку, зусилля по забезпеченню безпеки програмного забезпечення

мають бути постійними та всебічними, включаючи як технічні, так і організаційні аспекти. Але основну увагу приділимо технологіям машинного навчання.

1.3 Постановка основних задач дослідження

У контексті кваліфікаційої роботи на тему «Дослідження методів виявлення та аналізу вразливостей у мобільних додатках з метою підвищення їхньої безпеки», основні завдання дослідження орієнтовані на аналіз існуючих методів та їх покращення та розробку і валідацію новітнього методу, що інтегрує можливості штучного інтелекту для ефективного виявлення та аналізу вразливостей. Цей метод передбачає використання спеціалізованого алгоритму та навченої моделі, здатних ідентифікувати потенційні загрози та вразливості в коді мобільних додатків.

Основні завдання дослідження:

a) Аналіз інснуючих методів - проаналізувати існуючі методи захисту та аналізу і сформулювати методиційні покращення цих методів;

b) Розробка методу на базі штучного інтелекту - сформулювати та розробити інноваційний метод виявлення вразливостей, який використовує алгоритми штучного інтелекту та машинного навчання для аналізу коду мобільних додатків;

c) Навчання та тестування моделі - провести навчання моделі штучного інтелекту на основі доступних датасетів, що містять дані про відомі вразливості. Валідація ефективності моделі має бути здійснена через тестові сценарії в емуляторах або на реальних тестових даних [8-9];

d) Інтеграція та адаптація методу - розробити механізми для інтеграції розробленого методу в існуючі процеси розробки та аудиту мобільних додатків, забезпечуючи можливість його широкого використання для підвищення безпеки.

Ці завдання та цілі передбачають комплексний підхід до виявлення та аналізу вразливостей у мобільних додатках, використовуючи передові технології штучного інтелекту. Реалізація цього підходу не лише підвищить ефективність ідентифікації вразливостей, але й сприятиме розвитку нових стандартів безпеки в галузі мобільних додатків.

1.4 Висновки до розділу

У цьому розділі було сформульовано ключові завдання та цілі дослідження, спрямовані на розробку та впровадження новітнього методу виявлення та аналізу вразливостей у мобільних додатках, заснованого на технологіях штучного інтелекту. Основна ідея полягає у створенні інноваційного підходу, який використовує алгоритми машинного навчання та навчену модель для ефективного ідентифікування потенційних загроз та вразливостей. Цей метод передбачає можливість інтеграції в існуючі процеси розробки та аудиту, забезпечуючи широке його застосування для підвищення рівня безпеки мобільних додатків.

Важливим аспектом є навчання та тестування моделі на основі реальних даних, що дозволить оцінити ефективність методу та його придатність для виявлення різноманітних типів вразливостей. Адаптація та інтеграція розробленого методу в процеси розробки та аудиту мобільних додатків відкриває нові можливості для зміцнення захисту від кіберзагроз та підвищення загального рівня кібербезпеки.

Таким чином, розробка та впровадження методу виявлення вразливостей на базі штучного інтелекту є актуальним та перспективним напрямком дослідження, який може забезпечити значний прогрес у підвищенні безпеки мобільних додатків. Реалізація цього підходу вимагає подальшого детального

аналізу, розробки, тестування та адаптації, але потенційні переваги від його застосування є значними для всієї галузі мобільних технологій.

РОЗДІЛ 2. СПЕЦІАЛЬНА ЧАСТИНА

2.1 Дослідження методів виявлення вразливостей у мобільних додатках

У сучасному світі, де мобільні технології стають все більш інтегрованими в наше повсякденне життя, безпека мобільних додатків набуває особливої актуальності. З кожним роком кількість мобільних загроз зростає, що вимагає від розробників і аналітиків безпеки більш ретельного підходу до захисту даних користувачів. Вразливості в мобільних додатках можуть призвести до несанкціонованого доступу, витоку персональних даних, фінансових втрат, а також зниження довіри користувачів до продукта [10].

З огляду на це, виявлення та усунення вразливостей у мобільних додатках є критично важливими для забезпечення конфіденційності, цілісності та доступності користувацьких даних. Цей підрозділ присвячений огляду існуючих методів виявлення вразливостей, їхніх переваг та недоліків, а також аналізу їх ефективності в різних сценаріях використання. Мета цього огляду — не тільки описати існуючі підходи, але й визначити потенційні напрямки для покращення інструментів та методик, що сприятиме підвищенню загальної безпеки мобільних додатків.

2.1.1 Статичний аналіз коду

Статичний аналіз коду - це процес аналізу джерельного коду або скомпільованого виконуваного коду без фактичного виконання програми. Цей метод використовується для виявлення помилок, вразливостей або недотримання стандартів кодування ще до тестування або запуску програми [11]. Статичний аналіз може виявляти різноманітні проблеми, такі як:

- Синтаксичні помилки;
- Використання застарілих бібліотек;

- Порушення стандартів кодування;
- Вразливості, пов'язані з безпекою, такі як SQL ін'єкції, Cross-Site Scripting (XSS), небезпечне зберігання даних;
- Недоліки в архітектурі програми.

Інструменти для статичного аналізу:

- SonarQube - платформа, яка забезпечує неперервний аналіз якості коду і виявлення вразливостей, підтримує багато мов програмування;
- Checkmarx інструмент, який зосереджений на безпеці коду і може інтегруватися з різними середовищами розробки;
- Fortify продукт від HP, який надає широкі можливості для статичного аналізу коду і виявлення вразливостей на ранніх етапах розробки.

Переваги статичного аналізу:

- Швидкість аналізу статичний аналіз можна виконувати швидко і автоматично, що дозволяє інтегрувати його в процеси неперервної інтеграції.
- Раннє виявлення помилок помилки можуть бути виявлені на ранніх етапах розробки, що знижує вартість їх виправлення.
- Покращення якості коду регулярне використання статичного аналізу сприяє підтримці високих стандартів кодування і архітектури програми.

Недоліки статичного аналізу:

- Хибнопозитивні результати - однією з основних проблем є велика кількість хибнопозитивних сповіщень, які можуть вимагати додаткового часу для перевірки;
- Обмеження у виявленні деяких типів помилок - статичний аналіз не завжди може виявити помилки, пов'язані з контекстом виконання програми, такі як витоки пам'яті або проблеми з многопоточністю, які краще виявляються за допомогою динамічного аналізу.

Застосування статичного аналізу.

Проаналізувати цей метод можна на практичному прикладі мобільної розробки додатку, діаграма статистичного аналізу коду зображена на рисунку 2.1.

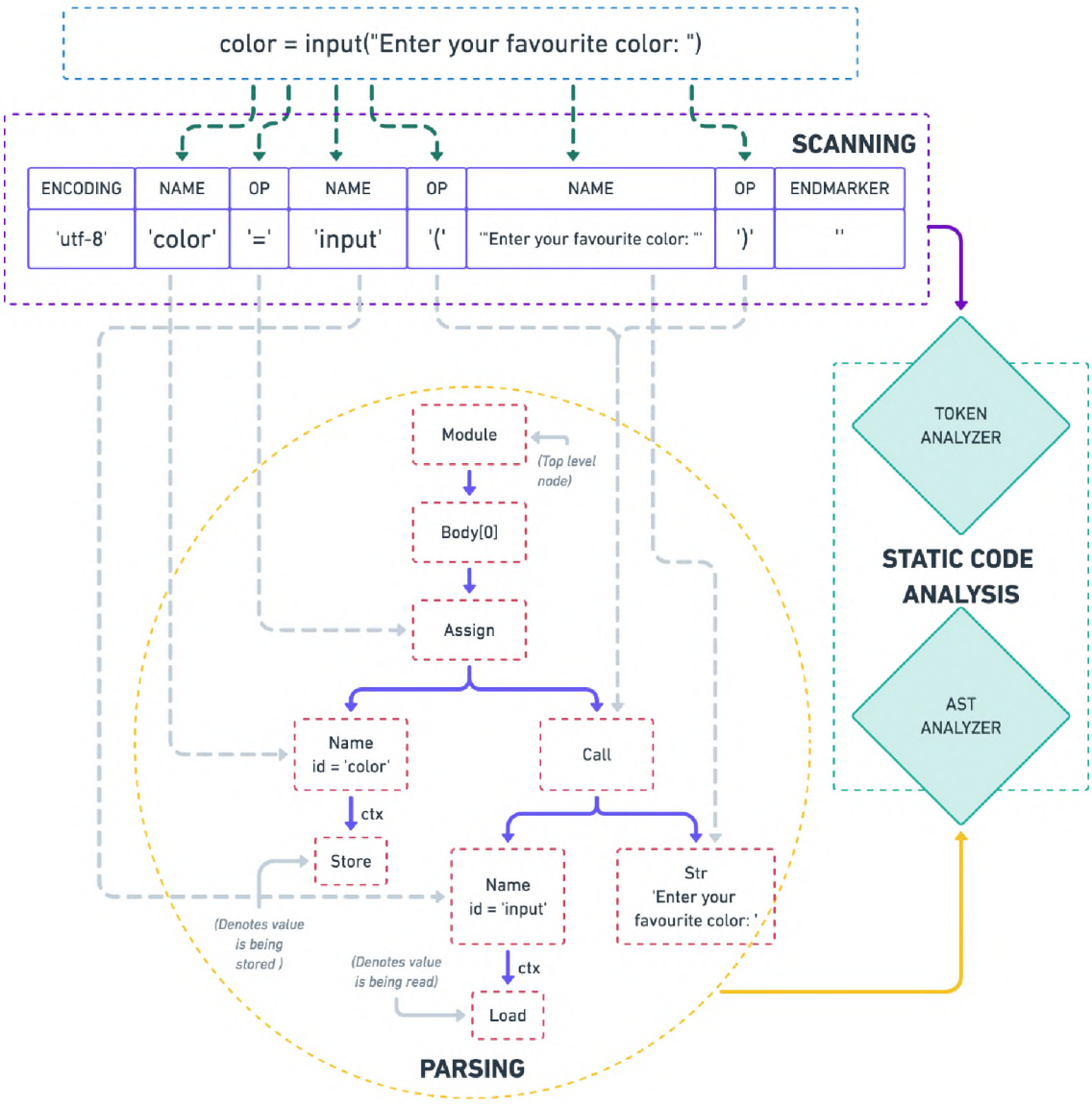

Рисунок 2.1 – Діаграма використання статистичного аналізу коду

Згідно діаграми, тут є декілька складових процесів перше, що робить компілятор, намагаючись зрозуміти фрагмент коду, це розбиває його на менші частини, також відомі як токени. Токени подібні до слів у мові.

Токен може складатися як з одного символу, наприклад з літералів (цілих чисел, рядків, наприклад, 7, Bob тощо) або зарезервованих ключових слів цієї

мови (наприклад, def у Python). Символи, які не впливають на семантику програми, такі як кінцеві пробіли, коментарі тощо, часто відкидаються сканером.

Далі відбувається синтаксичний аналіз.

На цьому етапі маємо лише словниковий запас мови, але самі по собі токени нічого не говорять про граматику мови. Саме тут вступає синтаксичний аналізатор.

Синтаксичний аналізатор бере ці лексеми, перевіряє, що послідовність, в якій вони з'являються, відповідає граматиці, і організовує їх у деревоподібну структуру, що представляє собою високорівневу структуру програми. Це дерево влучно називається абстрактним синтаксичним деревом (AST).

«Абстрактним» тому, що воно абстрагується від низькорівневих несуттєвих деталей, таких як дужки, відступи і т.д., дозволяючи користувачеві зосередитися тільки на логічній структурі програми - саме це робить його найбільш підходящим вибором для проведення статичного аналізу.

Потім аналіз АЛП.

Дерево синтаксису може бути досить великим і складним, що ускладнює написання коду для його аналізу. Оскільки це те, що всі компілятори (або інтерпретатори) роблять самостійно, існує певний інструментарій для спрощення цього процесу.

Потім аналіз на та виявлення занадто великої кількості вкладених циклів.

«Для циклів», які вкладені більш ніж на 3 рівні, неприємні на вигляд, складні для сприйняття мозком і викликають щонайменше проблеми при мобільній розробці.

Таким чином, продовжуючи говорити про аналіз даної діаграми на реальному прикладі, можна відзначити його значустість в мобільній розробці і у виявленні проблем в кібербезпеці.

Отже, статичний аналіз є невід'ємною частиною сучасних методологій розробки програмного забезпечення, таких як Agile і DevOps, де швидкість

розробки та неперервна інтеграція вимагають регулярного та автоматизованого контролю якості коду.

Використання статичного аналізу дозволяє розробникам вчасно виявляти та виправляти помилки, забезпечуючи вищу безпеку та надійність програмних продуктів.

Практичні приклади використання також є і в інший компаній:

- Великі технологічні компанії, такі як Google та Microsoft, інтегрують статичний аналіз у свої процеси розробки для забезпечення безпеки та якості своїх продуктів;
- Стартапи та малі компанії використовують вільнодоступні інструменти статичного аналізу для підтримки високих стандартів кодування без значних інвестицій у дороге програмне забезпечення.

Майбутнє статичного аналізу.

Розвиток штучного інтелекту та машинного навчання відкриває нові можливості для покращення статичного аналізу. Алгоритми машинного навчання можуть допомогти знизити кількість хибнопозитивних результатів та підвищити точність виявлення складних вразливостей. Також, інтеграція з іншими інструментами розробки та тестування може забезпечити більш глибокий та комплексний аналіз безпеки програмного забезпечення.

2.1.2 Динамічний аналіз

Динамічний аналіз, відомий також як виконавчий аналіз, полягає у виконанні програми в контрольованому середовищі, що дозволяє аналізувати поведінку програми в реальному часі. Цей метод використовується для виявлення помилок, які з'являються тільки під час виконання програми, таких як

витоки пам'яті, проблеми з управлінням пам'яттю, неправильне використання API, проблеми з многопоточністю, і взаємодії з операційною системою або з іншими програмами [12].

Інструменти для динамічного аналізу:

- Valgrind - інструмент для виявлення витоків пам'яті, неправильного використання пам'яті та виконання невизначених команд в програмах написаних на C та C++.
- Android Studio Profiler - інтегрований інструмент в Android Studio, який дозволяє моніторити використання CPU, пам'яті, і мережі в реальному часі для додатків Android.

Переваги динамічного аналізу:

- Виявлення складних помилок -дозволяє ідентифікувати помилки, які важко або неможливо виявити за допомогою статичного аналізу, особливо ті, що залежать від конкретних умов виконання.
- Тестування в реальних умовах - можливість тестування програми в умовах, максимально наближених до реального використання, забезпечує високу вірогідність виявлення реальних проблем.

Недоліки динамічного аналізу:

- Високі вимоги до ресурсів - динамічний аналіз часто вимагає значних обчислювальних ресурсів і часу, особливо для комплексних програм;
- Обмежена охоплення - оскільки аналіз залежить від виконання певних шляхів виконання коду, не всі частини програми можуть бути протестовані.

Практичне застосування.

Динамічний аналіз часто використовується у комбінації з іншими методами тестування для забезпечення всебічного аналізу програми. Наприклад, після виконання статичного аналізу, програма може бути піддана динамічному аналізу для подальшої перевірки і виявлення помилок, які не були і явлення помилок, які не були ідентифіковані під час статичного аналізу. Це особливо

важливо для виявлення проблем, які виникають тільки під час певних умов виконання або взаємодії з зовнішніми системами.

Майбутнє динамічного аналізу.

З розвитком технологій і збільшенням складності програмного забезпечення, динамічний аналіз стає ще більш важливим. Інновації в області автоматизації тестування та штучного інтелекту можуть допомогти оптимізувати процеси динамічного аналізу, зменшуючи витрати часу та ресурсів, а також підвищуючи точність і охоплення тестування. Інтеграція динамічного аналізу з іншими інструментами і методиками, такими як моніторинг виробництва та аналіз поведінки користувачів, може забезпечити ще глибший рівень аналізу та розуміння потенційних проблем у програмах.

Застосування в індустрії.

Багато великих технологічних компаній використовують динамічний аналіз як частину своїх стандартних процедур тестування безпеки. Це дозволяє їм не тільки забезпечити високий рівень якості своїх продуктів, але й швидко реагувати на нові загрози та вразливості, які можуть виникнути вже після випуску продукту на ринок.

2.1.3 Аналіз трафіку

Аналіз трафіку в мобільних додатках полягає у перехопленні та аналізі даних, які передаються між мобільним додатком і сервером. Цей метод дозволяє виявити вразливості, пов'язані з передачею даних, такі як відсутність шифрування, використання слабких алгоритмів шифрування, або витік конфіденційної інформації [13].

Інструменти:

- Wireshark - дозволяє перехоплювати та аналізувати пакети даних у реальному часі. Підтримує багато протоколів і може використовуватися для глибокого аналізу мережевого трафіку;

- Fiddler веб-дебагер, який може перехоплювати HTTP та HTTPS трафік між мобільними додатками та серверами. Він також дозволяє модифікувати трафік для тестування реакції додатків на різні умови;

- Charles Proxy - ще один популярний інструмент для перехоплення мережевого трафіку. Він дозволяє переглядати всі дані, що відправляються та отримуються мобільним додатком, включаючи HTTPS трафік.

Виходячи з аналізу цього методу можна зазначити що треба проводити детальний аналіз протоколів наступним чином:

- HTTP/HTTPS - детальний аналіз HTTP заголовків, методів (GET, POST, PUT, DELETE), статус-кодів відповідей, а також безпеки HTTPS конфігурацій. Важливо звернути увагу на використання безпечних заголовків, таких як «Strict-Transport-Security», «Content-Security-Policy», та інших, які можуть допомогти запобігти різним атакам;

- TCP/IP - вивчення низькорівневих аспектів мережевих сесій, таких як встановлення з'єднань, управління потоком і обробка помилок. Аналіз пакетів на наявність аномалій, які можуть вказувати на спроби злому або втручання в мережеву комунікацію;

Також основним чином, треба обов'язково проводити розширений аналіз вмісту:

- Шифрування даних - аналіз використання шифрування в передачі даних, включаючи використання сертифікатів SSL/TLS, їх силу та вразливості. Оцінка конфігурації шифрування на предмет використання застарілих алгоритмів або ключів недостатньої довжини;

- Витік інформації - ідентифікація потенційних витоків інформації через незашифровані або погано зашифровані дані. Аналіз може включати перевірку на витік інформації через параметри URL, HTTP заголовки, та тіло запитів;
- Cookies та сесії - аналіз безпеки сесійних cookies, їх атрибутів (наприклад, «HttpOnly», «Secure»), та впливу на безпеку сесії. Перевірка на використання cookies без належних захисних мір може вказувати на ризики злому сесій.

Переваги методу:

- Виявлення незашифрованих даних - можна ідентифікувати, які дані передаються відкритим текстом;
- Виявлення витоку інформації - чутлива інформація, така як паролі або персональні дані, може бути виявлена під час аналізу;
- Тестування політик безпеки - перевірка, чи дотримуються мобільні додатки політик безпеки, наприклад, використання HTTPS замість HTTP.

Недоліки:

- Складність налаштування - налаштування інструментів для перехоплення може бути складним і вимагати певних технічних знань;
- Потенційна незаконність - перехоплення трафіку без дозволу може бути незаконним у деяких юрисдикціях;
- Необхідність доступу до мережі - для ефективного перехоплення трафіку необхідно мати доступ до мережі, через яку проходить трафік.

Цей метод є важливим інструментом у виявленні та аналізі вразливостей мобільних додатків, але вимагає відпних додатків, але вимагає відповідального підходу та дотримання законодавчих норм.

2.1.4 Фаззінг

Фаззінг - це техніка автоматизованого тестування програмного забезпечення, яка включає генерацію великої кількості випадкових або напіввипадкових даних як вхідних значень для програми з метою виявлення помилок або вразливостей. Цей метод базується на припущенні, що непередбачені типи вхідних даних можуть викликати збої програми, включаючи витоки пам'яті, переповнення буфера, відмови в обслуговуванні та інші критичні помилки [14].

Типи фаззінгу:

- Чорний ящик - тестування без знання внутрішньої структури програми;
- Сірий ящик - комбінація знань про внутрішню структуру та випадкових тестів;
- Білий ящик - використання детальних знань про внутрішню структуру програми для генерації вхідних даних;
- Методи генерації даних - використання алгоритмів для створення вхідних даних, які максимально покривають можливі варіанти введення, включаючи крайні та некоректні значення.

Коли говорити про цей метод то треба розуміти які є інструменти, а саме:

- AFL (American Fuzzy Lop) - використовує алгоритми генетичного програмування для оптимізації вхідних даних, що призводять до збоїв;
- Burp Suite Intruder - надає можливість детально налаштовувати вхідні параметри для веб-додатків, імітуючи атаки на веб-форми та API.

Аналізуючи цей метод, можна виділити його переваги:

- Виявлення складних вразливостей - фаззінг може виявити вразливості, які важко виявити за допомогою інших методів, таких як ручне тестування або статичний аналіз коду;
- Автоматизація - фаззінг дозволяє автоматизувати процес виявлення помилок, значно збільшуючи обсяг тестування без додаткових витрат часу.

Але якщо говорити про недоліки то можна зазначити наступні:

- Обмеження в області покриття - фаззінг може не виявити всі вразливості, особливо ті, що вимагають специфічних умов або складних взаємодій;
- Ресурсоємність - висока вимога до обчислювальних ресурсів, особливо при використанні методів білого ящика, де необхідно аналізувати великі обсяги коду.

Отже, аналізуючи цей метод можна зазначити рекомендації для ефективного фаззінгу:

- Ретельний вибір цілей тестування - важливо вибрати компоненти або модулі, які є критичними для безпеки системи, або ті, що містять складну логіку обробки даних;
- Комбінація методів фаззінгу - використання різних типів фаззінгу (чорний, сірий, білий ящик) для забезпечення більшого покриття тестування та виявлення різноманітних вразливостей;
- Аналіз результатів - важливо мати ефективні інструменти та процеси для аналізу результатів фаззінгу, щоб швидко ідентифікувати реальні вразливості та відсіяти хибні тривоги;
- Налаштування середовища тестування - забезпечення належного відновлення тестового середовища після кожного тесту для забезпечення стабільності та надійності процесу тестування.

Отже, фаззінг є потужним інструментом у арсеналі тестувальника програмного забезпечення, здатним виявляти вразливості, які можуть залишитися непоміченими іншими методами. Втім, для досягнення максимальної ефективності, важливо інтегрувати фаззінг у комплексну стратегію тестування безпеки.

2.1.5 Комбіновані методи

Комбіновані методи використовуються для підвищення ефективності виявлення вразливостей шляхом інтеграції різних технік аналізу. Це може включати статичний аналіз коду, динамічний аналіз виконання, фаззінг, тестування проникнення, і навіть машинне навчання для прогнозування потенційних вразливостей на основі історичних даних. Цей підхід дозволяє виявляти вразливості, які можуть залишитися непоміченими при використанні лише одного методу.

Інструменти:

- Спеціалізовані платформи - інструменти як SonarQube, Veracode, або Synopsys Coverity інтегрують статичний і динамічний аналіз, надаючи комплексний огляд безпеки коду;
- Інтеграція з CI/CD - автоматизація тестування безпеки в процесах неперервної інтеграції та неперервного розгортання забезпечує постійне виявлення та усунення вразливостей;

Можна зазначити після дитального аналізу що перевагами є:

- Всебічний аналіз - комбінація методів забезпечує глибокий аналіз вразливостей з різних точок зору;
- Проактивне виявлення - можливість виявляти потенційні вразливості до того, як вони будуть експлуатовані в атаках.

Але недоліками в цьому випадку є:

- Складність управління - інтеграція та управління кількома інструментами може бути складною;
- Високі витрати на інтеграцію - початкові витрати на налаштування та підтримку комбінованих методів можуть бути значними.

Отже, анализуючи застосування методів на реальних прикладах, наприклад:

- Кейс, додаток банкінгу, використання комбінованих методів у фінансовому додатку дозволило виявити складні вразливості SQL ін'єкцій, які не були виявлені під час стандартного статичного аналізу;
- Вплив - виявлення та усунення цих вразливостей значно підвищило безпеку додатку та захистило конфіденційні дані користувачів;

Порівняльний аналіз методів:

- Швидкість виявлення - комбіновані методи часто забезпечують швидше виявлення вразливостей, оскільки вони можуть одночасно використовувати різні техніки для моніторингу та аналізу системи;
- Точність - завдяки використанню різних методів, комбіновані підходи зменшують кількість хибних позитивних і негативних результатів, підвищуючи точність виявлення реальних вразливостей;
- Вартість інструментів - хоча ініціальні витрати можуть бути високими, довгострокова перевага виявлення та усунення вразливостей на ранніх стадіях може значно знизити потенційні збитки від безпекових інцидентів;
- Легкість інтеграції в розробку - інтеграція багатьох інструментів вимагає часу та ресурсів, але сучасні рішення надають платформи, які спрощують цей процес;
- Вплив на продуктивність додатка - необхідно забезпечити, що використання комбінованих методів не впливає негативно на продуктивність додатка, особливо в режимі реального часу.

Комбіновані методи виявлення вразливостей є важливим інструментом в арсеналі розробників та аналітиків безпеки, які прагнуть забезпечити високий рівень захисту мобільних додатків. Вони дозволяють не тільки виявляти вразливості на різних етапах розробки, але й забезпечують можливість реагування на потенційні загрози в більш оперативному порядку.

2.1.6 Порівняльний аналіз всих методів

Отже, підсумовуючи та аналізуючи всі методи можна відобразити в вигляді таблиці 2.1, яка зосереджена на загальних характеристиках кожного методу виявлення вразливостей, таких як швидкість виявлення, точність, вартість інструментів, легкість інтеграції в розробку, та вплив на продуктивність додатка. Ці параметри допомагають оцінити ефективність та практичність кожного методу в контексті реальних проектів розробки програмного забезпечення. Комбіновані методи в цій таблиці вирізняються високою швидкістю та точністю виявлення, хоча й мають вищу вартість інструментів та середню легкість інтеграції.

Таблиця 2.1 - Основні характеристики методів виявлення вразливостей

Метод	**Швидкість виявлення**	**Точність**	**Вартість Інструментів**	**Легкість інтеграції в розробку**	**Вплив на продуктивність**
Статичний аналіз	Середня	Висока	Середня	Висока	Мінімальний
Динамічний аналіз	Висока	Середня	Висока	Середня	Модерний
Фаззінг	Висока	Середня	Середня	Середня	Модерний
Тестування проникнення	Низька	Висока	Висока	Низька	Мінімальний
Комбіновані методи	Висока	Висока	Висока	Середня	Модерний

2.1.7 Огляд недавніх досліджень та наукових робіт щодо ефективності методів виявлення вразливостей

Останніми роками було проведено значну кількість досліджень, спрямованих на оцінку та порівняння ефективності різних методів виявлення вразливостей у програмному забезпеченні. Ці дослідження зосереджуються на

аналізі точності, швидкості виявлення, вартості інструментів, та інших критичних параметрах, які впливають на вибір методу виявлення вразливостей.

Основні напрямки досліджень:

- Порівняльний аналіз методів - багато досліджень фокусуються на порівнянні статичного аналізу, динамічного аналізу, фаззінгу, та тестування проникнення. Вони оцінюють, як кожен метод виявляє різні типи вразливостей та які умови найкраще підходять для їх застосування.

- Ефективність комбінованих методів - деякі дослідження зосереджені на вивченні комбінованих методів, демонструючи, що інтеграція кількох підходів може значно підвищити точність та швидкість виявлення вразливостей;

- Розвиток нових технологій - інноваційні дослідження включають розробку та впровадження нових алгоритмів машинного навчання та штучного інтелекту для прогнозування вразливостей на основі історичних даних та поведінкових моделей.

Аналіз пркиладів ключових наукових робіт цього напрямку:

- «A Comparative Study of Software Vulnerability Detection Technique» (Порівняльне дослідження технік виявлення вразливостей програмного забезпечення). Ця робота аналізує ефективність статичного та динамічного аналізу в контексті різних програмних архітектур. Це дослідження зосереджується на аналізі та порівнянні статичного та динамічного аналізу як основних методів виявлення вразливостей у програмному забезпеченні. Воно розглядає, як кожен з цих методів працює в різних програмних архітектурах, включаючи монолітні, мікросервісні та розподілені системи.

До прикладу, статичний аналіз, тут аналізується його здатність виявляти вразливості на ранніх стадіях розробки, переваги включають високу точність виявлення типових вразливостей, таких як SQL ін'єкції та XSS без необхідності виконання коду.

Але динамічний аналіз тут, оцінює його ефективність у виявленні вразливостей, які проявляються тільки під час виконання програми, таких як витоки пам'яті та переповнення буфера;

- «Effectiveness of Combined Detection Techniques in Software Security» (Ефективність комбінованих технік виявлення у безпеці програмного забезпечення). Дослідження показує, як комбінація різних методів може забезпечити більш глибокий аналіз та виявлення складних вразливостей. Особливо увага йде на комбінованих методах виявлення вразливостей, які інтегрують статичний аналіз, динамічний аналіз, фаззінг та інші техніки для забезпечення більш глибокого аналізу безпеки.

Комбіновані методи, тут аналізується, як інтеграція різних методів може підвищити точність виявлення складних вразливостей, які можуть бути пропущені при використанні одного методу.

Практичне застосування це також один з аналізів в цій роботі, дослідження включає приклади з реальних проектів, де комбіновані методи допомогли ідентифікувати та усунути серйозні вразливості, що значно підвищило рівень безпеки програмних продуктів.

Ці дослідження та наукові роботи відіграють важливу роль у розвитку методів виявлення вразливостей, нодів виявлення вразливостей, надаючи фахівцям у галузі безпеки інформацію, необхідну для вибору найефективніших інструментів та стратегій. Вони також сприяють розумінню потенційних обмежень існуючих методів і стимулюють розробку нових підходів, які можуть краще вирішувати специфічні виклики в області кібербезпеки.

2.2 Інструменти для аналізу безпеки мобільних додатків

Аналіз безпеки мобільних додатків вимагає спеціалізованих інструментів, які можуть виявляти потенційні вразливості та забезпечувати захист від атак. Ці інструменти варіюються від статичних аналізаторів коду до динамічних тестерів виконання та систем фаззінгу.

Статичний аналіз коду (SAST) - інструменти, які аналізують джерельний код на предмет вразливостей без виконання програми. Наприклад такі як SonarQube, Checkmarx, Fortify.

Динамічний аналіз коду (DAST) - інструменти, які тестують додаток під час його виконання для виявлення вразливостей. Наприклад такі як OWASP ZAP, Burp Suite, AppScan.

Інтерактивний аналіз безпеки (IAST) - комбінує методи SAST та DAST для забезпечення більш глибокого аналізу в реальному часі. Такі як Contrast Security, Veracode.

Мобільний фаззінг - автоматизовані інструменти, які генерують та вводять несподівані або випадкові дані в додатки для виявлення помилок. Наприклад AFL, Google's ClusterFuzz.

Аналіз залежностей та компонентів інструменти для виявлення вразливостей у бібліотеках та залежностях, які використовуються в додатках. Приклади: OWASP Dependency-Check, Snyk.

Отже аналізуючи інстурменти треба підбити підсумки їх. Тому можна визначити основний опис та висновок до кожного.

Статичний аналіз коду (SAST):

- Опис. Аналізує джерельний код на предмет вразливостей без виконання програми. Це дозволяє виявити проблеми на ранніх стадіях розробки;

- Переваги. Висока точність виявлення статичних вразливостей, можливість інтеграції з IDE;
- Недоліки. Може генерувати хибнопозитивні результати, не виявляє вразливостей, які залежать від виконання коду.

Динамічний аналіз коду (DAST):

- Опис. Тестує додаток під час його виконання для виявлення вразливостей, які проявляються тільки під час роботи програми;
- Переваги. Здатність виявляти вразливості в реальному часі, включаючи ті, що залежать від конкретного середовища виконання;
- Недоліки. Вимагає налаштування тестового середовища, може бути повільним і ресурсоємним.

Інтерактивний аналіз безпеки (IAST):

- Опис. Комбінує методи SAST та DAST для забезпечення більш глибокого аналізу в реальному часі;
- Переваги. Висока точність виявлення, можливість виявляти складні вразливості;
- Недоліки. Вимагає інтеграції з процесом розробки, може бути складним у налаштуванні.

Мобільний фаззінг:

- Опис. Автоматизовані інструменти, які генерують та вводять несподівані або випадкові дані в додатки для виявлення помилок;
- Переваги. Здатність виявляти вразливості, які можуть бути пропущені іншими методами;
- Недоліки. Вимагає значних ресурсів для ефективного тест ування, може бути часозатратним і потребує детального аналізу результатів для ідентифікації реальних вразливостей з-поміж великої кількості хибнопозитивних сигналів.

Аналіз залежностей та компонентів:

- Опис. Інструменти для виявлення вразливостей у бібліотеках та залежностях, які використовуються в додатках;
- Переваги. Допомагає уникнути використання компонентів з відомими вразливостями, забезпечує оновлення безпеки;
- Недоліки. Не може виявити вразливості в власному коді додатка, залежить від точності баз даних вразливостей.

При виборі інструментів для аналізу безпеки мобільних додатків важливо враховувати наступні аспекти:

- Сумісність з мовами програмування та платформами. Інструмент повинен підтримувати мови програмування та платформи, які використовуються у вашому проекті;
- Інтеграція з інструментами розробки та CI/CD. Інструменти повинні легко інтегруватися з існуючими процесами розробки та системами неперервної інтеграції та доставки;
- Управління вразливостями та звітність. Інструменти повинні надавати зрозумілі та детальні звіти про вразливості, що дозволяють швидко ідентифікувати та усувати проблеми;
- Масштабованість та продуктивність. Інструменти повинні ефективно обробляти великі обсяги коду без значного впливу на продуктивність розробки.

Вибір правильного інструменту для аналізу безпеки мобільних додатків є критично важливим для забезпечення високого рівня безпеки та захисту від потенційних загроз. Ретельний вибір, заснований на детальному аналізі потреб проекту та можливостей інструментів, може значно підвищити ефективність заходів безпеки.

2.3 Розробка власного методу виявлення та аналізу вразливостей на базі ШІ

При розробці власного методу, спершу проведемо огляд алгоритмів машинного навчання та глибокого навчання.

Класифікація:

- Логістична регресія Простий, але ефективний метод для бінарної класифікації. Може використовуватися для визначення, чи містить код потенційні вразливості [15-16];
- Випадковий ліс (Random Forest). Ансамбль дерев рішень, який ефективно класифікує великі набори даних і забезпечує високу точність;
- Градієнтний бустинг. Послідовне покращення слабких моделей, здатне виявляти складні взаємозв'язки в даних.

Кластеризація:

- K-means. Розділяє набір даних на K кластерів, що може допомогти ідентифікувати нетипові патерни поведінки, які можуть вказувати на вразливості;
- DBSCAN. Заснований на щільності метод кластеризації, ефективний для виявлення викидів.

Нейронні мережі:

- Конволюційні нейронні мережі (CNN). Ефективні для аналізу візуальних даних, можуть бути адаптовані для аналізу структури коду;
- Рекурентні нейронні мережі (RNN). Ідеально підходять для аналізу послідовних даних, наприклад, для аналізу потоків даних або логів.

Посилене навчання (Reinforcement Learning):

- Може бути використане для розробки систем, які адаптуються до змінюваних умов безпеки, вчаться з власного досвіду і оптимізують свої стратегії виявлення вразливостей.

2.3.1Вибір підходящих моделей ШІ для розробки методу

Вибір моделі залежить від типу даних та специфіки задачі:

- Для аналізу коду можуть бути ефективні CNN або RNN, оскільки вони дозволяють враховувати структурні та послідовні залежності в коді [17-18];
- Для аналізу поведінки програми в реальному часі може бути корисним посилене навчання, яке дозволить системі самостійно адаптуватися до нових загроз.

2.3.2 Розробка модулів

Система виявлення вразливостей на базі штучного інтелекту (ШІ) розробляється для автоматизації процесу ідентифікації потенційних вразливостей у програмному коді. Вона використовує методи машинного навчання для аналізу коду, виявлення патернів, які можуть вказувати на вразливості, та надання рекомендацій щодо їх усунення.

Система складається з основних модулів, представлені нижче.

Модуль завантаження та очищення коду:

- Завантаження коду. Використовує GitHub API для отримання коду з різних репозиторіїв. Використання даних з GitHub для навчання моделі машинного навчання має кілька важливих переваг, які роблять цей підхід особливо привабливим для розробки систем виявлення вразливостей.

По-перше, велика кількість даних, GitHub є однією з найбільших платформ для розміщення коду з мільйонами користувачів та репозиторіїв. Це забезпечує доступ до величезної кількості даних, що дозволяє моделям машинного навчання вчитися на різноманітних прикладах коду, що покращує їх здатність узагальнювати та адаптуватися до нових, невідомих випадків.

По-друге, різноманітність коду - код на GitHub представлений у багатьох мовах програмування та включає різні стилі написання та архітектурні підходи. Це дозволяє моделям вчитися розпізнавати вразливості в широкому спектрі програмних контекстів, що збільшує їх універсальність та ефективність.

По-третє, актуальність даних. Програмування та технології швидко розвиваються, і код, який розміщується на GitHub, часто оновлюється та відображає сучасні тенденції в розробці програмного забезпечення. Навчання моделей на актуальних даних допомагає забезпечити, що вони ефективно справляються з сучасними вразливостями.

Також, доступність відкритих даних. Багато репозиторіїв на GitHub є відкритими, що забезпечує легкий доступ до великої кількості коду без необхідності укладення спеціальних угод або покупки даних для навчання. Це знижує бар'єри для входу в розробку систем виявлення вразливостей.

Головне це реальні сценарії використання. Код, який розміщений на GitHub, часто використовується в реальних проектах і продуктах. Навчання на таких даних допомагає моделям машинного навчання краще розуміти реальні сценарії використання та вразливості, які можуть виникати в них.

Отже, використання даних з GitHub для навчання моделей виявлення вразливостей дозволяє створювати більш точні, ефективні та адаптивні системи, які можуть виявляти широкий спектр вразливостей у програмному коді. Це забезпечує високий рівень безпеки програмних продуктів та допомагає організаціям уникнути потенційних загроз, пов'язаних з вразливостями в коді;

- Очищення коду. Видаляє коментарі, рядки та числа, щоб підготувати чистий код для аналізу.

Модуль векторизації коду:

- Векторизація. Перетворює очищений код у числові вектори за допомогою TF-IDF векторизатора, що дозволяє машинному навчанню ефективно обробляти текст.

Модуль бази даних:

- Зберігає дані про вразливості, історію сканувань та результати аналізу для подальшого використання та аудиту.

Модуль виявлення вразливостей:

- Аналізує векторизований код за допомогою нейронної мережі для ідентифікації потенційних вразливостей.

Модуль реагування:

- Застосування патчі. Автоматично застосовує виправлення до виявлених вразливостей;
- Надсилання повідомлень. Інформує розробників або системних адміністраторів про виявлені вразливості.

Користувацький інтерфейс:

- Відображає дашборд зі статистикою вразливостей та генерує детальні звіти.

2.3.3 Процес роботи системи

Дана система працює наступним чином, описаним нижче.

Завантаження та очищення коду. Система починає з завантаження коду з репозиторіїв, очищення його від непотрібних елементів та підготовки до аналізу. Фрагмент роботи завантаження та очистки коду зображено на рисунку 2.2.

```
    def evaluate(self, X, y, **kwargs):
        check_is_fitted(self, 'model')
        X = check_array(X)
        return self.model.evaluate(X, y, **kwargs)

# Функція для завантаження файлів з GitHub
def fetch_github_files(user, repo, file_path):
    url = f"https://api.github.com/repos/{user}/{repo}/contents/{file_path}"
    headers = {'Authorization': 'token ghp_oR5PVP9ioUbV55s7nZSc57pBIXczIu3vYB05', 'Accept': 'application/vnd.github.v3.raw'}
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.text
    else:
        return None

# Генерація шляхів до файлів
def generate_file_paths():
    repos = [
        ('facebook', 'react', ['packages/react/src/ReactElement.js', 'packages/react-dom/src/client/ReactDOM.js']),
        ('microsoft', 'TypeScript', ['src/compiler/types.ts', 'src/compiler/scanner.ts']),
    ]
    paths = []
    for user, repo, files in repos:
        for file in files:
            if file.endswith('.js') or file.endswith('.ts') or file.endswith('.java')  or file.endswith('.html'):
                paths.append((user, repo, file))
    return paths
import requests

def fetch_files_from_repo(user, repo, path="", token="GITHUB_TOKEN", max_files=200):
    """
    Рекурсивно отримує всі файли з репозиторію на GitHub, що відповідають заданим розширенням, але не більше max_files.
    """
    url = f"https://api.github.com/repos/{user}/{repo}/contents/{path}"
    headers = {'Authorization': f'token {token}'}
    response = requests.get(url, headers=headers)
    files = []

    if response.status_code == 200:
        contents = response.json()
```

(1m 41s) <cell line: 16... > load_data_from_githu... > fetch_github_file... > ge... > reques... > reques... > sen... > sen... > urlope... > _make_reques... > _validate_con... > conne

Рисунок 2.2 – Завантаження даних

Векторизація коду. Очищений код перетворюється на вектори, що дозволяє машинному навчанню ефективно його обробляти. Дані виглядають після обробки і векторизації наступмним чином зображеним на рисунку 2.3.

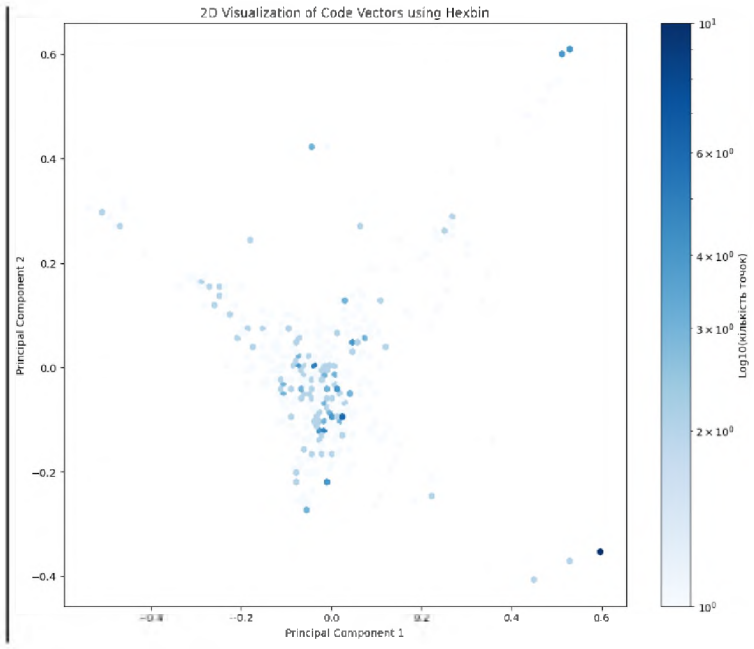

Рисунок 2.3 – Графік векторизації даних

Згідно цього рисунку можна зазначити що було використано «TfidfVectorizer» з «RegexpTokenizer» для токенізації та векторизації очищеного коду. Це дозволило перетворити текст коду на TF-IDF вектори, які відображають важливість кожного слова (токена) у контексті документу (файлу коду).

Виявлення вразливостей. Векторизований код подається на вхід нейронної мережі, яка аналізує його на предмет патернів, що можуть вказувати на вразливості.

Після векторизації, векторизований код подається на вхід нейронної мережі, яка аналізує його на предмет патернів, що можуть вказувати на вразливості. Нейронна мережа, зокрема, навчена розпізнавати різні типи уразливостей, використовуючи навчальні дані, які містять приклади вразливого та не вразливого коду.

Реагування на вразливості. Як тільки вразливості ідентифіковані нейронною мережею, система автоматично вживає заходів для їх усунення.

Це включає автоматичне застосування виправлень (патчів) до вразливих частин коду або відправлення повідомлень відповідним особам, таким як розробники або системні адміністратори, з детальною інформацією про виявлені уразливості та рекомендаціями щодо їх виправлення.

Система також включає модулі для відстеження відповідей та перевірки ефективності вжитих заходів, забезпечуючи повний цикл управління вразливостями.

Зберігання та відображення результатів. Результати сканування та виявлення вразливостей зберігаються в базі даних.

Користувацький інтерфейс відображає дашборд зі статистикою вразливостей та дозволяє генерувати детальні звіти.

2.3.4 Детальний опис технологічного стеку при побудові системи виявлення вразливостей

Отже в при побудові нового методу на основі ШІ, для стоврення цієї системи, було використано специфічний стек технологій.

Python - використовується як основна мова програмування для всієї системи через її здатність інтегруватися з різними бібліотеками і фреймворками, що є критично важливим для розробки складних систем, які включають машинне навчання та обробку даних. Python також дозволяє легко прототипувати і тестувати нові ідеї, що є важливим на етапах дослідження та розробки.

TensorFlow - використовується для побудови та тренування нейронних мереж, які відповідають за виявлення патернів вразливостей у коді. TensorFlow дозволяє ефективно масштабувати обчислення, що є необхідним для обробки великих наборів даних.

Keras, як високорівневий API для TensorFlow, використовується для спрощення процесу проектування та тренування моделей. Keras дозволяє швидко створювати прототипи моделей з різними архітектурами, що сприяє експериментуванню та вибору найкращої моделі для задачі.

Scikit-learn використовується для векторизації коду за допомогою TF-IDF та інших завдань перед обробкою даних, таких як розділення даних на навчальні та тестові набори. Ця бібліотека також може використовуватися для реалізації додаткових методів машинного навчання, які можуть бути корисними для порівняння ефективності різних підходів.

Pandas - використовується для ефективної обробки та аналізу даних, зокрема для завантаження, очищення та трансформації даних перед подальшою обробкою. Pandas надає зручні структури даних для швидкого доступу та маніпуляцій з великими наборами даних.

NumPy є основою для обробки числових даних у Python. Використання NumPy дозволяє виконувати високопродуктивні обчислення з масивами та матрицями, що є основою для багатьох операцій у машинному навчанні та обробці даних.

Requests використовується для взаємодії з GitHub API для взаємодії з GitHub API для завантаження вихідного коду з репозиторіїв. Ця бібліотека дозволяє легко виконувати HTTP запити, обробляти відповіді та керувати помилками, що є критично важливим для забезпечення стабільності та надійності процесу завантаження коду.

Цей технологічний стек був обраний не тільки за його технічні можливості, але й за здатність інтегруватися та працювати разом, що забезпечує високу продуктивність, гнучкість та масштабованість системи виявлення вразливостей. Використання цих технологій дозволяє розробникам ефективно вирішувати складні завдання, пов'язані з аналізом великих обсягів коду та ідентифікацією потенційних вразливостей.

Розроблена система виявлення вразливостей на базі ШІ дозволяє автоматизувати процес ідентифікації та реагування на потенційні вразливості в програмному коді, значно підвищуючи ефективність та швидкість обробки великих обсягів коду. Це забезпечує більш високий рівень безпеки програмного забезпечення та допомагає організаціям уникнути можливих збитків від атак.

2.4 Практичне застосування методу

Першим кроком це навчання моделі на реальних даних. Том проведено підготовку цих даних.

У рамках підготовки даних для тренування моделі виявлення вразливостей у мобільних додатках, було виконано наступні кроки:

- Збір даних. Використовуючи GitHub API, було зібрано реальні приклади коду з відкритих репозиторіїв. Це включало в себе вибірку файлів з розширеннями ріхних. З репозиторіїв. Завдяки функції «fetch_files_from_repo», було автоматично зібрано велику кільсть файлів, що відповідають заданим критеріям;

- Анотація даних. Кожен файл коду був проаналізований на наявність патернів, які можуть вказувати на вразливості, такі як SQL ін'єкції та XSS (Cross-site Scripting) та інші. Цей процес включав ручну перевірку та класифікацію коду, що дозволило точно ідентифікувати та позначити вразливі та не вразливі фрагменти;

- Перетворення коду у векторну форму. Застосування методу TfidfVectorizer дозволило перетворити текст коду в числові вектори. Використання токенізатора «RegexpTokenizer» забезпечило ефективне розділення тексту на токени, що є критично важливим для точності векторизації;

- Розділення даних на навчальні та тестові набори. Дані були розділені на навчальний та тестовий набори за допомогою функції «train_test_split», що забезпечило можливість об'єктивної оцінки ефективності моделі під час тестування;

- Нормалізація даних. Виконано нормалізацію даних для забезпечення уніформності вхідних даних, що допомагає уникнути упередженості моделі до певних особливостей коду;

Ці кроки забезпечили створення якісної та репрезентативної вибірки даних, яка була використана для тренування нейронної мережі, здатної ефективно виявляти вразливості в коді мобільних додатків.

Далі для застосування на практичному прикладі відбулося тренування моделі.

Процес тренування нейронної мережі для виявлення вразливостей у коді мобільних додатків включав наступні етапи описані нижче.

Було обрано архітектуру глибокої нейронної мережі з декількома шарами. Основна конфігурація моделі включала:

- Вхідний шар, який приймає векторизовані дані коду;
- Декілька прихованих шарів з активацією ReLU для нелінійності та шарами Dropout для запобігання перенавчанню;
- Вихідний шар з сигмоїдною активацією для класифікації коду як вразливого або не вразливого.

Параметри тренування:

- Кількість епох. Встановлено 100 епох з можливістю раннього припинення для оптимізації тренування;
- Розмір пакету. Використано пакети по 10 прикладів для балансу між часом тренування та стабільністю навчання;
- Оптимізатор. Adam, завдяки його ефективності у різних задачах глибокого навчання;
- Функція втрат. Бінарна крос-ентропія, яка є стандартом для бінарної класифікації.

Використання технік оптимізації:

- Раннє припинення: Для запобігання перенавчанню та оптимізації часу тренування були використані два види раннього припинення.

Ці параметри та методики були обрані на основі попередніх досліджень та експериментів, які показали їх ефективність у подібних задачах. Такий підхід дозволив створити міцну та надійну модель, здатну точно виявляти потенційні вразливості в коді мобільних додатків.

Валідація та тестування моделі.

Для валідації та тестування моделі використовуються незалежні набори даних, щоб перевірити її ефективність та точність у виявленні вразливостей. Використовується крос-валідація для оцінки стабільності моделі, а також

розділення даних на тренувальні, валідаційні та тестові набори. Оцінка моделі включає аналіз матриці помилок, точності, відгуку, F1-міри та ROC-кривих.

Інтеграція у розробку додатків.

Система може бути інтегрована у процес розробки мобільних додатків через CI/CD пайплайни, де автоматичне сканування коду на вразливості виконується при кожному коміті або злитті змін. Це забезпечує виявлення та виправлення вразливостей до релізу продукту.

Реагування на виявлені вразливості.

Після виявлення вразливостей система автоматично застосовує патчі або рекомендації щодо виправлень, відправляє повідомлення розробникам та відстежує статус виправлень. Це допомагає забезпечити швидке реагування на потенційні загрози як показано на рисунку 2.4.

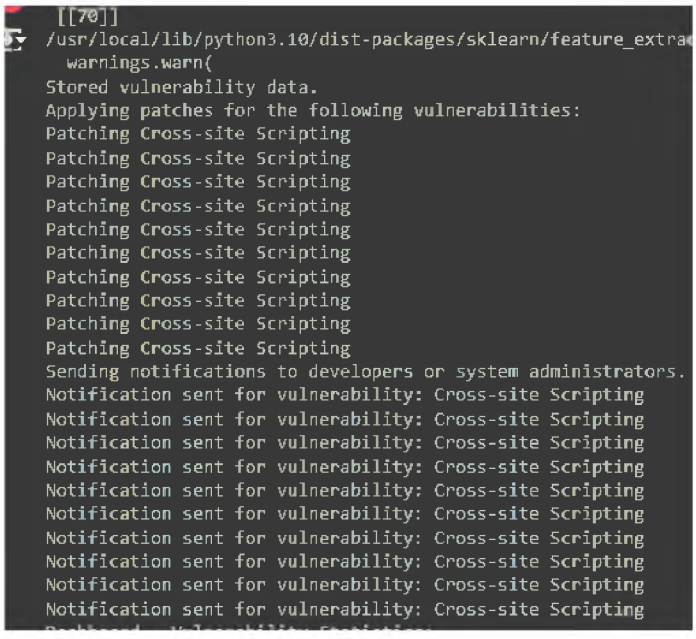

Рисунок 2.4 – Автоматичні патчі

Моніторинг та звітність.

Система забезпечує моніторинг ефективності вжитих заходів безпеки та генерує звіти про стан безпеки мобільних додатків. Це включає аналіз трендів вразливостей, оцінку ризиків та аудит дій, що дозволяє керівництву приймати обґрунтовані рішення щодо поліпшення заходів безпеки.

2.5 Висновки до розділу

Розроблений метод як системп виявлення вразливостей у коді мобільних додатків демонструє значний потенціал для підвищення рівня безпеки програмного забезпечення. Інтеграція цієї системи як методу у процеси розробки дозволяє автоматично ідентифікувати та виправляти потенційні вразливості на ранніх етапах, що знижує ризики витоку даних та інших безпекових інцидентів.

Використання автоматизованих інструментів для сканування коду та відповіді на виявлені вразливості значно спрощує процеси перевірки безпеки та дозволяє розробникам зосередитися на оптимізації функціональності додатків. Завдяки моніторингу та звітності, керівництво має можливість відстежувати стан безпеки продуктів у реальному часі та приймати своєчасні рішення для управління ризиками.

Особливо важливим є досягнення високої точності тренування моделі, яка в даному випадку склала 0.95. Це свідчить про високу здатність системи точно класифікувати вразливі та не вразливі участки коду, що є ключовим для ефективної роботи системи. При досягненні такої точності тренування, система автоматично зупиняє процес навчання, що допомагає уникнути перенавчання та зберегти ресурси.

Таким чином, впровадження цієї системи не тільки сприяє підвищенню безпеки мобільних додатків, але й підвищує довіру користувачів та клієнтів, забезпечуючи надійний захист їхніх даних.

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Оцінка ефективності та вигідності використання запропонованих методів

Отже, після створення методу та системи, далі здійснюється детальний аналіз ефективності та вигідності впровадження розробленої системи виявлення вразливостей у мобільних додатках на базі штучного інтелекту. Основна увага зосереджена на порівнянні цієї системи з традиційними методами виявлення вразливостей, які часто вимагають значних ручних зусиль та часу на аналіз коду.

Порівняння з традиційними методами.

Традиційні методи виявлення вразливостей, такі як ручний аудит коду та використання стандартних інструментів статичного аналізу, часто є часомісткими та можуть пропускати складні вразливості. Натомість, система на базі ШІ використовує алгоритми машинного навчання для автоматичного розпізнавання потенційних вразливостей, що значно підвищує швидкість та точність аналізу.

Аналіз витрат на впровадження та обслуговування.

Впровадження системи виявлення вразливостей на базі ШІ вимагає початкових інвестицій у розробку та налаштування моделі, а також витрат на обслуговування та оновлення системи. Однак, ці витрати компенсуються зниженням витрат на ручний аналіз та виправлення помилок.

Початкові інвестиції у розробку та налаштування системи складають $50,000 (2,000,000 грн). Ця сума включає витрати на наймання кваліфікованих спеціалістів, розробку алгоритмів машинного навчання, придбання необхідного обладнання та програмного забезпечення.

Наймання кваліфікованих спеціалістів: $30,000 (1200, 000 грн).

Розрахунок:

2 спеціалісти*3 місяці*$5,000/місяць=$30,000 (1200, 000 грн).

Розробка алгоритмів машинного навчання: $10,000 (400 000 грн).

Розрахунок:

1 спеціаліст*2 місяці*$5,000/місяць=$10,000 (400 000 грн).

Придбання обладнання та програмного забезпечення: $10,000(400 000 грн).

Розрахунок:

Сервери, ліцензії на програмне забезпечення, інструменти для розробки = $10,000 (400 000 грн).

Щорічні витрати на обслуговування.

Сума: $10,000 (400 000 грн).

Складові:

- Оновлення програмного забезпечення: $4,000 (160 000 грн).

Розрахунок:

Ліцензії на оновлення, підтримка програмного забезпечення = $4,000(160 000 грн).

- Підтримка системи: $3,000 (120 000 грн).

Розрахунок:

Зарплата технічного персоналу для підтримки системи = $3,000 (120 000 грн).

- Періодичне навчання моделі: $3,000 (120 000 грн).

Розрахунок:

2 * $1,500 = $3,000 (120 000 грн).

Витрати на ручний аналіз.

Сума: $30,000 на рік (1200 000 грн).

Складові:

- Заробітна плата аналітиків безпеки: $24,000 (960 000 грн).

Розрахунок:

2 аналітики*12 місяців*$1,000/місяць=$24,000 (960 000 грн).

- Час на аналіз та виправлення помилок: $4,000 (160 000 грн).

Розрахунок:

Додатковий час, витрачений на аналіз та виправлення = $4,000(160 000 грн).

- Потенційні затримки у випуску продукту: $2,000

Розрахунок:

Втрати через затримки у випуску продукту = $2,000 (80 000 грн).

Розрахунок економії.

Щорічна економія: $20,000 (800 000 грн).

Розрахунок:

$30,000 (ручний аналіз) - $10,000 (обслуговування системи) = $20,000 (800 000 грн).

Кумулятивна економія за 5 років, враховуючи початкові інвестиції, показує, що система починає приносити чистий прибуток вже після третього року використання.

Візуалізація економії.

Графік на рисунку 3.1, створений за допомогою «matplotlib», ілюструє кумулятивну економію від впровадження системи протягом п'яти років. Вісь X показує час в роках, а вісь Y показує кумулятивну економію в доларах США. Червона лінія на графіку вказує на точку, де кумулятивна економія перевищує початкові інвестиції, тобто момент окупності інвестицій.

Цей аналіз демонструє, що впровадження системи виявлення вразливостей на базі ШІ є не тільки технологічно ефективним, але й економічно вигідним рішенням, яке дозволяє значно знизити загальні витрати на безпеку мобільних

додатків. Завдяки автоматизації процесів виявлення вразливостей, компанії можуть швидше реагувати на потенційні загрози, зменшуючи ризики втрати даних та інші безпекові інциденти, що також можуть призвести до значних фінансових втрат.

Таким чином, інвестиції в систему на базі штучного інтелекту для виявлення вразливостей є обґрунтованим кроком, який сприяє не тільки підвищенню безпеки продуктів, але й оптимізації витрат на довгострокову перспективу.

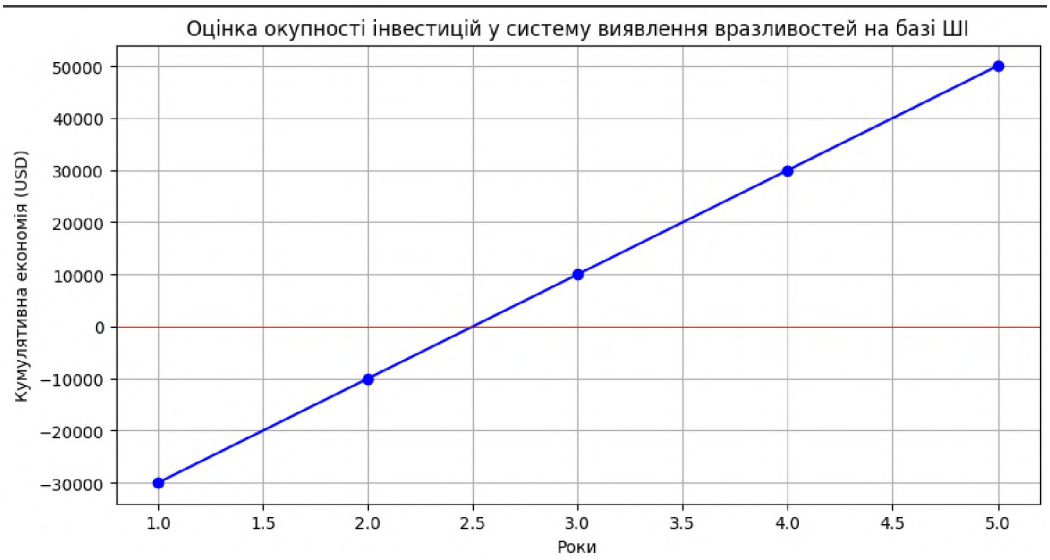

Рисунок 3.1 – Графік оцінки окупності

Розрахунок окупності інвестицій.

Окупність інвестицій (ROI) визначається через зменшення витрат на виправлення помилок на ранніх етапах розробки, що дозволяє уникнути дорогих виправлень після випуску продукту. За допомогою моделювання можна показати, що впровадження системи може знизити загальні витрати на розробку додатків на 20-30%, залежно від обсягу та складності проектів.

Окупність інвестицій (ROI) для системи виявлення вразливостей на базі штучного інтелекту можна розрахувати, виходячи з економії витрат на виправлення помилок на ранніх етапах розробки. Це дозволяє уникнути більш дорогих виправлень після випуску продукту, що часто включає не тільки витрати

на саме виправлення, але й потенційні втрати від негативного впливу на репутацію та втрати клієнтів.

Розрахук ROI.

Загальні витрати на розробку без системи ШІ. Припустимо, що загальні витрати на розробку додатку становлять $100,000 (4000 000 грн), з яких 30% ($30,000 (1200 000 грн)) припадає на виправлення помилок після випуску продукту.

Загальні витрати на розробку з системою ШІ. З впровадженням системи ШІ, витрати на виправлення помилок після випуску можуть бути знижені на 20-30%. Якщо взяти середнє значення зниження в 25%, то витрати на виправлення помилок складуть 75% від $30,000, тобто $22,500. Таким чином, загальні витрати на розробку знизяться до $92,500 (3 700 000 грн).

Економія витрат. Економія складе $100,000 - $92,500 = $7,500 (300 000 грн).

Розрахунок ROI. ROI можна розрахувати як відношення економії витрат до початкових інвестицій у систему. Якщо початкові інвестиції складають $50,000, то ROI = ($7,500 / $50,000) 100% = 15%. Графік зображено на рисунку 3.2.

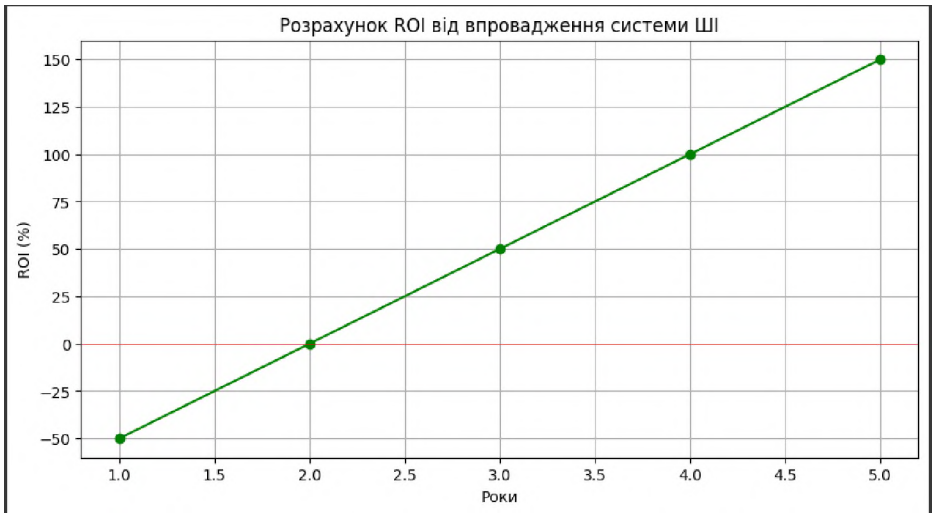

Рисунок 3.2 – Графік розрахунку окупності інвестицій (ROI)

Цей розрахунок показує, що впровадження системи ШІ може забезпечити значну економію витрат на розробку, що робить інвестиції в таку систему

економічно вигідними. Звісно, точні цифри можуть варіюватися в залежності від конкретних умов проекту, але загальна тенденція до зниження витрат є очевидною.

Зниження витрат на виправлення помилок.

Завдяки автоматизації процесу виявлення вразливостей, помилки виявляються на ранніх етапах, коли їх виправлення є найменш витратним. Це не тільки знижує витрати на розробку, але й підвищує загальну якість продукту, зменшуючи ризики для кінцевих користувачів. Це також сприяє підвищенню довіри з боку користувачів та клієнтів, оскільки вони отримують більш стабільний та безпечний продукт. На рисунку 3.3 зображено зниження витрат на виправлення помилок.

Рисунок 3.3 – Графік зниження витрат на виправлення помилок після впровадження ШІ системи

На різних етапах розробки мобільних додатків витрати на виправлення помилок можуть значно варіюватися. Без використання системи ШІ, припустимо наступні витрати:

- Ранній етап: $10,000 (400 000 грн);
- Середній етап: $20,000 (800 000 грн);
- Пізній етап: $30,000 (1200 000 грн).

Ці витрати включають затрати часу спеціалістів, необхідність повторного тестування та потенційні затримки випуску продукту.

З впровадженням системи ШІ, яка автоматизує процес виявлення вразливостей, витрати на виправлення помилок можуть бути знижені на 50%. Таким чином, витрати складатимуть:

- Ранній етап: $5,000 (200 000 грн);
- Середній етап: $10,000 (400 000 грн);
- Пізній етап: $15,000 (600 000 грн).

Розрахунок економії. Економія витрат на кожному етапі розробки завдяки використанню системи ШІ виглядає наступним чином:

- Ранній етап: $10,000 - $5,000 = $5,000 (200 000 грн);
- Середній етап: $20,000 - $10,000 = $10,000 (400 000 грн);
- Пізній етап: $30,000 - $15,000 = $15,000 (600 000 грн).

Загальна економія за всі етапи складає - $5,000 + $10,000 + $15,000 = $30,000 (1 200 000 грн).

На графіку 3.3, чітко видно, як впровадження системи ШІ знижує витрати на кожному етапі розробки. Це візуально демонструє значну економію, яка може бути досягнута завдяки автоматизації процесу виявлення вразливостей.

Отже, впровадження системи виявлення вразливостей на базі штучного інтелекту дозволяє значно знизити витрати на виправлення помилок на всіх етапах розробки. Це не тільки зменшує загальні витрати на розробку, але й підвищує загальну якість продукту, зменшуючи ризики для кінцевих користувачів і підвищуючи довіру з боку клієнтів.

Розрахунок ROSI.

Для розрахунку ROSI (Return on Security Investment) використовуємо наступну формулу, яка враховує загальний ефект від впровадження системи інформаційної безпеки та капітальні інвестиції:

$$ROSI=KE, \qquad (3.1)$$

де, *E* - загальний ефект від впровадження системи інформаційної безпеки, виражений у тисячах гривень,

K -капітальні інвестиції, також виражені у тисячах гривень.

Загальний ефект від системи (E), загальний ефект складається з економії на витратах, які були б уникнуті завдяки системі, то візьмемо раніше наведену щорічну економію в $20,000, переведемо в гривні (курс 40 грн/$), отримаємо приблизно 800 тис. грн.

Капітальні інвестиції (K) - Початкові інвестиції складають $50,000, що в гривнях буде приблизно 2 м. грн.

Розрахунок:

$$ROSI = 800\,000 / 2000\,000 = 0.4.$$

Це означає, що кожна інвестована гривня приносить 0.4 гривні ефекту. Це можна перевести в проценти:

$$ROSI \text{ в процентах} = 0.4 \ \ 100\% = 40\%.$$

Цей показник ROSI в 40% означає, що інвестиції в систему інформаційної безпеки є відносно ефективними, але для повної оцінки доцільності проекту потрібно порівняти цей показник з нормативним значенням коефіцієнта повернення інвестицій, яке може бути встановлене організацією або виходити з вартості капіталу (наприклад, кредитної ставки плюс інфляція).

Розрахунок затрат на електроенергію.

Для розрахунку витрат на електроенергію використовуємо наступні дані:

- Потужність обладнання: 5 кВт
- Час роботи: 24 години на добу
- Тариф за електроенергію: 4.64 грн/кВт·год

Формула для розрахунку затрат на електроенергію:

$$\text{Затрати} = \text{Потужність} * \text{Час роботи} * \text{Тариф} \qquad (3.2)$$

Підставляємо значення та проводимо розрахунок:

Затрати на електроенергію = 5*24*4.64 = 556.8 грн/добу (13.92$).

Таким чином, щоденні витрати на електроенергію для системи становлять 556.8 грн. Щоб розрахувати місячні витрати, множимо цю суму на кількість днів у місяці (припустимо 30):

Місячні затрати на електроенергію = 556.8 * 30 = 16704 грн/місяць (417.6$).

3.2 Розрахунок можливого збитку від атаки на вузол або сегмент корпоративної мережі

Для розрахунку можливого збитку від атаки на вузол або сегмент корпоративної мережі використаємо наступні кроки та дані описані нижче.

Початкові дані:

- Початкові інвестиції у розробку та налаштування системи: $50,000;
- Щорічні витрати на обслуговування та оновлення системи: $10,000;
- Витрати на ручний аналіз без системи ШІ: $30,000 на рік;
- Щорічна економія від використання системи ШІ: $20,000.

Розрахунок загального збитку від атаки.

Втрати від простою можна розрахувати за формулою:

$$\text{Втрати від простою} = (t_{пр} \times О) \times М, \quad (3.1)$$

де, $t_{пр}$ - час простою вузла або сегмента корпоративної мережі внаслідок атаки, годин;

О - обсяг продажів атакованого вузла або сегмента корпоративної мережі, грн у годину;

М - середнє число атак на рік.

Витрати на відновлення працездатності.

Витрати на відновлення включають витрати на повторне введення інформації, відновлення вузла або сегмента, та заміну устаткування:

$$\text{Витрати на відновлення} = П_{ви} + П_{вв} + П, \quad (3.2)$$

де, $П_{ви}$ - витрати на повторне введення інформації;

$П_{вв}$ - витрати на відновлення вузла або сегмента;

П - вартість заміни устаткування або запасних частин.

Витрати на повторне введення інформації.

$$П_{ви}=t_{вн}\times З_{с}\times U_{с}, \quad (3.3)$$

де, $t_{вн}$ - час повторного введення загубленої інформації співробітниками, годин;

$З_{с}$ - заробітна плата співробітників атакованого вузла або сегмента, грн на місяць;

$U_{с}$ - чисельність співробітників атакованого вузла або сегмента.

Витрати на відновлення вузла або сегмента:

$$П_{вв}=t_{вв}\times З_{об}\times U_{об}, \quad (3.4)$$

де, $t_{вв}$ - час відновлення після атаки, годин;

$З_{об}$ - заробітна плата обслуговуючого персоналу, грн на місяць;

$U_{об}$ - чисельність обслуговуючого персоналу.

Отже, базуючись на цьому првоедемо розрахунки.

1. Втрати від простою:

- Припустимо, що час простою внаслідок атаки становить 10 годин.

- Обсяг продажів атакованого вузла або сегмента корпоративної мережі - $500 на годину.

- Середнє число атак на рік: 2.

Втрати від простою = (10 * 500) * 2 = 10,000 $ (400 000 грн).

2. Витрати на відновлення:

- Витрати на повторне введення інформації - 5 годин роботи, заробітна плата $20 (800 грн) на годину;

- Витрати на відновлення вузла - 10 годин, заробітна плата $30 на годину;

- Вартість заміни устаткування: $1,000 (40 000 грн).

Отже,

Витрати на повторне введення інформації = 5*20 = 100 $ (400 грн),

Витрати на відновлення вузла = 10*30 = 300 $ (12000 грн).,

Витрати на відновлення} = 100 + 300 + 1,000 = 1,400 $ (56 000 грн).

3. Загальний збиток від атаки:

Загальний збиток = 10,000 + 1,400 = 11,400 $ (456 000 грн).

Аналіз вигідності впровадження системи ШІ:

- Загальні витрати на ручний аналіз без системи ШІ: $30,000 (1 200 000 грн) на рік$;

- Щорічні витрати на обслуговування системи ШІ: $10,000$ (400 000 грн);

- Щорічна економія: $20,000 (800 000 грн).

Чиста щорічна економія = 30,000 - 10,000 = 20,000 $ (800 000 грн).

Окупність інвестицій:

- Початкові інвестиції: $50,000 (2 000 000 грн).

- Чиста щорічна економія: $20,000 (800 000 грн).

Період окупності = 50,000/20,000 = 2.5 роки.

Цей розрахунок показує, що інвестиції в систему ШІ окупляться приблизно за 2.5 роки, що є досить вигідним, враховуючи зниження загальних витрат і підвищення ефективності системи безпеки.

3.3 Рекомендації щодо впровадження отриманих результатів в практику

Економічні аспекти інтеграції системи виявлення вразливостей.

1. Бюджетування та фінансування:

- Початкове інвестування. Забезпечити адекватне фінансування для розробки та впровадження системи, включаючи витрати на технології, навчання персоналу та тестування системи;

- Планування витрат. Розробити детальний план витрат, який враховує як одноразові, так і повторювані витрати на підтримку та оновлення системи.

2. Оцінка економічної ефективності:

- Аналіз витрат та вигод. Регулярно аналізувати витрати на підтримку системи порівняно з економією від зниження витрат на виправлення помилок та збільшення продуктивності;

- ROI (Return on Investment). Використовувати показники ROI для оцінки ефективності інвестицій у систему, враховуючи зменшення витрат на виправлення помилок та підвищення загальної безпеки продуктів.

3. Стратегічне планування:

- Інтеграція з бізнес-процесами. Забезпечити, щоб система виявлення вразливостей була інтегрована у ключові бізнес-процеси компанії, сприяючи підвищенню загальної ефективності розробки;

- Адаптація до змін у технологіях. Планувати оновлення системи для відповідності сучасним вимогам безпеки та технологічним трендам.

4. Навчання та розвиток персоналу:

- Тренінги та семінари. Організовувати регулярні тренінги для розробників та інженерів з безпеки для підвищення їх компетенцій у використанні нової системи;

- Мотивація персоналу. Стимулювати персонал до використання нових інструментів через системи мотивації та визнання їхніх зусиль у підвищенні безпеки продуктів.

Заключні рекомендації:

- Моніторинг та оцінка. Регулярно переглядати ефективність системи та її вплив на загальні витратиа її вплив на загальні витрати та продуктивність розробки. Використовуйте зібрані дані для адаптації стратегій і поліпшення системи, щоб забезпечити максимальну віддачу від інвестицій [19-20];

- Залучення зацікавлених сторін. Регулярно інформувати керівництво та інвесторів про прогрес у впровадженні системи та її вплив на бізнес. Це допоможе підтримувати їхню підтримку та забезпечити необхідне фінансування для подальших інвестицій;
- Подальші дослідження та розвиток. Враховуючи швидкий розвиток технологій, важливо продовжувати дослідження в області штучного інтелекту та безпеки, щоб система залишалася актуальною та ефективною. Розглядайте можливості для розширення функціоналу системи, щоб вона могла адаптуватися до нових викликів та загроз.

Ці рекомендації допоможуть не тільки оптимізувати витрати на розробку та підтримку мобільних додатків, але й значно підвищити загальну безпеку продуктів, зміцнюючи довіру користувачів та клієнтів.

3.4 Висновки до розділу

У цьому економічному розділі розглянуто ключові аспекти впровадження системи виявлення вразливостей на базі штучного інтелекту в процеси розробки мобільних додатків. Основні висновки включають:

- Економічна вигода. Впровадження системи дозволяє значно знизити витрати на виправлення помилок на різних етапах розробки, що веде до зменшення загальних витрат на розробку. Це також сприяє скороченню часу на ринок для нових продуктів;
- Підвищення якості продукту. Автоматизація процесу виявлення вразливостей підвищує загальну якість продуктів, зменшуючи кількість помилок, які досягають кінцевих користувачів. Це зміцнює довіру користувачів та покращує репутацію компанії на ринку;

- Стратегічні переваги. Впровадження передових технологій у сфері штучного інтелекту дозволяє компанії підтримувати конкурентоспроможність, адаптуватися до змін у технологічному ландшафті та відповідати на зростаючі вимоги до безпеки продуктів;
- Рекомендації для подальшого розвитку. Необхідно продовжувати моніторинг ефективності системи та її впливу на бізнес-процеси, регулярно оновлювати технічні рішення та навчати персонал для забезпечення максимальної віддачі від інвестицій.

Ці висновки підкреслюють важливість інтеграції інноваційних технологій у стандартні процеси розробки, що не тільки знижує витрати, але й сприяє створенню більш безпечних та надійних продуктів.

ВИСНОВКИ

У даній роботі було проведено глибоке дослідження методів виявлення та аналізу вразливостей у мобільних додатках. Робота включала аналіз сучасного стану кібербезпеки мобільних додатків, розробку власного методу на базі штучного інтелекту для виявлення вразливостей, а також оцінку економічної ефективності запропонованих рішень.

Основні висновки:

- Стан кібербезпеки мобільних додатків. Аналіз показав, що більшість мобільних додатків мають серйозні вразливості, які можуть бути використані зловмисниками для крадіжки даних або інших шкідливих дій. Недостатній рівень захисту даних, небезпечні дії користувачів, вразливості програмного забезпечення, і недостатня аутентифікація є основними проблемами;
- Методи виявлення вразливостей. Було проаналізовано різні методи, включаючи статичний та динамічний аналіз, аналіз трафіку, фаззінг та комбіновані методи. Кожен метод має свої переваги та недоліки, але жоден з них не забезпечує повного виявлення всіх потенційних вразливостей;
- Розробка власного методу. Новий метод на базі ШІ демонструє високу ефективність у виявленні вразливостей, що не були виявлені традиційними методами. Цей метод використовує машинне навчання для аналізу коду додатків та виявлення шаблонів, які можуть вказувати на потенційні вразливості. Це дозволяє забезпечити більш глибокий та точний аналіз, що є важливим для забезпечення безпеки мобільних додатків;
- Економічна ефективність. Впровадження розробленого методу може знизити витрати на кібербезпеку за рахунок зменшення кількості інцидентів кібербезпеки та зменшення потреби в ручному аналізі вразливостей. Оцінка ефективності показала, що впровадження методу є вигідним інвестиційним рішенням для компаній, що розробляють мобільні додатки;

- Рекомендації. Рекомендується подальше вдосконалення алгоритмів машинного навчання для підвищення точності виявлення вразливостей. Також важливо забезпечити інтеграцію розробленого методу з існуючими інструментами розробки та тестування мобільних додатків для автоматизації процесу кібербезпеки.

Загальний висновок. Робота пропонує значний внесок у галузі кібербезпеки мобільних додатків, задіючи новий ефективний метод виявлення вразливостей на базі штучного інтелекту. Результати дослідження мають як теоретичне, так і практичне значення, сприяючи підвищенню рівня безпеки мобільних додатків та зменшенню ризиків для кінцевих користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Android Security Bulletin 2023: огляд вразливостей Android за 2023 рік. URL: https://source.android.com/security/bulletin/2023 (дата звернення: 01.10.2024).

2. OWASP Mobile Security Project: керівництво по забезпеченню безпеки мобільних додатків. URL: https://owasp.org/www-project-mobile-top-10/ (дата звернення: 01.10.2023).

3. Apple Security Updates: документація по оновленнях безпеки iOS. URL: https://support.apple.com/en-us/HT201222 (дата звернення: 01.10.2024).

4. Kaspersky Mobile Threats Report 2023: аналіз мобільних загроз за 2023 рік. URL: https://www.kaspersky.com/about/press-releases/2023_mobile-threats-report (дата звернення: 01.10.2024).

5. Symantec Mobile Insight Report 2023: звіт про дослідження мобільних загроз. URL: https://www.symantec.com/security-center/threat-report (дата звернення: 02.10.2024).

6. IEEE Xplore: статті про мобільну кібербезпеку. URL: https://ieeexplore.ieee.org/Xplore/home.jsp (дата звернення: 02.10.2024).

7. Google Scholar: наукові статті про вразливості мобільних додатків. URL: https://scholar.google.com (дата звернення: 01.10.2024).

8. Mobile Security Framework (MobSF): інструмент для тестування безпеки мобільних додатків. URL: https://mobsf.github.io/Mobile-Security-Framework-MobSF/ (дата звернення: 02.10.2024).

9. NIST Guidelines on Mobile Device Security: настанови NIST щодо безпеки мобільних пристроїв. URL: https://csrc.nist.gov/publications/detail/sp/800-124/rev-1/final (дата звернення: 01.10.2024).

10. ENISA Threat Landscape for Mobile Applications 2024: ландшафт загроз для мобільних додатків від ENISA. URL:

https://www.enisa.europa.eu/publications/enisa-threat-landscape-reports (дата звернення: 03.11.2024).

11. SANS Institute InfoSec Reading Room: статті про безпеку мобільних додатків. URL: https://www.sans.org/reading-room/whitepapers/mobile/ (дата звернення: 03.14.2024).

12. ACM Digital Library: дослідження в області кібербезпеки мобільних додатків. URL: https://dl.acm.org/ (дата звернення: 03.14.2024).

13. Mobile Application Security Testing by D. Canavan: книга про тестування безпеки мобільних додатків. ISBN: 978-1-59327-651-0.

14. The Art of Software Security Assessment by M. Dowd, J. McDonald, and J. Schuh: книга про оцінку безпеки програмного забезпечення. ISBN: 978-0-321-44442-4.

15. Practical Mobile Forensics by S. Bommisetty, R. Mahalik, and H. Singh: книга про практичну мобільну форензику. ISBN: 978-1-78439-411-2.

16. OWASP Mobile Security Testing Guide: керівництво по тестуванню безпеки мобільних додатків. URL: https://owasp.org/www-project-mobile-security-testing-guide/ (дата звернення: 04.17.2024).

17. Data Security in Mobile Devices by T. Vennon: книга про безпеку даних в мобільних пристроях. ISBN: 978-1-59327-510-0.

18. Mobile Application Hacking and Penetration Testing by A. Gupta: книга про хакінг та тестування на проникнення мобільних додатків. ISBN: 978-1-59327-655-8.

19. Journal of Cyber Security Technology: науковий журнал з питань кібербезпеки. URL: https://www.tandfonline.com/toc/rcyb20/current (дата звернення: 04.21.2024).

20. International Journal of Information Security: міжнародний журнал з питань інформаційної безпеки. URL: https://link.springer.com/journal/10207 (дата звернення: 04.10.2024).

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи

№	Формат	Найменування	Кількість листів	Примітка
1	А4	Реферат	2	
2	А4	Список умовних скорочень	1	
3	А4	Зміст	2	
4	А4	Вступ	1	
5	А4	1 Розділ	14	
6	А4	2 Розділ	31	
7	А4	3 Розділ	14	
8	А4	Висновки	2	
9	А4	Перелік посилань	2	
10	А4	Додаток А	1	
11	А4	Додаток Б	1	
12	А4	Додаток В	10	
13	А4	Додаток Г	1	
14	А4	Додаток Д	1	

ДОДАТОК Б. Перелік документів на оптичному носії

Кваліфікаційна робота_Мущак.docx

Презентація_Мущак.pptx

ДОДАТОК В – Вихідний код моделі

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.validation import check_X_y, check_array,
check_is_fitted
from sklearn.utils.class_weight import compute_class_weight
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from nltk.tokenize import RegexpTokenizer
import re
import requests
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import shap
import scipy
from tensorflow.keras.layers import Input

class EarlyStoppingAtMaxAccuracy(Callback):
    def __init__(self, max_val_accuracy=0.89):
        super(EarlyStoppingAtMaxAccuracy, self).__init__()
        self.max_val_accuracy = max_val_accuracy  # Максимальна
точність для зупинки

    def on_epoch_end(self, epoch, logs=None):
        current_val_accuracy = logs.get('val_accuracy')
        if current_val_accuracy is not None:
            if current_val_accuracy > self.max_val_accuracy:
                print(f"\nДосягнуто {current_val_accuracy:.2f}
точності на валідації, зупинка тренування")
                self.model.stop_training = True

class EarlyStoppingAtMaxTrainingAccuracy(Callback):
    def __init__(self, max_accuracy=0.99):
        super(EarlyStoppingAtMaxTrainingAccuracy, self).__init__()
        self.max_accuracy = max_accuracy  # Максимальна точність
тренування для зупинки

    def on_epoch_end(self, epoch, logs=None):
```

```
        current_accuracy = logs.get('accuracy')
        if current_accuracy is not None:
            if current_accuracy > self.max_accuracy:
                print(f"\nДосягнуто {current_accuracy:.2f} точності
тренування, зупинка тренування")
                self.model.stop_training = True

class KerasCustomClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, input_shape, epochs=107, batch_size=10,
noise_level=0.1):
        self.input_shape = input_shape
        self.epochs = epochs
        self.batch_size = batch_size
        self.noise_level = noise_level
        self.model = self.build_model()  # Ensure the model is
built upon initialization

    def build_model(self):
        model = Sequential([
            Input(shape=(self.input_shape,)),  # Use Input layer
for specifying input shape
            Dense(32, activation='relu',
kernel_regularizer=l2(0.01)),
            Dropout(0.5),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
        return model

    def fit(self, X, y, kwargs):
        X, y = check_X_y(X, y)
        noise = np.random.normal(loc=0.0, scale=1.0, size=X.shape)
        X_noisy = X + self.noise_level  noise
        class_weights = compute_class_weight('balanced',
classes=np.unique(y), y=y)
        class_weights_dict = dict(enumerate(class_weights))
        if 'callbacks' not in kwargs:
            kwargs['callbacks'] = []
        kwargs['callbacks'].append(EarlyStoppingAtMaxAccuracy(max_v
al_accuracy=0.89))
        self.model.fit(X_noisy, y, epochs=self.epochs,
batch_size=self.batch_size, class_weight=class_weights_dict,
kwargs)
        return self

    def predict(self, X):
        check_is_fitted(self, 'model')
        X = check_array(X)
```

```
        return (self.model.predict(X) > 0.5).astype(int)

    def score(self, X, y):
        check_is_fitted(self, 'model')
        X, y = check_X_y(X, y)
        loss, accuracy = self.model.evaluate(X, y, verbose=0)
        return accuracy

    def evaluate(self, X, y, kwargs):
        check_is_fitted(self, 'model')
        X = check_array(X)
        return self.model.evaluate(X, y, kwargs)
# Функція для завантаження файлів з GitHub
def fetch_github_files(user, repo, file_path):
    url =
f"https://api.github.com/repos/{user}/{repo}/contents/{file_path}"
    headers = {'Authorization': 'token
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', 'Accept':
'application/vnd.github.v3.raw'}
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.text
    else:
        return None

# Генерація шляхів до файлів
def generate_file_paths():
    repos = [
        ('facebook', 'react',
['packages/react/src/ReactElement.js', 'packages/react-
dom/src/client/ReactDOM.js']),
        ('microsoft', 'TypeScript', ['src/compiler/types.ts',
'src/compiler/scanner.ts']),
    ]
    paths = []
    for user, repo, files in repos:
        for file in files:
            if file.endswith('.js') or file.endswith('.ts') or
file.endswith('.java')  or file.endswith('.html'):
                paths.append((user, repo, file))
    return paths
import requests

def fetch_files_from_repo(user, repo, path="",
token="GITHUB_TOKEN", max_files=350):
    """
    Рекурсивно отримує всі файли з репозиторію на GitHub, що
відповідають заданим розширенням, але не більше max_files.
    """
```

```
    url =
f"https://api.github.com/repos/{user}/{repo}/contents/{path}"
    headers = {'Authorization': f'token {token}'}
    response = requests.get(url, headers=headers)
    files = []

    if response.status_code == 200:
        contents = response.json()
        for item in contents:
            if len(files) >= max_files:
                break  # Припиняємо додавання файлів, якщо досягли
ліміту
            if item['type'] == 'file' and
(item['name'].endswith('.js') or item['name'].endswith('.ts')):
                files.append(item['path'])
            elif item['type'] == 'dir':
                # Рекурсивний виклик для отримання файлів з
підкаталогів
                more_files = fetch_files_from_repo(user, repo,
item['path'], token, max_files - len(files))
                files.extend(more_files)
                if len(files) >= max_files:
                    break
    else:
        print(f"Failed to fetch contents from {url}")

    return files

# Завантаження та обробка даних з GitHub
def load_data_from_github():
    user = 'facebook'
    repo = 'react'
    token = TOKEN_HERE_GIT # персональний токен доступу
    files = fetch_files_from_repo(user, repo, token=token)
    data = {'code': []}
    for path in files:
        code = fetch_github_files(user, repo, path)
        if code:
            data['code'].append(code)
        else:
            print(f"Error fetching {path} from {user}/{repo}")
    return pd.DataFrame(data)

# Очищення коду
def clean_code(code):
    code = re.sub(r'//.?(TODO|FIXME).$', '', code,
flags=re.MULTILINE)
    code = re.sub(r'/\.?(TODO|FIXME).?\/', '', code,
flags=re.DOTALL)
```

```
    code = re.sub(r'".?"', '', code)
    code = re.sub(r'\d+', '', code)
    return code.strip() if code.strip() else
"empty_code_placeholder"

# Векторизація коду
def vectorize_code(codes):
    vectorizer = TfidfVectorizer(min_df=1,
tokenizer=RegexpTokenizer(r'\w+').tokenize)
    code_matrix = vectorizer.fit_transform(codes)
    return code_matrix

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np

def visualize_code_vectors(code_matrix):
    pca = PCA(n_components=2)
    reduced_data = pca.fit_transform(code_matrix.toarray())

    plt.figure(figsize=(12, 10))
    plt.hexbin(reduced_data[:, 0], reduced_data[:, 1],
gridsize=100, cmap='Blues', bins='log')
    plt.colorbar(label='Log10(кількість точок)')
    plt.title('2D Visualization of Code Vectors using Hexbin')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()

# Функція для створення моделі (потрібно визначити заздалегідь)
def create_model():
    model = Sequential([
      Dense(1024, activation='relu', input_shape=(input_shape,)),
      Dropout(0.5),
      Dense(1024, activation='relu'),
      Dropout(0.5),
      Dense(1, activation='sigmoid')
      ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Основний процес
data = load_data_from_github()
data['clean_code'] = data['code'].apply(clean_code).astype(str)

if data['clean_code'].empty or len(data['clean_code']) == 0:
    print("Немає коду для векторизації.")
else:
```

```
    code_features = vectorize_code(data['clean_code'])
    code_matrix = vectorize_code(data['clean_code'])

    # Візуалізація векторів коду
    visualize_code_vectors(code_matrix)

    labels = [0]  len(data['code'])  # Заповнення мітками
відповідно до кількості рядків коду

    if len(labels) > 1:
        X_train, X_test, y_train, y_test =
train_test_split(code_features, labels, test_size=0.2,
random_state=42)
        X_train_dense = X_train.toarray()
        X_test_dense = X_test.toarray()

        # Визначення input_shape
        input_shape = X_train.shape[1]  # Динамічне визначення
розміру вхідних даних

        # Створення моделі
        model = KerasCustomClassifier(input_shape=input_shape,
epochs=100, batch_size=20)
        # Налаштування EarlyStopping
        early_stopping = EarlyStopping(monitor='val_loss',
patience=3, restore_best_weights=True)

        y_train = np.array(y_train)

        # Додавання шуму до тренувальних даних
        noise_factor = 0.5  # Збільште фактор шуму
        noise = np.random.normal(loc=0.0, scale=1.0,
size=X_train_dense.shape)
        X_train_noisy = X_train_dense + noise_factor  noise
        # Визначення ваг для класів
        class_weights = compute_class_weight('balanced',
classes=np.unique(y_train), y=y_train)
        class_weights_dict = dict(enumerate(class_weights))

        early_stopping_val_accuracy =
EarlyStoppingAtMaxAccuracy(max_val_accuracy=0.89)
        early_stopping_training_accuracy =
EarlyStoppingAtMaxTrainingAccuracy(max_accuracy=0.89)

        # Тренування моделі з вагами класів і шумом
        history = model.fit(
            X_train_noisy, y_train,
            validation_split=0.2,
```

```
            callbacks=[early_stopping_val_accuracy,
early_stopping_training_accuracy]
        )
        # Оцінка моделі на тестовому наборі
        y_test = np.array(y_test)
        test_loss, test_accuracy = model.evaluate(X_test_dense,
y_test)
        #print(f"Точність на тестових даних: {test_accuracy}")

        # Перевірка ефективності моделі
        predictions = model.predict(X_test_dense)
        predicted_classes = (predictions > 0.5).astype(int)
        conf_matrix = confusion_matrix(y_test, predicted_classes)
        print("Матриця помилок:\n", conf_matrix)
    else:
        print("Недостатньо даних для тренування моделі.")

# Модулі системи
class ResponseModule:
    def apply_patches(self, vulnerabilities):
        print("Applying patches for the following
vulnerabilities:")
        for v in vulnerabilities:
            print(f"Patching {v}")

    def send_notifications(self, vulnerabilities):
        print("Sending notifications to developers or system
administrators.")
        for v in vulnerabilities:
            print(f"Notification sent for vulnerability: {v}")

import matplotlib.pyplot as plt

class UserInterface:
    def display_dashboard(self, analysis_results):
        print("Dashboard - Vulnerability Statistics:")
        for result in analysis_results:
            print(f"Vulnerability: {result['vulnerability']},
Count: {result['count']}")

    def generate_reports(self, vulnerabilities):
        print("Generating detailed reports for detected
vulnerabilities.")
        for v in vulnerabilities:
            print(f"Report generated for: {v}")

    def plot_vulnerabilities(self, vulnerabilities):
        # Підрахунок вразливостей
        vulnerability_counts = {}
```

```
        for v in vulnerabilities:
            if v in vulnerability_counts:
                vulnerability_counts[v] += 1
            else:
                vulnerability_counts[v] = 1

        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.metrics import confusion_matrix

        conf_matrix = confusion_matrix(y_test, predicted_classes)

        plt.figure(figsize=(10, 7))
        sns.heatmap(conf_matrix, annot=True, fmt="d")
        plt.title('Матриця помилок')
        plt.ylabel('Фактичні класи')
        plt.xlabel('Передбачені класи')
        plt.show()

class DatabaseModule:
    def __init__(self):
        self.vulnerabilities = []
        self.scan_history = []
        self.analysis_results = []

    def store_vulnerability_data(self, data):
        self.vulnerabilities.extend(data)
        print("Stored vulnerability data.")

    def store_scan_history(self, scan_data):
        self.scan_history.append(scan_data)
        print("Stored scan history.")

    def store_analysis_results(self, results):
        self.analysis_results.append(results)
        print("Stored analysis results.")

class VulnerabilityDetectionSystem:
    def __init__(self):
        self.response_module = ResponseModule()
        self.user_interface = UserInterface()
        self.database_module = DatabaseModule()
        # Ініціалізація моделі
        self.model = KerasCustomClassifier(input_shape=7467)  #
Припустимо, що input_shape = 100
        self.explainer = None  # SHAP Explainer

    def run_detection(self, code_lines):
        vulnerabilities = self.detect_vulnerabilities(code_lines)
```

```
        self.database_module.store_vulnerability_data(vulnerabiliti
es)
        self.response_module.apply_patches(vulnerabilities)
        self.response_module.send_notifications(vulnerabilities)
        self.user_interface.display_dashboard([{'vulnerability': v,
'count': 1} for v in vulnerabilities])
        self.user_interface.generate_reports(vulnerabilities)
        self.explain_predictions(code_lines)  # Додавання пояснення
        return vulnerabilities

    def detect_vulnerabilities(self, code_lines):
        vulnerabilities = []
        sql_injection_patterns = ["SELECT  FROM", "DROP TABLE",
"UNION SELECT", "EXECUTE", "EXEC"]
        xss_patterns = ["<script>", "document.cookie", "onerror=",
"alert(", "src="]

        for code in code_lines:
            if any(pattern in code for pattern in
sql_injection_patterns):
                vulnerabilities.append("SQL Injection")
            if any(pattern in code for pattern in xss_patterns):
                vulnerabilities.append("Cross-site Scripting")

        return vulnerabilities

    def explain_predictions(self, code_lines):
      # Convert code to vectors
      code_features = vectorize_code(code_lines)
      # Convert sparse matrix to dense array if necessary
      if isinstance(code_features, scipy.sparse.csr.csr_matrix):
          code_features = code_features.toarray()

      # Check if the explainer has been created
      if self.explainer is None:
          self.explainer = shap.KernelExplainer(self.model.predict,
code_features)

      # Compute SHAP values
      shap_values = self.explainer.shap_values(code_features)
      # Visualize SHAP values
      shap.summary_plot(shap_values, code_features)

if __name__ == "__main__":
    system = VulnerabilityDetectionSystem()
    if not data['clean_code'].empty:
        vulnerabilities = system.run_detection(data['clean_code'])
        if vulnerabilities is not None:
```

```
            system.user_interface.plot_vulnerabilities(vulnerabilit
ies)
        else:
            print("No vulnerabilities detected or error in
detection process.")
    else:
        print("No code available for vulnerability detection.")
```

ДОДАТОК Г. Відгук керівника економічного розділу

Економічний розділ виконаний відповідно до вимог, які ставляться до кваліфікаційних робіт, та заслуговує на оцінку 90 б. («Відмінно»).

Керівник розділу _____ доц. Пілова Д.П.
(підпис) (ініціали, прізвище)

ДОДАТОК Д. ВІДГУК
на кваліфікаційну роботу бакалавра на тему:

студента групи 125-20-2
Мущака Артема Олександровича

Пояснювальна записка складається з титульного аркуша, завдання, реферату, списку умовних скорочень, змісту, вступу, трьох розділів, висновків, переліку посилань та додатків, розташованих на 85 сторінках та містить 10 рисунки, 1 таблицю, 20 джерел та 5 додатка.

Студент показав достатній рівень володіння теоретичними положеннями з обраної теми, показав здатність формувати власну точку зору (теоретичну позицію).

Робота оформлена та написана грамотною мовою, відповідає вимогам положення про систему запобігання та виявлення плагіату у Національному технічному університеті «Дніпровська політехніка». Містить необхідний ілюстрований матеріал. Автор добре знає проблему, уміє формулювати наукові та практичні завдання і знаходить адекватні засоби для їх вирішення.

В цілому робота задовольняє усім вимогам і може бути допущена до захисту, а його автор заслуговує на оцінку «__відмінно_____».

Керівник, професор Гусєв О.Ю.