Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

«Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту»

Виконав: студент 4 курсу, гр. 2КІТС-20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем

Заремблюк Д. О.

Керівник: доц. каф. МБІС
Шиян А. А.

«6» червня 2024 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Черняк О. І.

«6» червня 2024 р.

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

Вінниця ВНТУ – 2024

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

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

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 22 " березня 2024 р.

З А В Д А Н Н Я
на бакалаврську дипломну роботу студенту
Заремблюку Дмитру Олександровичу

1. Тема роботи «Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту»

Керівник роботи доц. Шиян Анатолій Антонович

затверджені наказом вищого навчального закладу від " 11 " березня 2024року № 80
2. Термін подання студентом роботи 15.06.2024 р.
3. Вихідні дані до роботи Набір даних моніторингу активності користувачів для навчання моделі машинного навчання. Електронні джерела по темі.
4. Зміст текстової частини:
1. Теоретичні основи використання хмарних сховищ та штучного інтелекту
2. Проектування системи захищеної віддаленої роботи
3. Розроблення програмного додатку та його тестування
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Презентація у форматі PowerPoint з ілюстраціями архітектури програми, скріншотами інтерфейсу та результатами тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	доц. Шиян А. А.		
Розділ II	доц. Шиян А. А.		
Розділ III	доц. Шиян А. А.		

7. Дата видачі завдання 22 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Ознайомлення з темою та постановкою завдання	22.03.2024	24.03.2024	
2	Пошук та вивчення літератури за темою дослідження	25.03.2024	31.03.2024	
3	Аналіз існуючих підходів до виявлення зловмисних посилань	01.04.2024	07.04.2024	
4	Обгрунтування вибору методу KNN та розроблення підходу	08.04.2024	14.04.2024	
5	Вибір мови програмування та середовища розробки	15.04.2024	21.04.2024	
6	Розроблення інтерфейсу програмного модуля	22.04.2024	28.04.2024	
7	Реалізація функціоналу виявлення зловмисних посилань	29.04.2024	05.05.2024	
8	Тестування програмного модуля	06.05.2024	12.05.2024	
9	Оформлення текстової частини роботи	13.05.2024	22.05.2024	
10	Переддипломний захист			
11	Захист бакалаврської дипломної роботи			

Студент _____ Заремблюк Д. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шиян А. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана бакалаврська дипломна робота присвячена розробці програми для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту. Метою дослідження є створення ефективного механізму забезпечення безпеки та конфіденційності даних під час віддаленої роботи.

У дипломній роботі розглядаються сучасні методи та технології захисту даних у хмарних сховищах. Особлива увага приділяється використанню алгоритмів штучного інтелекту для виявлення загроз та аномалій у поведінці користувачів, що забезпечує додатковий рівень безпеки. Основні компоненти системи включають аутентифікацію, авторизацію, шифрування даних та моніторинг активності користувачів.

Розроблена програма забезпечує високий рівень захисту даних, використовуючи методи машинного навчання для аналізу користувацької активності та виявлення потенційних загроз. Програма також включає функції шифрування даних для забезпечення конфіденційності та цілісності інформації, а також механізми аутентифікації та авторизації для контролю доступу до даних.

Ключові слова: захищена віддалена робота, хмарне сховище, штучний інтелект, конфіденційність, цілісність даних, аутентифікація, авторизація, кібербезпека.

ABSTRACT

This bachelor's thesis is devoted to the development of an application for secure remote work of a user in the cloud storage using artificial intelligence. The aim of the research is to create an effective mechanism for ensuring data security and confidentiality during remote work.

The thesis examines modern methods and technologies for data protection in cloud storage. Particular attention is paid to the use of artificial intelligence algorithms to detect threats and anomalies in user behaviour, which provides an additional level of security. The main components of the system include authentication, authorisation, data encryption and monitoring of user activity.

The developed software provides a high level of data protection by using machine learning methods to analyse user activity and identify potential threats. The application also includes data encryption functions to ensure the confidentiality and integrity of information, as well as authentication and authorisation mechanisms to control access to data.

Keywords: secure remote work, cloud storage, artificial intelligence, confidentiality, data integrity, authentication, authorisation, cybersecurity.

ЗМІСТ

ВСТУП

У сучасному цифровому світі, де інформація є одним з найцінніших ресурсів, питання безпеки даних стає надзвичайно важливим. Використання хмарних технологій значно спрощує доступ до даних і дозволяє працювати віддалено, але також створює нові виклики у сфері кібербезпеки. З розвитком технологій і збільшенням обсягу переданої інформації питання захисту даних у хмарному середовищі стає ще більш актуальним. Використання штучного інтелекту для захисту даних у хмарному середовищі є перспективним напрямком, оскільки забезпечує автоматизоване виявлення загроз і підвищення рівня безпеки. Завдяки цьому забезпечується надійний захист даних від несанкціонованого доступу, що є критично важливим для сучасних інформаційних систем.

Метою даної дипломної роботи є розробка програми, яка забезпечує захищену віддалену роботу користувача у хмарному сховищі з використанням штучного інтелекту. Для досягнення цієї мети було поставлено такі завдання:

1. Провести огляд існуючих методів захисту хмарних сховищ.

2. Дослідити можливості застосування штучного інтелекту для забезпечення безпеки даних.

3. Розробити архітектуру системи, що поєднує хмарні технології та алгоритми штучного інтелекту.

4. Реалізувати програму і протестувати її функціональність та безпеку.

5. Оцінити ефективність розробленого рішення.

Об'єктом дослідження є хмарні технології та методи кібербезпеки. Предметом дослідження є розробка програми для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту.

Для досягнення поставлених завдань використовувалися наступні методи дослідження: аналіз літературних джерел, проектування архітектури програмного забезпечення, розробка алгоритмів на основі штучного інтелекту, програмування, тестування та аналіз результатів.

Новизна роботи полягає у розробці комплексного додатку, який поєднує хмарні технології та штучний інтелект для забезпечення безпеки даних. Запропонований підхід дозволяє автоматизувати процеси виявлення та реагування на загрози, що значно підвищує рівень захисту інформації. Розроблений додаток демонструє інноваційний підхід до забезпечення кібербезпеки у хмарному середовищі.

Практична цінність роботи полягає у забезпеченні надійного захисту даних користувачів, які працюють віддалено, шляхом застосування штучного інтелекту для автоматичного виявлення та реагування на кіберзагрози. Використання розробленої програми дозволить організаціям знизити ризики, пов'язані з витоком конфіденційної інформації, та підвищити загальний рівень безпеки інформаційних систем.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ВИКОРИСТАННЯ ХМАРНИХ СХОВИЩ ТА ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Огляд методів захисту хмарних сховищ

Хмарні сховища стали невід'ємною частиною сучасної інформаційної інфраструктури, забезпечуючи зручний і масштабований доступ до даних. Однак, зростання обсягів даних і кількості користувачів підвищує ризики несанкціонованого доступу та інших кіберзагроз. У цьому контексті важливо розглянути основні методи захисту, які застосовуються для забезпечення безпеки даних у хмарних сховищах.

Шифрування даних є однією з найефективніших і найбільш широко використовуваних технологій захисту інформації. Суть методу полягає у перетворенні даних у зашифрований формат, який може бути прочитаний тільки за наявності відповідного ключа дешифрування. Шифрування може бути реалізовано на різних рівнях: на стороні клієнта, на стороні сервера та під час передачі даних (транзитне шифрування). Кожен з цих підходів має свої переваги та недоліки. Шифрування на стороні клієнта забезпечує високу конфіденційність, оскільки дані шифруються перед відправкою на сервер. Це означає, що навіть у випадку компрометації сервера дані залишаються недоступними без ключа дешифрування. Однак, це вимагає відповідної інфраструктури та програмного забезпечення на стороні користувача. Шифрування на стороні сервера здійснюється після отримання даних від клієнта і забезпечує захист даних під час їх зберігання. Цей метод зручний у використанні, оскільки не вимагає додаткових зусиль від користувачів, але вимагає високого рівня довіри до постачальника хмарних послуг.

Транзитне шифрування захищає дані під час їх передачі між клієнтом і сервером, використовуючи протоколи шифрування, такі як TLS (Transport Layer Security). Це забезпечує захист від перехоплення даних під час їх

передачі через мережу. Існують різні алгоритми шифрування, серед яких AES (Advanced Encryption Standard) є найбільш популярним завдяки своїй надійності та ефективності. Ефективність шифрування значною мірою залежить від управління ключами шифрування, яке включає їх генерацію, зберігання, розповсюдження і знищення. Неправильне управління ключами може призвести до компрометації всієї системи безпеки.

Багатофакторна аутентифікація (MFA) є ще одним критично важливим методом захисту хмарних сховищ. MFA вимагає від користувача підтвердити свою особу за допомогою кількох незалежних факторів, що знижує ймовірність несанкціонованого доступу навіть у разі компрометації одного з факторів. Основні фактори аутентифікації включають щось, що користувач знає (наприклад, пароль або ПІН-код), щось, що користувач має (наприклад, мобільний телефон для отримання одноразового коду або апаратний токен) та щось, чим користувач є (біометричні дані, такі як відбитки пальців, розпізнавання обличчя або голосу).

Впровадження MFA значно підвищує безпеку, оскільки для отримання доступу до даних потрібно успішно пройти декілька рівнів перевірки. Водночас, це може створити певні незручності для користувачів, тому важливо збалансувати безпеку і зручність використання. Наприклад, використання біометричних даних може спростити процес аутентифікації, але вимагає наявності відповідного обладнання та програмного забезпечення.

Системи управління доступом (Access Control) визначають, які користувачі мають доступ до яких даних і які дії вони можуть виконувати. Існує кілька моделей управління доступом, серед яких дискреційне управління доступом (DAC), мандатне управління доступом (MAC) та рольове управління доступом (RBAC). Дискреційне управління доступом (DAC) дозволяє власнику ресурсу визначати, хто має до нього доступ і які дії можуть виконуватися. Ця модель є гнучкою, але менш безпечною,

оскільки користувачі можуть передавати свої права доступу іншим. Мандатне управління доступом (MAC) визначає доступ до ресурсів на основі політик безпеки, встановлених адміністратором, що забезпечує високий рівень безпеки, оскільки права доступу не можуть бути змінені користувачами.

Рольове управління доступом (RBAC) визначає права доступу на основі ролей, призначених користувачам. Це дозволяє ефективно керувати доступом у великих організаціях, де користувачі можуть виконувати різні ролі. Управління доступом може бути реалізоване за допомогою різних технологій і інструментів, таких як системи ідентифікації та автентифікації (IAM), що забезпечують централізоване управління доступом.

Моніторинг активності є важливим компонентом забезпечення безпеки хмарних сховищ, оскільки дозволяє виявляти та реагувати на підозрілу активність у режимі реального часу. Основні аспекти моніторингу активності включають журналювання подій, аналіз поведінки користувачів (UBA) та використання систем виявлення вторгнень (IDS). Журналювання подій записує всі значущі події, такі як спроби входу, зміни у даних, доступ до конфіденційної інформації. Це дозволяє відстежувати дії користувачів і виявляти аномалії. Аналіз поведінки користувачів використовує алгоритми машинного навчання для аналізу поведінки користувачів і створення профілів нормальної активності, де відхилення від цих профілів можуть свідчити про потенційні загрози.

Системи виявлення вторгнень аналізують трафік мережі та активність користувачів для виявлення підозрілих дій, що можуть свідчити про спроби несанкціонованого доступу або атаки. Моніторинг активності є ключовим елементом у системах захисту, оскільки дозволяє своєчасно виявляти та реагувати на загрози, забезпечуючи таким чином проактивний підхід до кібербезпеки.

Крім основних методів, існує багато інших технологій та підходів, що використовуються для захисту даних у хмарних сховищах. Сегментація мережі розділяє мережу на ізольовані сегменти, що знижує ризик розповсюдження атак у разі компрометації одного з сегментів. Захист від DDoS-атак включає використання спеціалізованих сервісів та технологій для виявлення і нейтралізації розподілених атак відмови у обслуговуванні, що можуть призвести до недоступності сервісів. Резервне копіювання даних забезпечує створення копій даних, які можуть бути використані для відновлення у разі втрати або пошкодження основних даних. Це є важливим заходом для забезпечення безперервності бізнесу.

Огляд сучасних методів захисту хмарних сховищ показав, що існує широкий спектр технологій та підходів, які використовуються для забезпечення безпеки даних. Основними методами є шифрування, багатофакторна аутентифікація, системи управління доступом та моніторинг активності. Кожен з цих методів має свої переваги та недоліки і відіграє важливу роль у забезпеченні конфіденційності, цілісності та доступності даних.

Шифрування даних є базовим методом, який забезпечує захист інформації від несанкціонованого доступу навіть у випадку перехоплення даних. Багатофакторна аутентифікація значно підвищує рівень безпеки, вимагаючи декількох незалежних факторів для підтвердження особи користувача. Системи управління доступом дозволяють контролювати, хто і до яких даних має доступ, що є критично важливим для запобігання внутрішнім загрозам. Моніторинг активності забезпечує постійний нагляд за діями користувачів та систем, дозволяючи виявляти та реагувати на підозрілі дії в режимі реального часу.

Аналіз сучасних рішень у сфері захисту хмарних сховищ показує, що багато з них фокусуються на традиційних методах, які, хоча і є ефективними, не завжди можуть вчасно виявити нові, більш складні

загрози. Впровадження інноваційних рішень, зокрема з використанням штучного інтелекту, дозволяє створювати більш адаптивні та динамічні системи захисту, які здатні оперативно реагувати на нові виклики у сфері кібербезпеки.

Таким чином, ефективний захист хмарних сховищ вимагає комбінованого підходу, який включає використання як традиційних методів, так і сучасних технологій штучного інтелекту для створення багаторівневих та інтегрованих систем захисту. Це дозволяє забезпечити високий рівень безпеки даних та підтримувати довіру користувачів до хмарних технологій..

1.2 **Аналіз технологій штучного інтелекту для забезпечення безпеки**

У сучасному світі, де обсяги цифрової інформації зростають експоненціально, кіберзагрози стають все більш складними та непередбачуваними. З огляду на це, традиційні методи захисту часто виявляються недостатньо ефективними. Штучний інтелект (ШІ) пропонує нові можливості для підвищення рівня кібербезпеки, забезпечуючи швидкий та точний аналіз даних, виявлення загроз і автоматизацію захисних заходів. У цьому розділі ми детально розглянемо основні методи та алгоритми ШІ, що застосовуються у сфері кібербезпеки, їх переваги та приклади практичного використання.

Методи штучного інтелекту в кібербезпеці

Штучний інтелект застосовує різні методи для аналізу великих обсягів даних і виявлення кіберзагроз. Одними з найважливіших методів є машинне навчання (ML), глибоке навчання (DL) та обробка природної мови (NLP). Кожен з цих методів має свої особливості та області застосування, які роблять його ефективним інструментом у боротьбі з кіберзлочинністю.

Машинне навчання є основою багатьох рішень у кібербезпеці. Воно дозволяє системам самостійно вчитися на основі аналізу історичних даних

і покращувати свої показники з часом. Основними підходами у машинному навчанні є навчання з учителем, навчання без учителя та напівкероване навчання.

Метод навчання з учителем (Supervised Learning) використовує попередньо мічені дані для навчання моделей. Він ефективний для задач класифікації, таких як виявлення шкідливого програмного забезпечення (малваре) або фішингових атак. Наприклад, модель може навчитися розпізнавати ознаки шкідливого файлу на основі великої кількості прикладів шкідливих і нешкідливих файлів.

Метод навчання без учителя (Unsupervised Learning) аналізує дані без попередньої класифікації, що дозволяє виявляти аномалії та нові загрози. Зокрема, алгоритми кластеризації можуть групувати подібні події або дії, допомагаючи виявити незвичну активність, яка може бути ознакою атаки.

Напівкероване навчання (Semi-Supervised Learning) комбінує малі набори мічених даних з великими наборами немічених, що дозволяє покращити точність моделей. Це особливо корисно в ситуаціях, коли мічені дані важко отримати або вони є дорогими.

Глибоке навчання, що базується на багатошарових нейронних мережах, здатне аналізувати великі обсяги даних і виявляти складні патерни. Глибокі нейронні мережі можуть застосовуватися для різних завдань у кібербезпеці, таких як виявлення вторгнень і аналіз поведінки користувачів.

Метод виявлення вторгнень (Intrusion Detection) використовує нейронні мережі, які можуть аналізувати мережевий трафік у реальному часі, виявляючи ознаки атак, таких як сканування портів або атаки типу DDoS. Завдяки здатності виявляти складні патерни в даних, ці моделі можуть ефективно визначати відхилення від норми.

Аналіз поведінки (Behavioral Analysis) допомагає нейронній мережі вивчати поведінкові патерни користувачів і систем, виявляючи аномалії, що

можуть свідчити про компрометацію облікового запису або внутрішні загрози. Наприклад, якщо користувач раптово починає виконувати незвичні дії, система може це зафіксувати та попередити адміністратора.

Обробка природної мови (NLP)

Обробка природної мови (NLP) дозволяє аналізувати текстові дані, що є критично важливим для виявлення фішингових атак та інших загроз, що використовують соціальну інженерію. Основні застосування NLP у кібербезпеці включають аналіз електронних листів та автоматизований аналіз логів.

Аналіз електронних листів: NLP допомагає розпізнавати ознаки фішингових повідомлень, аналізуючи структуру, стиль і зміст тексту. Наприклад, моделі NLP можуть навчитися виявляти типові ознаки фішингових атак, такі як незвичні формулювання або вимоги термінових дій.

Аналіз логів: Автоматизований аналіз лог-файлів за допомогою NLP допомагає виявляти підозрілі дії та аномалії в роботі систем. Це може включати виявлення незвичних запитів до баз даних або несанкціонованих змін у конфігурації системи.

У сфері кібербезпеки використовуються різні алгоритми ШІ, кожен з яких має свої переваги і специфічні застосування. Основні з них включають логістичну регресію, рішення дерева, метод опорних векторів, кластеризацію та нейронні мережі.

Штучний інтелект відіграє важливу роль у забезпеченні кібербезпеки, дозволяючи виявляти та нейтралізувати загрози більш ефективно і швидко. Використання методів машинного навчання, глибокого навчання та обробки природної мови дозволяє створювати інноваційні рішення, які підвищують рівень захисту даних і систем у сучасному цифровому світі. Розвиток технологій ШІ відкриває нові можливості для боротьби з кіберзагрозами, роблячи їх незамінним інструментом у сфері кібербезпеки. У майбутньому

можна очікувати подальше удосконалення цих технологій і розширення їх застосування для забезпечення більш надійного захисту від нових і все більш складних загроз.

1.3 Порівняння існуючих рішень

Сучасні методи захисту хмарних сховищ включають як традиційні підходи, так і інноваційні рішення, засновані на технологіях штучного інтелекту (ШІ). Розглянемо детально кожен із цих підходів, а також їхні переваги та недоліки.

Традиційні методи захисту даних у хмарних сховищах, такі як шифрування та багатофакторна аутентифікація (MFA), давно зарекомендували себе як надійні та ефективні засоби забезпечення безпеки. Шифрування даних забезпечує конфіденційність інформації, гарантуючи, що лише авторизовані користувачі можуть отримати доступ до зашифрованих даних. Використовуються як симетричні, так і асиметричні алгоритми шифрування, кожен з яких має свої особливості та переваги.

Багатофакторна аутентифікація підвищує рівень безпеки, вимагаючи від користувачів підтвердження своєї особи за допомогою декількох незалежних факторів, таких як пароль, одноразовий код, надісланий на мобільний телефон, або біометричні дані. Це значно знижує ризик несанкціонованого доступу, навіть якщо зловмисник отримає доступ до пароля користувача.

Однак, традиційні методи мають свої обмеження. Вони не завжди здатні виявити нові та складні загрози, такі як атаки нульового дня, оскільки орієнтовані на захист від відомих вразливостей. Крім того, налаштування та використання традиційних методів може бути складним і схильним до людських помилок, що також впливає на загальний рівень безпеки системи.

Рішення, що використовують технології ШІ, пропонують новий підхід до забезпечення кібербезпеки у хмарних сховищах. ШІ та машинне навчання (МН) дозволяють автоматизувати процеси виявлення загроз та реагування на них у реальному часі. Це досягається за рахунок аналізу великих обсягів даних і виявлення аномалій, які можуть вказувати на потенційні загрози.

Алгоритми машинного навчання можуть створювати профілі нормальної поведінки користувачів і виявляти відхилення від цих профілів. Наприклад, якщо користувач раптово починає діяти нетипово, система може це виявити і позначити як потенційну загрозу. Використання нейронних мереж для аналізу даних забезпечує високу точність виявлення атак і зниження кількості хибнопозитивних спрацювань.

Важливим аспектом є те, що системи на основі ШІ можуть самонавчатися і вдосконалювати свої моделі на основі нових даних. Це дозволяє їм адаптуватися до нових типів загроз і постійно підвищувати свій рівень ефективності. Проте, такі рішення мають свої недоліки. Вони складні у розробці та впровадженні, вимагають значних обчислювальних ресурсів і регулярного оновлення моделей.

Порівняння традиційних методів і рішень на основі ШІ показує, що обидва підходи мають свої сильні та слабкі сторони. Традиційні методи захисту, такі як шифрування і багатофакторна аутентифікація, забезпечують надійний захист від відомих загроз, але їхня здатність виявляти нові та складні атаки обмежена. Крім того, традиційні методи можуть бути уразливими до людських помилок і мають меншу гнучкість у реагуванні на нові виклики.

Рішення на основі ШІ, навпаки, демонструють високу ефективність у виявленні та реагуванні на сучасні кіберзагрози. Вони автоматизують процеси виявлення аномалій та аналізу поведінки користувачів, що

дозволяє оперативно реагувати на нові типи атак. Однак, їхня розробка та впровадження потребують значних ресурсів і постійної підтримки.

Найбільш ефективним підходом до забезпечення безпеки хмарних сховищ є поєднання традиційних методів захисту і нових технологій штучного інтелекту. Це дозволяє створити комплексну систему захисту, яка поєднує надійність і перевіреність традиційних методів з динамічністю і адаптивністю сучасних алгоритмів машинного навчання. Такий підхід забезпечує високий рівень безпеки та здатність протистояти сучасним викликам у сфері кібербезпеки.

Поєднання шифрування та багатофакторної аутентифікації з аналізом поведінки користувачів і виявленням аномалій на основі ШІ забезпечує всебічний захист даних у хмарних сховищах. Це дозволяє знизити ризик несанкціонованого доступу, захистити дані від перехоплення і забезпечити своєчасне виявлення та реагування на нові загрози. Впровадження таких комплексних рішень підвищує рівень довіри до хмарних технологій і сприяє їх більш широкому використанню у різних галузях.

1.4 Висновок до розділу 1 та постановка задачі

Отже, у даному розділі було проведено аналіз методів захисту хмарних сховищ від несанкціонованого доступу, розглянуто існуючі методи забезпечення безпеки, такі як шифрування, багатофакторна аутентифікація та управління доступом. Також проведено дослідження використання технологій штучного інтелекту для виявлення загроз і аномалій у поведінці користувачів, а також порівняння традиційних методів захисту та рішень на основі ШІ, враховуючи їхні переваги та недоліки. Таким чином, у процесі проведення загального огляду теоретичного матеріалу, були поставленні наступні задачі:

– розробити алгоритм роботи захищеної системи віддаленої роботи у хмарному сховищі;

– здійснити програмну реалізацію системи з використанням технологій штучного інтелекту;

– дослідити коректність та ефективність роботи розробленої системи.

З виконанням усіх поставлених задач, буде досягнуто головна мета даної дипломної роботи – буде здійснено розробку захищеної програми для віддаленої роботи користувачів у хмарному сховищі з використанням технологій штучного інтелекту для забезпечення високого рівня безпеки даних.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ЗАХИЩЕНОЇ ВІДДАЛЕНОЇ РОБОТИ

2.1 Аналіз можливих вдосконалень систем захисту

У сучасному світі, де обсяги даних, що передаються через мережу Інтернет, постійно зростають, питання забезпечення захисту інформації набуває особливої актуальності. Традиційні методи захисту даних, такі як паролі та шифрування, вже не завжди можуть забезпечити необхідний рівень безпеки. Тому виникає потреба у впровадженні нових технологій, зокрема штучного інтелекту, для покращення систем захисту даних.

Першим напрямом вдосконалення є впровадження багатофакторної аутентифікації (БФА). Традиційні методи аутентифікації, засновані лише на паролях, мають низку недоліків, таких як легкість підбору або викрадення паролів. БФА включає комбінацію декількох незалежних факторів аутентифікації, що може значно підвищити рівень безпеки. Використання біометричних даних, таких як відбитки пальців або розпізнавання обличчя, стає все більш популярним завдяки зручності та високому рівню захисту.

Другим важливим аспектом є впровадження методів штучного інтелекту (ШІ) та машинного навчання (МН) для виявлення та запобігання кіберзагрозам. Завдяки здатності аналізувати великі обсяги даних у реальному часі, ШІ може виявляти аномалії у поведінці користувачів, що можуть вказувати на потенційні загрози. Аналіз поведінки користувачів дозволяє виявити патерни, що відрізняються від звичайної поведінки, та автоматично реагувати на виявлені загрози, блокуючи підозрілі сесії або сповіщаючи адміністраторів системи.

Третім напрямом вдосконалення є покращення управління доступом за допомогою систем управління правами доступу (DRM). Це дозволяє контролювати, хто і які дії може виконувати з даними, що зберігаються у

хмарному сховищі. Гранульований контроль доступу забезпечує можливість задавати різні рівні доступу для різних користувачів або груп, що сприяє підвищенню безпеки даних. Моніторинг активності дозволяє відстежувати всі дії, виконувані користувачами, що дозволяє виявляти та реагувати на підозрілі дії.

Іншим важливим аспектом є інтеграція нових технологій з існуючими системами безпеки. Це забезпечує сумісність з уже наявними рішеннями для управління доступом, моніторингу та аналізу загроз. Централізоване управління дозволяє об'єднати різні системи у єдину платформу, що забезпечує централізоване управління безпекою. Використання автоматизованих засобів для реагування на загрози знижує навантаження на IT-персонал та підвищує оперативність реагування.

Шифрування даних є ще одним з найважливіших методів захисту інформації. Використання сучасних алгоритмів шифрування, таких як AES-256, забезпечує захист даних як у стані спокою, так і під час передачі. Керування криптографічними ключами забезпечує безпечне зберігання та управління ключами, що є критично важливим для підтримання цілісності та конфіденційності даних. Енд-то-енд шифрування гарантує, що дані залишаються зашифрованими на всіх етапах передачі, від відправника до отримувача.

Постійний аудит систем безпеки та відповідність вимогам регуляторних органів є також критично важливими для забезпечення надійного захисту даних. Регулярні перевірки безпеки дозволяють виявляти вразливості та усувати їх, забезпечуючи постійне підвищення рівня захисту. Відповідність стандартам, таким як GDPR, HIPAA, PCI DSS, гарантує захист конфіденційної інформації та відповідність вимогам законодавства. Документування процесів та ведення записів щодо заходів безпеки забезпечує прозорість та контроль, що є необхідним для забезпечення надійного управління безпекою.

Аналіз можливих вдосконалень систем захисту даних показує, що використання сучасних технологій, таких як багатофакторна аутентифікація, штучний інтелект, системи управління правами доступу, криптографічні методи та регулярний аудит, дозволяє значно підвищити рівень безпеки. Інтеграція цих технологій у комплексну систему захисту даних забезпечує динамічний та адаптивний підхід до кібербезпеки, що є необхідним у сучасному цифровому середовищі.

2.2 Вдосконалення системи методом використання штучного інтелекту

Використання штучного інтелекту (ШІ) для вдосконалення систем захисту даних у хмарних сховищах є одним із найбільш перспективних напрямів сучасної кібербезпеки. ШІ дозволяє не лише автоматизувати процеси виявлення та реагування на загрози, але й підвищити точність цих процесів, знижуючи кількість хибних спрацьовувань.

По-перше, використання алгоритмів машинного навчання (МН) дозволяє створювати системи, які здатні аналізувати великі обсяги даних та виявляти аномалії у поведінці користувачів. Наприклад, алгоритми МН можуть вивчати поведінку користувачів протягом певного періоду часу, формуючи профілі нормальної активності. На основі цих профілів система може виявляти відхилення від нормальної поведінки, що можуть свідчити про потенційні загрози, такі як спроби несанкціонованого доступу або внутрішні загрози.

По-друге, впровадження систем оповіщення та реагування на інциденти на основі ШІ дозволяє значно скоротити час реагування на загрози. Традиційні системи безпеки часто покладаються на ручне втручання ІТ-фахівців для реагування на інциденти, що може призводити до затримок у реагуванні та підвищення ризику втрати даних. ШІ може

автоматично аналізувати інциденти безпеки та пропонувати або здійснювати необхідні заходи реагування, що знижує навантаження на IT-персонал та підвищує оперативність реагування.

По-третє, використання методів глибокого навчання (Deep Learning) у сфері кібербезпеки дозволяє створювати більш точні та ефективні моделі для виявлення загроз. Глибокі нейронні мережі можуть аналізувати складні патерни у даних, що не можуть бути виявлені традиційними методами. Наприклад, вони можуть ідентифікувати нові види атак, які раніше не зустрічалися, або розпізнавати складні мультиетапні атаки, які включають кілька різних методів компрометації системи.

Крім того, ШІ може бути використаний для покращення систем багатофакторної аутентифікації (БФА). Алгоритми МН можуть аналізувати поведінкові біометричні дані, такі як динаміка набору тексту або модель руху миші, для додаткової аутентифікації користувачів. Це дозволяє створювати більш гнучкі та безпечні системи аутентифікації, що можуть автоматично адаптуватися до змін у поведінці користувачів.

Важливим аспектом використання ШІ є також інтеграція з існуючими системами управління правами доступу (DRM). Це дозволяє автоматизувати процеси управління доступом та забезпечити більш точний контроль над тим, хто і які дії може виконувати з даними. Наприклад, ШІ може автоматично змінювати права доступу на основі аналізу поведінки користувачів або виявлених загроз, забезпечуючи динамічне управління доступом.

Окремо варто згадати про роль ШІ у забезпеченні захисту даних у стані спокою та під час передачі. Використання алгоритмів шифрування на основі ШІ дозволяє створювати більш стійкі до атак криптографічні схеми. Наприклад, ШІ може використовуватися для автоматичного генерації ключів шифрування або для виявлення слабких місць у існуючих криптографічних алгоритмах.

Розглядаючи впровадження ШІ у систему захисту даних, необхідно також враховувати питання конфіденційності та відповідності вимогам регуляторних органів. ШІ може використовуватися для автоматичного моніторингу та аудиту відповідності стандартам, таким як GDPR, HIPAA або PCI DSS. Це дозволяє забезпечити постійний контроль над безпекою даних та відповідність вимогам законодавства.

Використання штучного інтелекту для вдосконалення систем захисту даних у хмарних сховищах має значний потенціал. Завдяки можливостям аналізу великих обсягів даних, виявлення аномалій, автоматизації реагування на інциденти та покращення систем аутентифікації, ШІ дозволяє створювати більш ефективні та динамічні механізми захисту. Інтеграція ШІ з існуючими системами безпеки та управління правами доступу забезпечує комплексний підхід до кібербезпеки, що є необхідним у сучасному цифровому середовищі.

2.3 Розробка алгоритму програми для захищеної віддаленої роботи

Процес розробки алгоритму програми для забезпечення захищеної віддаленої роботи користувача у хмарному сховищі є багатоступеневим і вимагає врахування різних аспектів кібербезпеки. Алгоритм повинен забезпечувати аутентифікацію, авторизацію, моніторинг активності, шифрування даних та реагування на інциденти. Відповідно до проаналізованої інформації в попередніх підрозділах, буде розроблено структурний алгоритм роботи програми.

Для початку створимо схематичний алгоритм авторизації та поетапний шлях перевірки введеного логіну та паролю (рис. 2.1).

Розглянемо покроково структуру програми:

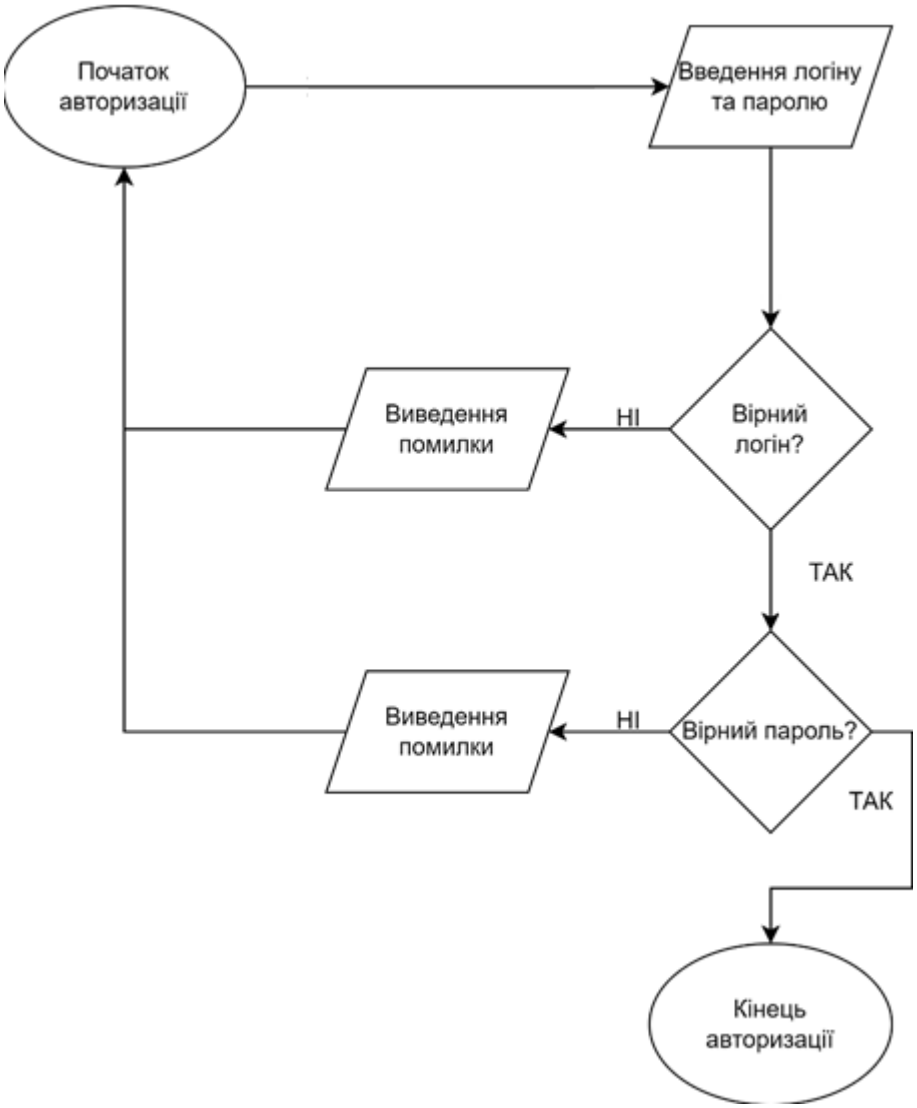


Рисунок 2.1 – Схема авторизації користувача

Ця схема авторизації надає можливість користувачам безпечно входити в систему за допомогою логіна та пароля. Вона дозволяє перевірити правильність логіна та пароля перед наданням доступу до системи. У випадку неправильного логіна або пароля, система повідомляє про помилку та дає користувачеві можливість спробувати знову.

Розглянемо детальніше дану структуру по кроках:

Крок 1. Початок авторизації:

Користувач ініціює процес авторизації, запускаючи програму.

Крок 2. Введення логіну та паролю:

Користувач вводить свій логін та пароль у відповідні поля.

Крок 3. Вірний логін?:

Система перевіряє, чи введений користувачем логін є дійсним існуючим логіном, зареєстрованим в системі.

Крок 3.1. НІ:

Якщо логін невірний, система виводить повідомлення про помилку, повідомляючи користувача, що введений логін не знайдений у системі.

Потім система переходить до кроку «Початок авторизації», де користувач може спробувати ввести логін знову та продовжити авторизацію.

Крок 3.2. ТАК:

Якщо логін вірний, система переходить до наступного кроку.

Крок 4. Вірний пароль?:

Система перевіряє, чи введений користувачем пароль відповідає логіну, який був введений раніше.

Крок 4.1. НІ:

Якщо пароль невірний, система виводить повідомлення про помилку, повідомляючи користувача, що введений пароль не співпадає з паролем, пов'язаним з введеним логіном.

Потім система переходить до кроку «Початок авторизації», де користувач може спробувати ввести логін та пароль знову та продовжити авторизацію.

Крок 4.2. ТАК:

Якщо пароль вірний, система переходить до наступного кроку.

Крок 5. Кінець авторизації:

Система успішно авторизувала користувача. Користувач має доступ до основного функціоналу програми, що вимагає авторизації.

Наступна схема демонструє процес виявлення аномалей за допомогою моделі машинного навчання при роботі з файлами в хмарному сховищі. В залежності від дій користувача, модель може визначити дій

користувача як нормальні чи аномальні. У випадку виявлення аномалій буде виведене повідомлення про аномальні дій зі сторони користувача (рис. 2.2).

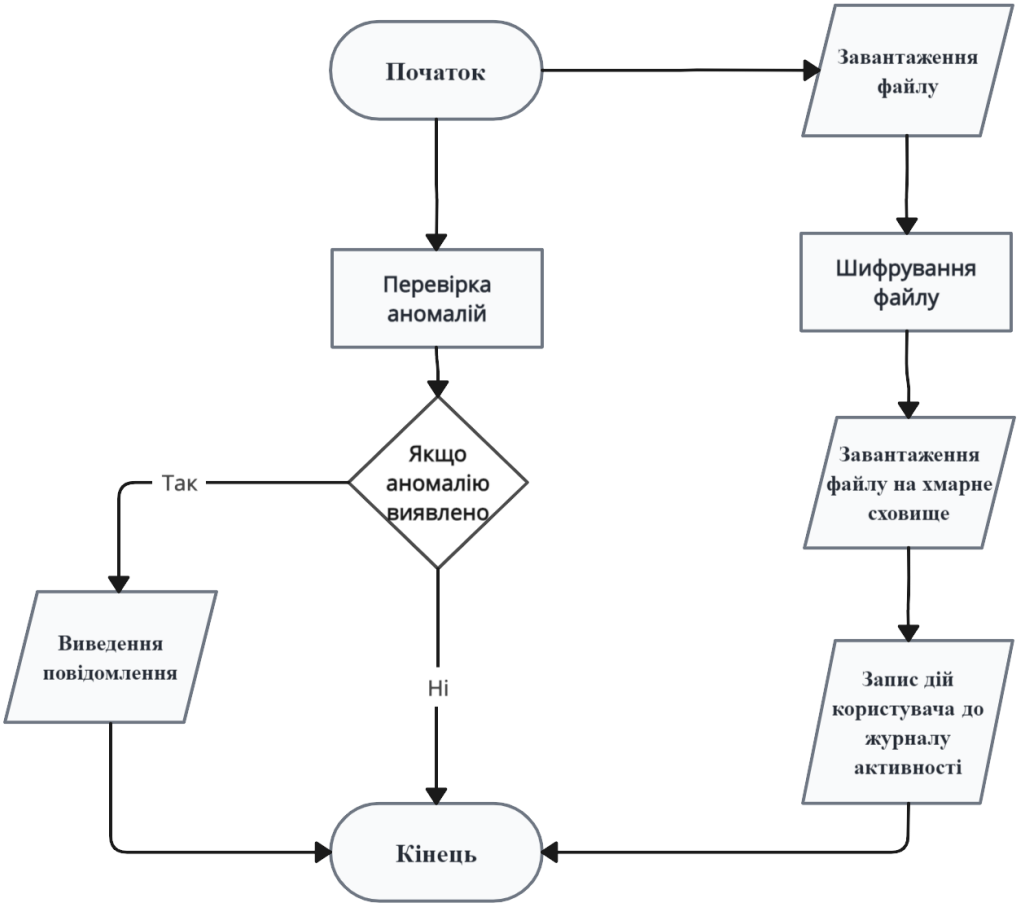


Рисунок 2.2 – Схема виявлення аномалій при роботі з хмарним сховищем

Здійснимо покроковий опис даної структури:

Крок 1. Початок завантаження файлів в систему:

Користувач ініціює процес завантаження файлів.

Крок 2. Вибір файлу для завантаження

Користувач натискає кнопку "Виберіть файл для завантаження".

Відкривається діалогове вікно вибору файлу.

Користувач вибирає файл для завантаження.

Крок 3. Шифрування файлу

Відбувається шифрування файлу перед завантаження його до хмарного сховища.

Крок 4. Завантаження файлу на хмарне сховище

Після шифрування файлу він завантажується до хмарного сховища.

Дія користувача записується в журнал активності.

Крок 5. Перевірка аномалій

Далі використовується модель машинного навчання, яка перевіряє зібрані дані про активність користувачів з журналу активності такі як незвичний час доступу, обсяг завантажених даних або IP-адреси.

Крок 5.1. ТАК:

Якщо модуль виявляє аномалії, відображається відповідне повідомлення на інтерфейсі користувача.

Крок 5.2. Ні:

Якщо аномалії не виявлено, буде продовжуватися відображатися повідомлення "Аномалій не виявлено".

2.4 Висновок до розділу 2

У даному розділі було проведено аналіз можливих вдосконалень систем захисту даних у хмарному середовищі з метою підвищення рівня безпеки та ефективності захищеної віддаленої роботи користувачів. В рамках аналізу розглянуто ключові аспекти впровадження штучного інтелекту для виявлення та запобігання кіберзагрозам, а також важливість багатофакторної аутентифікації та інтеграції з системами управління правами доступу. Встановлено, що методи машинного навчання мають великий потенціал для покращення виявлення загроз, дозволяючи ідентифікувати аномалії у поведінці користувачів та автоматично реагувати на них.

Додатково було розроблено алгоритм програми для забезпечення захищеної віддаленої роботи користувачів, який включає етапи

аутентифікації, авторизації, моніторингу активності, шифрування даних та реагування на інциденти. Запропонований алгоритм забезпечує високий рівень захисту даних, дозволяючи мінімізувати ризики несанкціонованого доступу та забезпечити цілісність і конфіденційність інформації у хмарному сховищі.

Загалом, проведений аналіз та розробка алгоритму сприяють підвищенню рівня безпеки та надійності системи захищеної віддаленої роботи, що має велике значення у сучасному цифровому середовищі, де захист інформації є критично важливим.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ТА ЙОГО ТЕСТУВАННЯ

У даному розділі буде здійснено вибір середовища розробки програмного додатку та вибір мови програмування для його реалізації. Також буде здійснено розробку програмного додатку та тестування правильності та надійності його роботи, після чого буде здійснено аналіз практичного застосування розробленого додатку.

3.1 Обґрунтування вибору середовища розробки та мови програмування

При виборі середовища розробки та мови програмування для проекту "Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту" було враховано кілька ключових факторів: функціональні вимоги проекту, зручність розробки, підтримка необхідних бібліотек та фреймворків, а також безпека і продуктивність.

Python був обраний як основна мова програмування з кількох причин. По-перше, Python має багатий набір бібліотек, які підтримують різні аспекти розробки, включаючи обробку даних, машинне навчання та роботу з хмарними сервісами. Наприклад, бібліотеки pandas, numpy, scikit-learn, tensorflow та keras використовуються для машинного навчання, тоді як google-api-python-client і oauth2client дозволяють інтегруватися з API Google Drive.

По-друге, Python відомий своєю простотою і читабельністю, що сприяє швидкому розвитку програмного забезпечення і полегшує підтримку коду. Це особливо важливо для командної роботи, де код повинен бути зрозумілим для всіх учасників команди. Велика спільнота розробників

Python забезпечує швидку підтримку і доступ до численних ресурсів, включаючи документацію, форуми і приклади коду, що допомагає швидко знаходити рішення для будь-яких проблем, які можуть виникнути під час розробки. Крім того, Python працює на різних операційних системах (Windows, macOS, Linux), що робить його універсальним вибором для різних середовищ розробки та використання.

Для розробки серверної частини та інтеграції з Google Drive API було обрано інтегроване середовище розробки (IDE) PyCharm. PyCharm надає численні інструменти для розробки, включаючи відладчик, рефакторинг коду, підтримку тестування, інтеграцію з системами контролю версій та багато іншого. Це значно підвищує продуктивність розробників і покращує якість коду. PyCharm спеціально розроблений для Python і має відмінну підтримку цієї мови, включаючи автозавершення коду, перевірку синтаксису та інші функції, що полегшують написання коду. Крім того, PyCharm підтримує популярні веб-фреймворки, такі як Flask і Django, які можуть бути використані для створення серверної частини додатку.

Для створення серверної частини додатку був обраний Flask. Flask є легким мікрофреймворком для Python, який надає велику гнучкість при розробці веб-додатків. Він дозволяє легко створювати прості та масштабовані серверні додатки. Flask слідує мінімалістичному підходу і надає тільки основні функції, необхідні для створення веб-додатків. Це дозволяє розробникам додавати лише ті компоненти, які дійсно потрібні, що зменшує складність і залежності проекту. Flask має відмінну документацію і велику спільноту, що полегшує навчання та вирішення проблем, які можуть виникнути під час розробки.

Вибір Python як основної мови програмування був зумовлений її простотою, багатством бібліотек та активною спільнотою. PyCharm був обраний як основне середовище розробки для його потужних інструментів розробки та підтримки Python. Flask був обраний для створення серверної

частини додатку через його легкість і гнучкість. Ці вибори забезпечують ефективну розробку, тестування та розгортання програми для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту.

Для розробки графічного інтерфейсу користувача було обрано бібліотеку PyQt5. PyQt5 є однією з найбільш популярних бібліотек для створення GUI в Python завдяки своїй потужності, гнучкості та широким можливостям налаштування. Вона дозволяє створювати сучасні та інтерактивні інтерфейси, які легко інтегруються з іншими частинами проекту. PyQt5 підтримує роботу з різними елементами управління, такими як кнопки, поля введення, випадаючі списки, а також надає можливість відображення веб-сторінок через компонент QWebEngineView.

Вибір PyQt5 також зумовлений наявністю відмінної документації та великої спільноти, що забезпечує швидкий доступ до прикладів, підказок та рішень для різних завдань. Це значно полегшує процес навчання та розробки, зменшуючи час на пошук інформації та вирішення проблем.

Крім того, PyQt5 дозволяє створювати кросплатформенні додатки, що можуть працювати на різних операційних системах без значних змін у коді. Це забезпечує універсальність та гнучкість розробки, що є важливим фактором для сучасних програмних проектів.

Ще одним важливим аспектом вибору PyQt5 є можливість легкої інтеграції з іншими бібліотеками та інструментами, що використовуються у проекті. Наприклад, інтеграція з бібліотеками для роботи з Google API, а також з бібліотеками для обробки та аналізу даних. Це дозволяє створювати єдине рішення, що включає всі необхідні компоненти для виконання завдань проекту.

Для організації та управління проектом було обрано систему контролю версій Git, а також платформу GitHub для зберігання коду і співпраці. Git забезпечує надійне відстеження змін у коді, можливість

працювати над різними гілками проекту та легкий відкат до попередніх версій. GitHub надає зручний веб-інтерфейс для управління репозиторіями, а також інструменти для спільної роботи, такі як pull requests, issues та wiki. Використання Git і GitHub дозволяє ефективно організувати процес розробки, забезпечуючи прозорість та контроль якості коду.

Для тестування додатків було обрано бібліотеку pytest. pytest є потужним і гнучким інструментом для написання та виконання тестів у Python. Вона підтримує модульні, функціональні та інтеграційні тести, що дозволяє забезпечити високу якість коду та виявити можливі помилки на ранніх етапах розробки. pytest має багату екосистему плагінів, що дозволяють розширювати функціональність та інтегруватися з іншими інструментами тестування та безперервної інтеграції.

Для безперервної інтеграції (CI) було обрано платформу GitHub Actions. GitHub Actions дозволяє автоматизувати процеси збірки, тестування та розгортання додатків безпосередньо з репозиторію на GitHub. Це забезпечує швидкий зворотний зв'язок для розробників, автоматизуючи рутинні завдання та підвищуючи продуктивність команди. Інтеграція GitHub Actions з GitHub забезпечує легкість налаштування та управління процесами CI/CD, а також високу надійність та гнучкість у виборі конфігурацій.

Таким чином, вибір середовища розробки та мови програмування базувався на критеріях простоти, гнучкості, продуктивності та підтримки сучасних інструментів і бібліотек. Python як основна мова програмування, PyCharm як середовище розробки, Flask для серверної частини, PyQt5 для графічного інтерфейсу, а також Git, GitHub, pytest і GitHub Actions для управління проектом, тестування та безперервної інтеграції — всі ці вибори сприяють ефективній розробці, тестуванню та розгортанню високоякісного програмного забезпечення, що відповідає сучасним стандартам та вимогам користувачів.

Для зберігання та обробки даних у проекті було обрано базу даних SQLite. SQLite є вбудованою реляційною базою даних, яка зберігається у вигляді одного файлу на диску. Вона не потребує налаштування серверу бази даних, що робить її ідеальною для невеликих до середніх за розміром додатків. Використання SQLite дозволяє зберігати дані локально під час розробки та тестування, забезпечуючи швидкий доступ до них. Крім того, Python має вбудовану підтримку SQLite через модуль sqlite3, що забезпечує легкість інтеграції.

Для взаємодії з базою даних було обрано ORM (Object-Relational Mapping) бібліотеку SQLAlchemy. SQLAlchemy надає потужний інструментарій для роботи з базами даних, дозволяючи розробникам працювати з даними у вигляді об'єктів Python. Це значно спрощує код, пов'язаний із взаємодією з базою даних, і забезпечує захист від SQL-ін'єкцій. Використання SQLAlchemy дозволяє легко змінювати базу даних у майбутньому, якщо виникне така необхідність, оскільки вона підтримує роботу з багатьма різними СУБД (системами управління базами даних).

Для розгортання додатку було обрано платформу Heroku. Heroku є хмарною платформою, яка дозволяє легко розгортати, управляти та масштабувати додатки. Використання Heroku забезпечує швидке розгортання додатку без необхідності налаштовувати інфраструктуру. Heroku підтримує автоматичне розгортання з репозиторіїв GitHub, що дозволяє інтегрувати процеси безперервної інтеграції та доставки (CI/CD). Крім того, Heroku надає можливість використовувати різні додатки для моніторингу, логування та управління базами даних, що робить її ідеальним вибором для розгортання веб-додатків.

Для реалізації функцій штучного інтелекту було обрано бібліотеки TensorFlow та scikit-learn. TensorFlow є однією з найпопулярніших бібліотек для розробки і тренування моделей машинного навчання. Вона забезпечує потужний інструментарій для побудови нейронних мереж та інших моделей

глибокого навчання. TensorFlow підтримує роботу з GPU, що значно прискорює тренування моделей. Використання TensorFlow дозволяє розробляти складні моделі машинного навчання, які можуть автоматично виявляти та реагувати на загрози у хмарному середовищі.

scikit-learn була обрана для реалізації класичних алгоритмів машинного навчання, таких як класифікація, регресія та кластеризація. scikit-learn є дуже зручною для швидкої побудови та тестування моделей, оскільки вона надає простий і зрозумілий API. Використання scikit-learn дозволяє легко експериментувати з різними алгоритмами та вибирати найкращий для конкретних завдань. Крім того, scikit-learn добре інтегрується з іншими бібліотеками Python, такими як pandas і numpy, що забезпечує зручність обробки і аналізу даних.

Для візуалізації даних було обрано бібліотеку Matplotlib. Matplotlib є потужним інструментом для створення різноманітних графіків і візуалізацій. Вона дозволяє створювати лінійні графіки, гістограми, кругові діаграми та інші типи візуалізацій, що допомагає аналізувати результати моделювання та виявляти тенденції у даних. Використання Matplotlib забезпечує можливість створення інформативних і наочних графіків, що є важливим для представлення результатів роботи додатку.

Загалом, вибір Python як основної мови програмування, PyCharm як середовища розробки, Flask для серверної частини, PyQt5 для графічного інтерфейсу, SQLAlchemy для взаємодії з базою даних, Heroku для розгортання, TensorFlow та scikit-learn для реалізації штучного інтелекту, а також Matplotlib для візуалізації даних забезпечує ефективну розробку, тестування, розгортання та підтримку проекту. Ці інструменти і технології дозволяють створити потужний, гнучкий і надійний додаток для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту, що відповідає сучасним вимогам та стандартам.

3.2 Програмна реалізація

Програмна реалізація є невід'ємною складовою процесу розробки програмного забезпечення. Вона включає в себе розробку алгоритмів, структур даних, функцій та модулів, які дозволяють програмі виконувати необхідні завдання.

У контексті нашого завдання, для створення програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту, ми використовуватимемо мову програмування Python та бібліотеку PyQt5 для розробки графічного інтерфейсу користувача.

Спочатку створимо файл , який міститиме всі необхідні бібліотеки для нашої програми.

Це дозволить легко встановити всі залежності за допомогою однієї команди.

Далі реалізуємо функції підключення до бази даних, що допоможе виконувати SQL-запити до неї, зберігати або витягати дані користувачів для аутентифікаії та реєстрації, а також ведення журналу активності самих користувачів.

```
def connect_db():
    return sqlite3.connect('users.db')
def create_users_table():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute(users (id, username, password))
    conn.commit()
    conn.close()
```

Наступним кроком створюємо функцію, яка реалізує можливість аутентифікації користувачів, забезпечуючи можливість входу для користувачів у систему.

```
def login_user(username, password):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM users WHERE username = ?', (username,))
```

```
    user = cursor.fetchone()
    conn.close()
    if user and check_password_hash(user[2], password):
        return True
    return False
```

Після цього реалізовуємо функції для шифрування та розшифрування файлів.

Генеруємо новий ключ для шифрування даних.

```
from cryptography.fernet import Fernet
from config import Config
def generate_key():
    return Fernet.generate_key()
```

Реалізуємо шифрування файлу за допомогою вказаного ключа.

```
def encrypt_file(file_path, key=Config.ENCRYPTION_KEY):
    with open(file_path, 'rb') as file:
        data = file.read()
    fernet = Fernet(key)
    encrypted = fernet.encrypt(data)
    with open(file_path, 'wb') as file:
        file.write(encrypted)
```

Створюємо функцію для дешифрування файлу за допомогою того самого ключа.

```
def decrypt_file(file_path, key=Config.ENCRYPTION_KEY):
    with open(file_path, 'rb') as file:
        data = file.read()
    fernet = Fernet(key)
    decrypted = fernet.decrypt(data)
    with open(file_path, 'wb') as file:
        file.write(decrypted)
```

Наступним кроком розробимо функцію для моніторингу активності користувачів.

Налаштовуємо систему логування для запису активності користувачів та реалізуємо функція запису повідомлення про моніторинг активності користувача в лог-файл

```
import logging
import config
logging.basicConfig(filename=config.LOG_FILE, level=logging.INFO)
def monitor_activity():
   logging.info("Моніторинг активності користувача")
```

Далі створюємо функції для завантаження та вивантаження файлів з Google Drive, використовуючи Google Drive API.

Завантажуємо облікові дані з файлу сервісного акаунту та створюєом об'єкт для взаємодії з Google Drive API, використовуючи отримані облікові дані.

```
from google.oauth2 import service_account
from googleapiclient.discovery import build
SCOPES = ['https://www.googleapis.com/auth/drive.file']
SERVICE_ACCOUNT_FILE = 'service_account.json'
credentials = service_account.Credentials.from_service_account_file(
      SERVICE_ACCOUNT_FILE, scopes=SCOPES)
service = build('drive', 'v3', credentials=credentials)
```

Встановлюється метадані файлу для завантаження на Google Drive за допомогою методу, який повертає ідентифікатор завантаженого файлу.

```
def upload_file_to_gdrive(file_path, gdrive_file_name):
   file_metadata = {'name': gdrive_file_name}
   media = MediaFileUpload(file_path, resumable=True)
   file = service.files().create(body=file_metadata, media_body=media, fields='id').execute()
   return file.get('id')
```

Створюється запит на отримання медіа-даних файлу за вказаним ідентифікатором файлу та його заваетаження зі сховища

```
def download_file_from_gdrive(gdrive_file_id, download_path):
   request = service.files().get_media(fileId=gdrive_file_id)
   fh = io.FileIO(download_path, 'wb')
   downloader = MediaIoBaseDownload(fh, request)
   done = False
   while done is False:
      status, done = downloader.next_chunk()
   return done
```

Потім реалізуємо функції виявлення аномалій у даних за допомогою попередньо натренованої моделі машинного навчання, яка повертає масив даних, що містять аномалії.

```
import numpy as np
import pickle
def detect_anomalies(data):
    with open('model.pkl', 'rb') as model_file:
        model = pickle.load(model_file)
    predictions = model.predict(data)
    anomalies = data[predictions == -1]
    return anomalies
```

Також розробимо головну програму, що містить всі основні функції, графічний інтерфейс користувача та обробку подій:

Реалізація графічного інтерфейсу вікна для авторизації

```
app = Flask(__name__)
app.config.from_object(Config)
class AuthWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Авторизація")
        self.setGeometry(100, 100, 400, 300)
        self.setWindowIcon(QIcon('icon.png'))
        self.layout = QVBoxLayout()
        self.login_label = QLabel("Логін")
        self.layout.addWidget(self.login_label)
        self.login_input = QLineEdit()
        self.layout.addWidget(self.login_input)
        self.password_label = QLabel("Пароль")
        self.layout.addWidget(self.password_label)
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)
        self.layout.addWidget(self.password_input)
        self.login_button = QPushButton("Увійти")
        self.login_button.clicked.connect(self.handle_login)
        self.layout.addWidget(self.login_button)
```

Створення функції для обробки входу користувача в вікні авторизації

```
    def handle_login(self):
  username = self.login_input.text()
  password = self.password_input.text()
  if login_user(username, password):
    self.open_main_window()
  else:
    self.login_label.setText("Невірний логін або пароль")
```

Реалізація функції для відкриття головного вікна після успішного входу

```
  def open_main_window(self):
  self.main_window = MainWindow(username=self.login_input.text())
  self.main_window.show()
  self.close()
```

Далі описуємо головне вікно додатку для перевірки аномалій, завантаження та скачування файлів.

Виконуємо створення графічного інтерфейсу головного вікна з можливістю вибором файлів для завантажування та скачування, а також міткою для відображення статусу аномальних дій та виходом з програми:

```
class MainWindow(QMainWindow):
  def __init__(self, username):
    super().__init__()
    self.setWindowTitle("Хмарний захист")
    self.setGeometry(100, 100, 400, 300)
    self.setWindowIcon(QIcon('icon.png'))
    self.username = username
    self.layout = QVBoxLayout()
    self.upload_button = QPushButton("Виберіть файл для завантаження")
    self.upload_button.clicked.connect(self.upload_file)
    self.layout.addWidget(self.upload_button)
    self.download_button = QPushButton("Виберіть файл для скачування")
    self.download_button.clicked.connect(self.download_file)
    self.layout.addWidget(self.download_button)
    self.anomaly_label = QLabel("Аномалії: Аномалій не виявлено")
    self.layout.addWidget(self.anomaly_label)
    self.exit_button = QPushButton("Вийти")
    self.exit_button.clicked.connect(self.close)
```

```
        self.layout.addWidget(self.exit_button)
        container = QWidget()
        container.setLayout(self.layout)
        self.setCentralWidget(container)
```

Опис функції для перевірки аномалії та оновення статусу аномалій в інтерфейсі.

```
    def check_anomalies(self):
        data = self.collect_user_activity_data()
        anomalies = detect_anomalies(data)
        if anomalies:
            self.anomaly_label.setText("Аномалії виявлено!")
        else:
            self.anomaly_label.setText("Аномалій не виявлено")
```

Функція збору даних про активність користувача з логів, а також читання лог-файлів і структурузація даних у вигляді списку.

```
    def collect_user_activity_data(self):
        with open(Config.MONITOR_LOG_FILE, 'r') as log_file:
            logs = log_file.readlines()
        data = []
        for log in logs:
            log_data = log.strip().split(',')
            data.append({
                'timestamp': log_data[0],
                'username': log_data[1],
                'action': log_data[2],
                'file_path': log_data[3],
                'local_ip': log_data[4],
                'public_ip': log_data[5]
            })
        return data
```

Наступна функція логує активність користувача. Записує дані про дії користувача у лог-файл, включаючи час, ім'я користувача, дію, шлях до файлу та IP-адреси

```
    def log_user_activity(self, action, file_path):
        local_ip, public_ip = self.get_user_ip()
        with open(Config.MONITOR_LOG_FILE, 'a') as log_file:
```

```
            log_file.write(f"{datetime.now()},{self.username},{action},{file_path},{local_ip},{public_ip}\n")
```

Дана функція виконує реалізацію отримання локальної та публічної IP-адреси користувача.

```
def get_user_ip(self):
            hostname = socket.gethostname()
        local_ip = socket.gethostbyname(hostname)
            try:
            public_ip = requests.get('https://api.ipify.org').text
        except requests.RequestException as e:
            public_ip = "Не вдалося отримати публічну IP-адресу"
        return local_ip, public_ip
if __name__ == '__main__':
    app = QApplication(sys.argv)
    auth_window = AuthWindow()
    auth_window.show()
    sys.exit(app.exec_())
```

Програмна реалізація має надзвичайно важливе значення в сучасному світі, де програми та додатки стають все складнішими та багатофункціональними. Вона дозволяє втілювати ідеї та концепції в життя, створюючи програмне забезпечення, яке надає користувачам необхідні функції та можливість ефективної взаємодії з інформацією.

3.3 Тестування розробленого додатку

Тестування програмного додатку є важливою частиною процесу розробки програмного забезпечення. У сучасному цифровому світі, де комп'ютерні програми використовуються в різних сферах життя, від бізнесу до розваг, зростає значення якісного та надійного програмного забезпечення.

Насамперед, під час тестування слід перевірити правильність роботи системи авторизації користувачів (рис. 3.1).

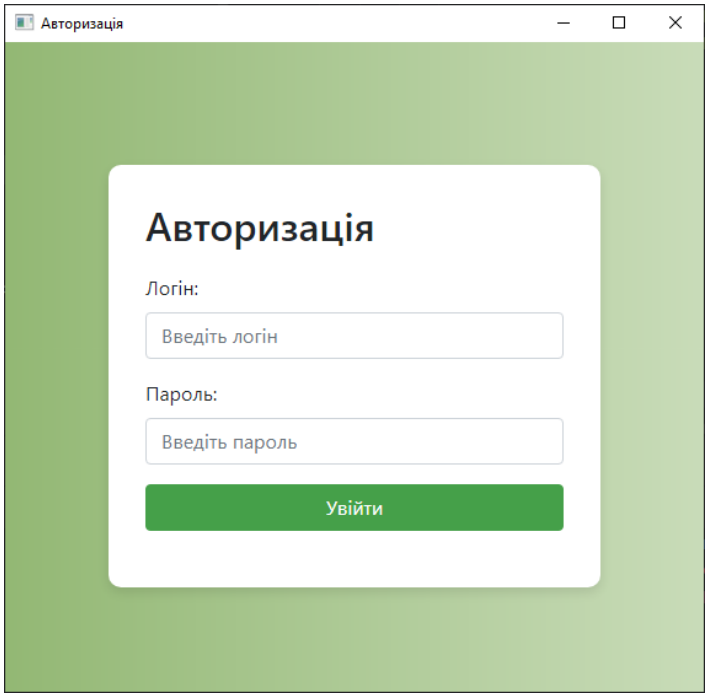


Рисунок 3.1 – Вікно авторизації

Для початку виконаємо перевірку чи зможе користувач авторизуватися при введені хибного логіну чи паролю (рис. 3.2).

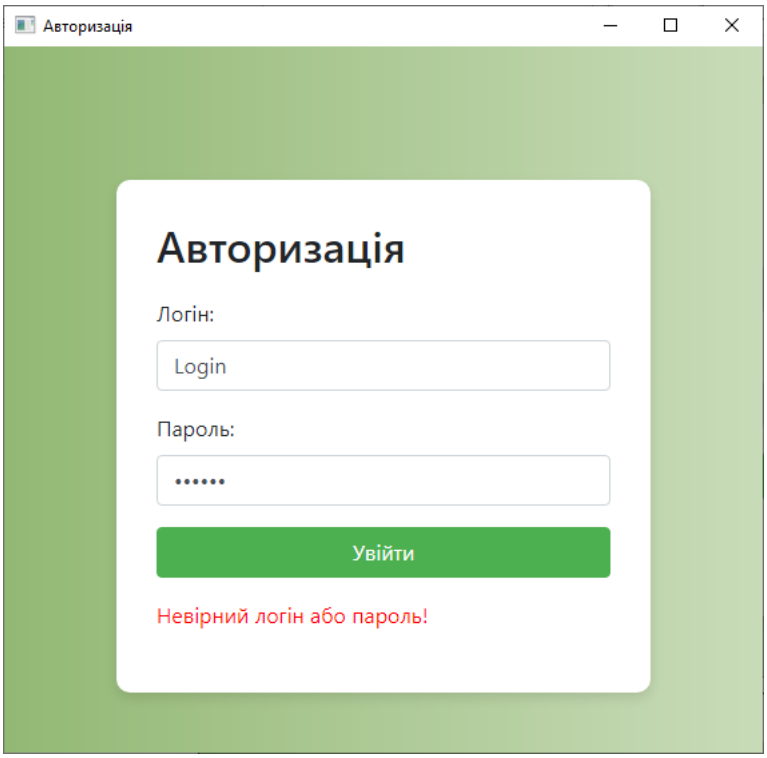


Рисунок 3.2 – Виведення помилки про невірний логін чи пароль

Як бачимо з рисунку 3.2, система повідомляє користувача про помилку при введенні неправильних даних.

Наступним кроком перевіримо, що відбудеться, якщо логін та пароль будуть коректними (рис. 3.3).

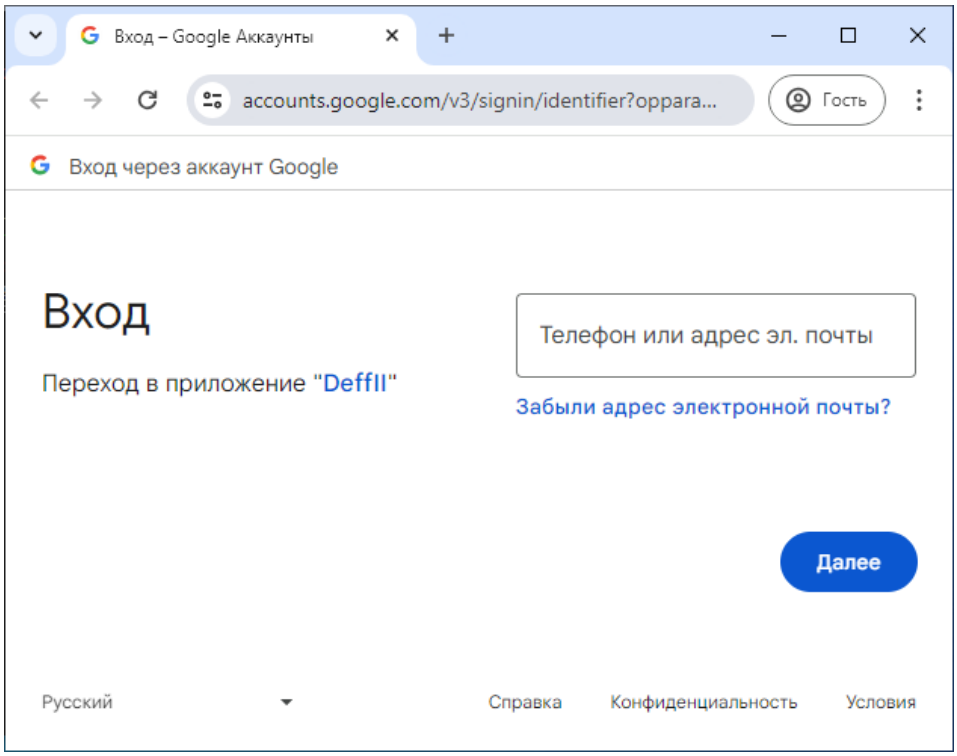


Рисунок 3.3 – Основний сайт хмарного сервісу Google для авторизації та інтеграції програми

Як видно на рисунку 3.3 при вводі коректних даних користувача відправляє до основної сторінки хмарного сервісу Google для авторизації в свій обліковий запис та підтвердження інтегрування програми.

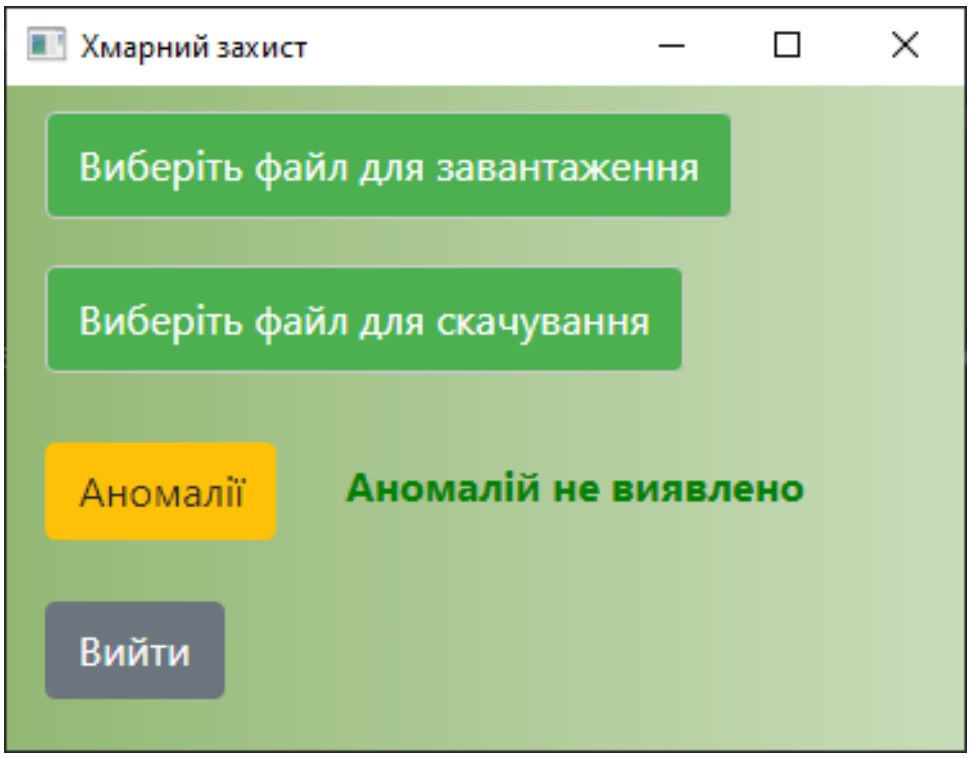


Рисунок 3.4 – Головна сторінка програми

Після авторизації в хмарному сервісі автоматично відбувається перехід до головної сторінки програми як показано на рисунку 3.4.

Отже, під час демонстрації графічного інтерфейсу було роз'яснено, як користуватися інтерфейсом програми. Також були надані рекомендації щодо взаємодії з інтерфейсом, обробки файлів та використання інших функцій програми.

ВИСНОВКИ

У даній роботі було проведено розробку програми для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту. Для цього було досліджено різні методи захисту даних та забезпечення безпеки доступу до хмарних сховищ, використовуючи сучасні технології та інструменти.

Було проаналізовано сучасні підходи до захисту даних у хмарних сховищах, розглянуто їхні переваги та недоліки. Було виявлено, що використання штучного інтелекту є дієвим рішенням для забезпечення високого рівня безпеки та ефективного управління доступом до даних. Після цього було проаналізовано методи інтеграції штучного інтелекту з системами безпеки, що дозволяє підвищити рівень захисту та забезпечити більш гнучке управління доступом.

Було здійснено обґрунтування вибору середовища розробки та мови програмування Python, оскільки це потужна мова, яка підтримує широкий спектр бібліотек для реалізації різних функціональностей, включаючи роботу з штучним інтелектом. Python також забезпечує високу швидкість розробки та підтримку спільноти, що є критично важливим у сучасних умовах.

Далі було розроблено програмний додаток, який був протестований, щоб переконатися в його працездатності та відповідності вимогам. Тестування включало перевірку правильності обмеження доступу до файлів, функцій захисту та інтеграції штучного інтелекту для забезпечення додаткових функцій аналізу та обробки даних.

Потім був здійснено огляд інтерфейсу програми, який дозволяє користувачам зручно та інтуїтивно використовувати додаток, надавати або обмежувати доступ до файлів та використовувати можливості штучного інтелекту для підвищення продуктивності.

Отже, можна зробити висновок, що запропоновані вдосконалення для захищеної віддаленої роботи у хмарному сховищі є досить актуальними та корисними, оскільки вони надають надійний захист від несанкціонованого доступу та забезпечують захист файлів користувачів. Використання штучного інтелекту додатково підвищує ефективність управління даними та безпекою, що робить цю систему важливим інструментом для сучасних користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stallings, W. (2018). *Cryptography and Network Security: Principles and Practice*. Pearson. URL: https://www.pearson.com/us/higher-education/program/Stallings-Cryptography-and-Network-Security-Principles-and-Practice-7th-Edition/PGM152901.html (дата звернення: 18.05.2024).
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. URL: https://www.deeplearningbook.org/ (дата звернення: 16.05.2023).
3. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications. URL: https://www.manning.com/books/deep-learning-with-python (дата звернення: 15.05.2023).
4. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. URL: https://www.springer.com/gp/book/9780387310732 (дата звернення: 17.05.2023).
5. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ (дата звернення: 18.05.2023).
6. Shiffman, D. (2015). *The Nature of Code*. The Magic Book Project. URL: https://natureofcode.com/ (дата звернення: 18.05.2023).
7. Kim, S., & Kim, K. (2019). *Advanced Security Technologies in Networking*. Springer. URL: https://www.springer.com/gp/book/9783030144882 (дата звернення: 19.05.2023).
8. Schneier, B. (2015). *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*. W.W. Norton & Company. URL:

https://www.schneier.com/books/data_and_goliath/ (дата звернення: 14.05.2023).

9. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html (дата звернення: 19.05.2023).
10. Dean, J., & Ghemawat, S. (2008). *MapReduce: Simplified Data Processing on Large Clusters*. Communications of the ACM. URL: https://dl.acm.org/doi/10.1145/1327452.1327492 (дата звернення: 18.05.2023).
11. Richard, S. (2017). *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. John Wiley & Sons. URL: https://www.wiley.com/enus/Cloud+Security%3A+A+Comprehensive+Guide+to+Secure+Cloud+Computing-p-9781119046949 (дата звернення: 17.05.2023).
12. Pope, B. (2019). *Cybersecurity: The Insights You Need from Harvard Business Review*. Harvard Business Review Press. URL: https://store.hbr.org/product/cybersecurity-the-insights-you-need-from-harvard-business-review/10409 (дата звернення: 15.05.2023).
13. Maheshwari, P., & Yadav, K. (2020). *Cloud Computing with Security*. CRC Press. URL: https://www.routledge.com/Cloud-Computing-with-Security/Maheshwari-adav/p/book/9780367421848 (дата звернення: 16.05.2023).
14. Zheng, J., & Abdallah, M. (2020). *Blockchain Technology for Cloud Storage*. Springer. URL: https://www.springer.com/gp/book/9783030433474 (дата звернення: 18.05.2023).
15. Kieseberg, P., & Schrittwieser, S. (2017). *Security and Privacy in Cloud Computing*. Springer. URL:

https://www.springer.com/gp/book/9783319514129 (дата звернення: 18.05.2023).

16. Wang, C., & Shroff, N. (2018). *Cloud Computing Security*. IEEE Journal on Selected Areas in Communications. URL: https://ieeexplore.ieee.org/document/8402564 (дата звернення: 19.05.2023).
17. Harris, S. (2019). *CISSP All-in-One Exam Guide*. McGraw-Hill Education. URL: https://www.mhprofessional.com/covers/9781260142655 (дата звернення: 15.05.2023).
18. Kumar, S., & Raj, P. (2018). *Mobile Edge Computing: A Primer*. Springer. URL: https://www.springer.com/gp/book/9783319977061 (дата звернення: 17.05.2023).
19. Zou, X., & Hua, Z. (2016). *High Performance Cloud Auditing and Applications*. Springer. URL: https://www.springer.com/gp/book/9783319273118 (дата звернення: 18.05.2023).
20. Peterson, L., & Davie, B. (2020). *Computer Networks: A Systems Approach*. Morgan Kaufmann. URL: https://www.elsevier.com/books/computer-networks/peterson/978-0-12-818200-0 (дата звернення: 18.05.2023).
21. Whitman, M. E., & Mattord, H. J. (2020). *Principles of Information Security*. Cengage Learning. URL: https://www.cengage.com/c/principles-of-information-security-6e-whitman/9781337102063 (дата звернення: 19.05.2023).
22. Tavildar, S., & Manickam, V. (2019). *Network Security Essentials: Applications and Standards*. Pearson. URL: https://www.pearson.com/store/p/network-security-essentials-

applications-and-standards/P100001093874 (дата звернення: 18.05.2023).

23. Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing*. Pearson. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 17.05.2023).
24. Tang, J., & Xiao, Y. (2019). *Security in Mobile Ad Hoc and Sensor Networks*. John Wiley & Sons. URL: https://www.wiley.com/en-us/Security+in+Mobile+Ad+Hoc+and+Sensor+Networks-p-9781119085146 (дата звернення: 18.05.2023).
25. Alpaydin, E. (2020). *Introduction to Machine Learning*. MIT Press. URL: https://mitpress.mit.edu/books/introduction-machine-learning-fourth-edition (дата звернення: 14.05.2023).
26. Nielson, M. (2015). *Neural Networks and Deep Learning*. Determination Press. URL: http://neuralnetworksanddeeplearning.com/ (дата звернення: 18.05.2023).
27. Russinovich, M., & Solomon, D. A. (2017). *Windows Internals*. Microsoft Press. URL: https://www.microsoftpressstore.com/store/windows-internals-part-1-system-architecture-processes-9780735684188 (дата звернення: 15.05.2023).
28. Rawat, D. B., & Zha, C. (2018). *Software-Defined Networking and Security*. Springer. URL: https://www.springer.com/gp/book/9783319985820 (дата звернення: 16.05.2023).
29. Sutton, R. (2019). *The NoSQL DBA*. O'Reilly Media. URL: https://www.oreilly.com/library/view/nosql-dba/9781492038276/ (дата звернення: 15.05.2023).

30. Zwicky, E. D., Cooper, S., & Chapman, D. B. (2017). *Building Internet Firewalls*. O'Reilly Media. URL: https://www.oreilly.com/library/view/building-internet-firewalls/9781449333484/ (дата звернення: 17.05.2023).

ДОДАТКИ

ДОДАТОК А. Технічне завдання

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор

_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:

«Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту»

08-72.БДР.010.00.068.ТЗ

Керівник бакалаврської дипломної роботи
доц., каф., МБІС:
_____ Шиян А. А.
"_____" _____

Вінниця – 2024 р.

1. Найменування та область застосування

Програма для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту. Область застосування: забезпечення безпеки користувачів під час віддаленої роботи у хмарному середовищі.

2.Підстава для розробки

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

3. Мета та призначення розробки

3.1 Мета розробки: розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту.

3.2 Призначення: розроблена програма виявляє та реагує на аномалії в поведінці користувачів.

4. Джерела розробки

4.1. Brownlee J. Machine Learning Mastery with Python: Understand Your Data, Create Accurate Models, and Work Projects End-to-End. Machine Learning Mastery, 2016.

4.2. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, 2017. 550 p.

4.3. Van Rossum G., Drake F. L. Python 3 Reference Manual. CreateSpace, 2009. 242 p.

4.4. Іванов Д. С., Петров В. А. Системи штучного інтелекту: основи та застосування. Київ: Наукова думка, 2021. 320 с.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програма повинна мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація програми не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програма повинна виконувати процес виявлення аномальної активності користувачів у хмарному сховищі.

5.2 Вимоги до надійності:

5.2.1 Програма повинна працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на навчання моделі КНН;

5.2.3 Програма повинна виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

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

6. Вимоги до програмної документації

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

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваної програми від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

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

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

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми		
2.	Аналіз предметної області обраної теми		
3.	Розробка алгоритму роботи		
4.	Написання бакалаврської роботи на основі розробленої теми		
5.	Передзахист бакалаврської дипломної роботи		
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи		
7.	Захист бакалаврської дипломної роботи		

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

- ПЗ до бакалаврської дипломної роботи;
- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента.

Технічне завдання до виконання прийняв _____Заремблюк Д. О.

ДОДАТОК Б. Лістинг програми

```
from werkzeug.security import generate_password_hash, check_password_hash
import sqlite3
def connect_db():
    return sqlite3.connect('users.db')
def create_users_table():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute(id, username, password)
    conn.commit()
    conn.close()
def register_user(username, password):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute(username)
    if cursor.fetchone():
        conn.close()
        return False
    cursor.execute(username, generate_password_hash(password))
    conn.commit()
    conn.close()
    return True
def login_user(username, password):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute(username)
    user = cursor.fetchone()
    conn.close()
    if user and check_password_hash(user[2], password):
        return True
    return False
create_users_table()
from cryptography.fernet import Fernet
from config import Config
def generate_key():
    return Fernet.generate_key()
def encrypt_file(file_path, key=Config.ENCRYPTION_KEY):
    with open(file_path, 'rb') as file:
        data = file.read()
    fernet = Fernet(key)
    encrypted = fernet.encrypt(data)
    with open(file_path, 'wb') as file:
        file.write(encrypted)
def decrypt_file(file_path, key=Config.ENCRYPTION_KEY):
    with open(file_path, 'rb') as file:
        data = file.read()
    fernet = Fernet(key)
    decrypted = fernet.decrypt(data)
    with open(file_path, 'wb') as file:
        file.write(decrypted)
import logging
import config
logging.basicConfig(filename=config.LOG_FILE, level=logging.INFO)
def monitor_activity():
```

```
    logging.info("Моніторинг активності користувача")
from google.oauth2 import service_account
from googleapiclient.discovery import build
SCOPES = ['https://www.googleapis.com/auth/drive.file']
SERVICE_ACCOUNT_FILE = 'service_account.json'
credentials = service_account.Credentials.from_service_account_file(
        SERVICE_ACCOUNT_FILE, scopes=SCOPES)
service = build('drive', 'v3', credentials=credentials)
def upload_file_to_gdrive(file_path, gdrive_file_name):
    file_metadata = {'name': gdrive_file_name}
    media = MediaFileUpload(file_path, resumable=True)
    file = service.files().create(body=file_metadata, media_body=media, fields='id').execute()
    return file.get('id')
def download_file_from_gdrive(gdrive_file_id, download_path):
    request = service.files().get_media(fileId=gdrive_file_id)
    fh = io.FileIO(download_path, 'wb')
    downloader = MediaIoBaseDownload(fh, request)
    done = False
    while done is False:
        status, done = downloader.next_chunk()
    return done
import numpy as np
import pickle
def detect_anomalies(data):
    with open('model.pkl', 'rb') as model_file:
        model = pickle.load(model_file)
    predictions = model.predict(data)
    anomalies = data[predictions == -1]
    return anomalies
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QWidget, QPushButton, QLabel,
QLineEdit, QFileDialog
from PyQt5.QtGui import QIcon
from google.oauth2 import service_account
from googleapiclient.discovery import build
import os
import json
import pickle
from datetime import datetime
from anomaly_detection import detect_anomalies
from encryption import encrypt_file, decrypt_file
from auth import login_user, register_user
from config import Config
import socket
import requests
app = Flask(__name__)
app.config.from_object(Config)
class AuthWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Авторизація")
        self.setGeometry(100, 100, 400, 300)
        self.setWindowIcon(QIcon('icon.png'))
        self.layout = QVBoxLayout()
        self.login_label = QLabel("Логін")
        self.layout.addWidget(self.login_label)
        self.login_input = QLineEdit()
```

```
        self.layout.addWidget(self.login_input)
        self.password_label = QLabel("Пароль")
        self.layout.addWidget(self.password_label)
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)
        self.layout.addWidget(self.password_input)
        self.login_button = QPushButton("Увійти")
        self.login_button.clicked.connect(self.handle_login)
        self.layout.addWidget(self.login_button)
        self.register_button = QPushButton("Реєстрація")
        self.register_button.clicked.connect(self.handle_register)
        self.layout.addWidget(self.register_button)
        container = QWidget()
        container.setLayout(self.layout)
        self.setCentralWidget(container)
    def handle_login(self):
        username = self.login_input.text()
        password = self.password_input.text()
        if login_user(username, password):
            self.open_main_window()
        else:
            self.login_label.setText("Невірний логін або пароль")
    def handle_register(self):
        username = self.login_input.text()
        password = self.password_input.text()
        if register_user(username, password):
            self.login_label.setText("Реєстрація успішна")
        else:
            self.login_label.setText("Помилка реєстрації")
    def open_main_window(self):
        self.main_window = MainWindow(username=self.login_input.text())
        self.main_window.show()
        self.close()
class MainWindow(QMainWindow):
    def __init__(self, username):
        super().__init__()
        self.setWindowTitle("Хмарний захист")
        self.setGeometry(100, 100, 400, 300)
        self.setWindowIcon(QIcon('icon.png'))
        self.username = username
        self.layout = QVBoxLayout()
        self.upload_button = QPushButton("Виберіть файл для завантаження")
        self.upload_button.clicked.connect(self.upload_file)
        self.layout.addWidget(self.upload_button)
        self.download_button = QPushButton("Виберіть файл для скачування")
        self.download_button.clicked.connect(self.download_file)
        self.layout.addWidget(self.download_button)
        self.anomaly_label = QLabel("Аномалії: Аномалій не виявлено")
        self.layout.addWidget(self.anomaly_label)
        self.exit_button = QPushButton("Вийти")
        self.exit_button.clicked.connect(self.close)
        self.layout.addWidget(self.exit_button)
        container = QWidget()
        container.setLayout(self.layout)
        self.setCentralWidget(container)
    def upload_file(self):
        options = QFileDialog.Options()
```

```
        file_path, _ = QFileDialog.getOpenFileName(self, "Виберіть файл для завантаження", "", "All Files
(*);;Python Files (*.py)", options=options)
        if file_path:
            self.log_user_activity(action="upload", file_path=file_path)
            encrypt_file(file_path)
            self.upload_to_google_drive(file_path)
    def download_file(self):
        options = QFileDialog.Options()
        file_path, _ = QFileDialog.getOpenFileName(self, "Виберіть файл для скачування", "", "All Files
(*);;Python Files (*.py)", options=options)
        if file_path:
            self.log_user_activity(action="download", file_path=file_path)
            self.download_from_google_drive(file_path)
            decrypt_file(file_path)
    def upload_to_google_drive(self, file_path):
        credentials = service_account.Credentials.from_service_account_file(
            app.config['SERVICE_ACCOUNT_FILE'], scopes=app.config['SCOPES'])
        service = build('drive', 'v3', credentials=credentials)
        file_metadata = {'name': os.path.basename(file_path)}
        media = MediaFileUpload(file_path, mimetype='application/octet-stream')
        file = service.files().create(body=file_metadata, media_body=media, fields='id').execute()
        print(f"File ID: {file.get('id')}")
    def download_from_google_drive(self, file_id, file_path):
        credentials = service_account.Credentials.from_service_account_file(
            app.config['SERVICE_ACCOUNT_FILE'], scopes=app.config['SCOPES'])
        service = build('drive', 'v3', credentials=credentials)
        request = service.files().get_media(fileId=file_id)
        with open(file_path, 'wb') as fh:
            downloader = MediaIoBaseDownload(fh, request)
            done = False
            while done is False:
                status, done = downloader.next_chunk()
                print(f"Download {int(status.progress() * 100)}%.")
    def check_anomalies(self):
        data = self.collect_user_activity_data()
        anomalies = detect_anomalies(data)
        if anomalies:
            self.anomaly_label.setText("Аномалії виявлено!")
        else:
            self.anomaly_label.setText("Аномалій не виявлено")
    def collect_user_activity_data(self):
        with open(Config.MONITOR_LOG_FILE, 'r') as log_file:
            logs = log_file.readlines()
        data = []
        for log in logs:
            log_data = log.strip().split(',')
            data.append({
                'timestamp': log_data[0],
                'username': log_data[1],
                'action': log_data[2],
                'file_path': log_data[3],
                'local_ip': log_data[4],
                'public_ip': log_data[5]
            })
        return data
    def log_user_activity(self, action, file_path):
        local_ip, public_ip = self.get_user_ip()
```

```
        with open(Config.MONITOR_LOG_FILE, 'a') as log_file:
            log_file.write(f"{datetime.now()},{self.username},{action},{file_path},{local_ip},{public_ip}\n")

    def get_user_ip(self):
            hostname = socket.gethostname()
        local_ip = socket.gethostbyname(hostname)
            try:
            public_ip = requests.get('https://api.ipify.org').text
        except requests.RequestException as e:
            public_ip = "Не вдалося отримати публічну IP-адресу"
        return local_ip, public_ip
if __name__ == '__main__':
    app = QApplication(sys.argv)
    auth_window = AuthWindow()
    auth_window.show()
    sys.exit(app.exec_())
```

ДОДАТОК В. Ілюстративний матеріал

Вінницький національний технічний університет

Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту

Виконав: студент
4 курсу групи
2КІТС-20б
Заремблюк Д. О.

Вінниця -2024

Мета: Здійснення розробки програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту для виявлення кіберзагроз

Об'єкт: процеси забезпечення безпеки користувачів під час віддаленої роботи у хмарному сховищі

Предмет: метод автоматизації виявлення кіберзігроз при роботі з хмарними сховищами за допомогою штучного інтелекту

Актуальність теми: У сучасному цифровому світі, де інформація є одним з найцінніших ресурсів, питання безпеки стає надзвичайно важливим. Використання хмарних технологій значно спрощує доступ до даних і дозволяє працювати віддалено, але також створює нові виклики у сфері кібербезпеки. З розвитком технологій і збільшенням обсягу переданої інформації питання захисту у хмарному середовищі стає ще більш актуальним.

Основні теоретичні аспекти розробки програми

1. Хмарні технології
Хмарні обчислення дозволяють користувачам отримувати доступ до ресурсів через Інтернет на вимогу. Вони включають зберігання даних, обробку даних та управління ресурсами.
Переваги хмарних технологій:

- Гнучкість: Можливість швидко адаптуватися до змін у вимогах бізнесу.
- Економічність: Платите тільки за ті ресурси, які використовуєте.
- Доступність: Доступ до ресурсів з будь-якої точки світу.

2. Машинне навчання
Машинне навчання (ML) — це підгалузь штучного інтелекту, яка дозволяє системам самостійно навчатися та вдосконалюватися на основі даних без явного програмування.
Типи машинного навчання:

- Кероване навчання : Модель навчається на маркованих даних.
- Некероване навчання : Модель виявляє закономірності у невизначених даних.
- Напівкероване навчання : Комбінація маркованих та немаркованих даних.

Основні теоретичні аспекти розробки програми

3. Виявлення аномалій
Виявлення аномалій (Anomaly Detection) — це процес виявлення невідповідностей у даних, які не відповідають очікуваній поведінці. Це важливо для виявлення потенційних загроз і аномальних дій у хмарних системах.
Методи виявлення аномалій:

- Статистичні методи: Використання статистичних властивостей для виявлення аномалій.
- Методи на основі машинного навчання : Використання алгоритмів, таких як K-Nearest Neighbors (KNN),для класифікації аномалій.

4. Захист даних і безпека
Захист даних у хмарному середовищі включає різні методи та технології для забезпечення конфіденційності, цілісності та доступності даних.
Методи захисту:

- Шифрування : Використання криптографічних методів для захисту даних.
- Аутентифікація та авторизація: Перевірка особи користувача та надання відповідних прав доступу.
- Моніторинг та аудит: Постійний моніторинг активності користувачів та аудит дій для виявлення потенційних загроз.

Аналіз існуючих методів

1

Класичні методи захисту

VPN (Virtual Private Network): VPN забезпечує шифрування трафіку між користувачем і сервером, що робить його важливим інструментом для захисту даних під час віддаленої роботи. Проте, VPN сам по собі не може виявити аномальну активність або атакуючих, які вже знаходяться всередині мережі.

2

Методи виявлення та реагування на інциденти

IPS (Intrusion Prevention System) Системи запобігання вторгнень працюють подібно до IDS, але також мають можливість автоматично блокувати підозрілі дії. IPS є більш проактивним рішенням, але можуть створювати помилкові спрацьовування (false positives).

3

Методи на основі штучного інтелекту

Виявлення аномалій (Anomaly Detection) Використання алгоритмів машинного навчання для аналізу поведінки користувачів і виявлення відхилень від норми. Наприклад, KNN (K-Nearest Neighbors) може бути використаний для класифікації дій користувачів як нормальних або аномальних. Цей метод дозволяє виявляти невідомі раніше загрози.

Обґрунтування вибору методу виявлення аномалій за допомогою алгоритму K-найближчих сусідів (KNN)

Це рішення базується на використанні алгоритмів машинного навчання, таких як KNN (K-Nearest Neighbors), що дозволяє аналізувати поведінку користувачів і класифікувати їхні дії як нормальні або аномальні Ось ключові причини вибору цього методу:

1. Ефективність

Алгоритми виявлення аномалій не обмежуються відомими сигнатурами загроз. Це означає, що вони можуть виявляти нові, раніше невідомі загрози, аналізуючи поведінку користувачів і визначаючи відхилення від нормальної діяльності. Таким чином, ми можемо забезпечити більш динамічний і адаптивний захист.

2. Гнучкість та адаптивність

KNN та інші алгоритми машинного навчання можуть бути легко адаптовані до різних сценаріїв використання. Вони здатні навчатися на основі історичних даних користувачів і постійно покращувати точність виявлення аномалій. Це дозволяє нашій системі бути гнучкою і адаптивною до змін у поведінці користувачів та нових загроз.

3. Масштабованість

Система на основі KNN може бути масштабована для роботи з великими обсягами даних. Це важливо в умовах хмарного сховища, де кількість користувачів і обсяг даних можуть значно варіюватися Алгоритм може бути оптимізований для роботи в розподілених середовищах, що дозволяє обробляти великі дані в реальному часі

Архітектура навчання моделі машинногго навчання

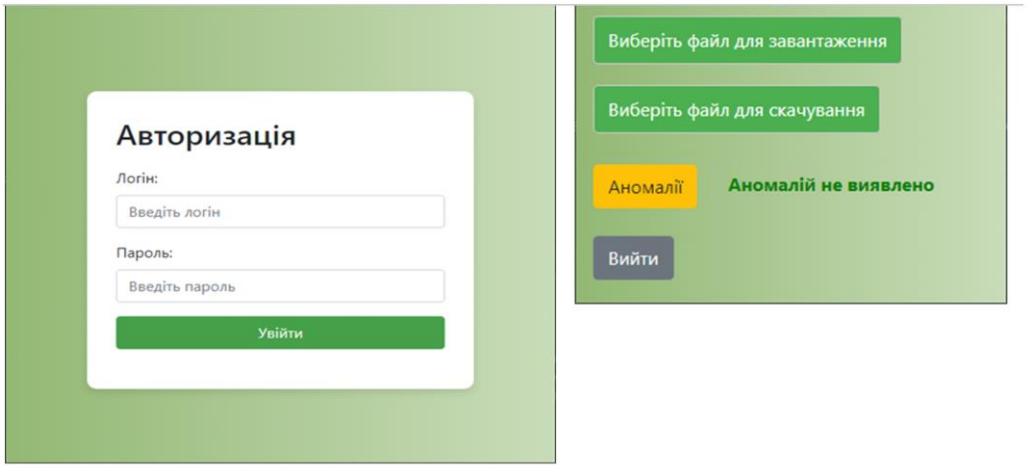
Архітектура навчання моделі KNN для виявлення аномалій у хмарному середовищі включає кілька ключових компонентів і етапів:

1. Збір даних.
Опис: Збір даних про активність користувачів з різних джерел, таких як логи доступу до файлів, мережеві логи і дані з хмарного сховища. Дані можуть включати IP-адреси, час доступу, обсяг завантажених даних тощо. Приклад: Лог доступу, що містить IP-адресу, час доступу, файл, до якого був доступ, і обсяг завантажених даних.

2. Попередня обробка даних.
Очищення даних: Видалення дублікатів, заповнення відсутніх значень.Нормалізація даних: Перетворення даних у спільний масштаб для забезпечення коректної роботи алгоритму KNN.Витягнення фіч: Вибір релевантних особливостей, які можуть бути корисними для виявлення аномалій.

3. Оцінка моделі.
Тестування моделі: Використання тестового набору даних для оцінки точності моделі.
4. Розгортання моделі.
Інтеграція з хмарним середовищем: Впровадження моделі у хмарну інфраструктуру для моніторингу активності користувачів в режимі реального часу

Загальний вигляд програмного додатку



Висновок

У даній роботі було розроблено програмний додаток для забезпечення захищеної віддаленої роботи користувачіву хмарному сховищі з використаннямштучного інтелекту. В ній поєднуютьсяметоди машинного навчаннядля виявлення аномалійу поведінці користувачівта інтеграцію з хмарним сервісом.

Програмний додаток має зручний графічний інтерфейс, який дозволяє користувачам завантажуватита переглядати файли у хмарному сховищі, а також отримуватисповіщення про виявлені аномалії. Програма аналізує активність користувачівза допомогою навченої моделі машинного навчанняі автоматично реагуєна підозрілуповедінку, запобігаючи можливим загрозам.

Розроблений програмний додаток є корисним інструментомдля організацій та користувачів,які прагнуть підвищити рівень кібербезпеки та забезпечити захист під час віддаленої роботи. Він допомагає швидко та ефективно ідентифікуватипотенційні загрози, мінімізуючи ризики, пов'язані з роботою у хмарному середовищі.

ДЯКУЮ ЗА УВАГУ!

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ
ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

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

Показники звіту подібності Unicheck

Оригінальність 96 % Схожість 4 %

Аналіз звіту подібності (відмітити потрібне):

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

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

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

Автор роботи _____ Заремблюк Д.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Шиян А.А.
(підпис) (прізвище, ініціали)