**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**
**Кафедра інформаційної безпеки**

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2024 р.

# Магістерська дисертація

**на здобуття ступеня магістра**

**за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»**
**спеціальності 125 «Кібербезпека»**

на тему: Модель майнінгу прав в хмарному середовищі на основі політики ABAC

Виконав здобувач ступеня магістра II курсу, групи ФБ-21мп

Купрієнко Артем Олександрович
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент кафедри інформаційної безпеки, канд. техн. наук, Гальчинський Л. Ю.

(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«___»_____2024 р.

# ЗАВДАННЯ
**на магістерську дисертацію здобувачу вищої освіти**
Купрієнку Артему Олександровичу

(прізвище, ім'я, по батькові)

1. Тема дисертації
«Модель майнінгу прав в хмарному середовищі на основі політики ABAC
науковий керівник дисертації доцент кафедри ІБ
к.т.н., доц. Гальчинський Леонід Юрійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» жовтня 2023 р. № 5236-с

2. Термін подання здобувачем вищої освіти дисертації «8» січня 2024 р.

3. Вихідні дані до роботи

_____

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

_____

6. Орієнтовний перелік публікацій: стаття “Агентна модель майнінгу прав доступу в хмарних середовищах”

7. Дата видачі завдання 10.09.2023

Календарний план

| № з/п | Назва етапів виконання магістерської дисертації | Термін виконання етапів дипломної роботи | Примітка |
|---|---|---|---|
| 1 | Вибір тематики | 10.05.2023 - 20.06.2023 | виконано |
| 2 | Огляд наукової літератури за темою дисертації | 01.07.2023 - 01.08.2023 | виконано |
| 3 | Написання оглядового розділу магістерської дисертації | 15.08.2023 - 15.09.2023 | виконано |
| 4 | Аналіз хмарних технологій та методів захисту | 20.09.2023 - 30.09.2023 | виконано |
| 5 | Вибір хмарного середовища | 01.10.2023 - 05.10.2023 | виконано |
| 6 | Написання програмної реалізації | 10.10.2023 - 10.12.2023 | виконано |
| 7 | Проходження переддипломної практики | 01.09.2023 - 26.10.2023 | виконано |
| 8 | Створення презентації для доповіді | 05.01.2024 - 07.01.2024 | виконано |
| 9 | Передзахист магістерської дисертації | 08.01.2024 - 08.01.2024 | виконано |
| 10 | Доопрацювання дисертації | 09.01.2024 - 14.01.2024 | виконано |
| 11 | Захист магістерської дисертації | 15.01.2024 - 15.01.2024 | виконано |

Здобувач ступеня магістра _____ Артем КУПРІЄНКО
**(підпис)** **(Власне ім’я, ПРІЗВИЩЕ)**

Керівник роботи _____ Леонід ГАЛЬЧИНСЬКИЙ
(підпис) **(Власне ім’я, РІЗВИЩЕ)**

# РЕФЕРАТ

Робота обсягом 92 сторінки містить 34 рисунки, 2 таблиці, 28 джерел за переліком посилань та 1 додаток

Метою роботи є дослідження наявних доступів користувачів на основі атрибутів та метод майнінгу прав в залежності їх прав та атрибутів

Об’єктом дослідження є механізм управління доступом на основі атрибутів користувачів

Методи дослідження: теоретичний, аналіз систем, що існують, аналіз наукових джерел, практичне використання

Результатом роботи є алгоритм знаходження, виявлення та усунення надлишкових прав користувачів розроблений за допомогою мови Python на хмарному постачальнику AWS

**Ключові слова**: AWS сервіси, майнінг алгоритм на основі ABAC, хмарне середовище, тестування на доступність

# ABSTRACT

The paper consists of 92 pages, 34 figures, 2 tables, 28 references and 1 appendix.

The purpose of the work is to study the existing user accesses based on attributes and the method of mining rights depending on their rights and attributes

The object of research is user attributes as the main mechanism of access control

Research methods: theoretical, analysis of existing systems, analysis of scientific sources, practical use

The result of the work is an algorithm for finding, detecting and eliminating redundant user rights developed using Python on the AWS cloud provider

**Keywords:** AWS services, ABAC-based mining algorithm, cloud environment, availability testing

# ЗМІСТ

# Перелік умовних позначень, символів, одиниць, скорочень і термінів

RBAC – Role-Based Access Control

DAC – Discretionary Access Control

MAC – Mandatory Access Control

IAM – Identity and Access Management

ОЗП - оперативний запам'ятовувальний пристрій

ХС – хмарне середовище

FOSS - Free Open Source Software, Відкрите програмне забезпечення

MPNABAC - Mining Positive and Negative Attribute-Based Access Control

ЛФА - лямбда функція агент

ЛФМ - лямбда функція майнер

CSP - Cloud Service Provider, постачальник хмарних послуг

# Вступ

**Актуальність** даної роботи обумовлена тим, що більшість систем, які основані на атрибутах, не мають можливості динамічно адаптувати наявні права доступу. Дана робота призначена для того, щоб зробити менш часозатратним та більш автоматизований процес зміни прав доступу на основі атрибутів у хмарному середовищі

**Метою** цієї роботи є дослідження процесу автоматизації знаходження, аналізу та призначення, відповідно до наявних атрибутів, надання прав доступу на основі атрибутів у хмарному середовищі

**Об'єктом дослідження** є механізм управління доступом атрибути користувачів на основі атрибутів користувачів

**Предметом дослідження** є автоматичне виявлення та усунення надлишкових прав доступу користувачів на основі атрибутів

**Задачами** є:

1. Дослідити наявні хмарні середовища, їх політики доступу в якості потенційного кандидата для поставленої задачі. Обрати найбільш використовуваний та адаптувати під цілі задачі
2. Дослідити як працює ABAC та як він може бути імплементований в обраному хмарному середовищі
3. Створити достатню кількість користувачів з різними атрибутами для проходження симуляції по майнінгу прав на ефективність роботи
4. Розробити алгоритм майнінгу прав
5. Розробити механізм присвоєння нових прав
6. Зробити перевірку наявних прав та аналіз потенційних конфліктних прав доступу

7. Зробити велику кількість користувачів для перевірки швидкості виконання алгоритму під великим навантаженням
8. Порівняти зі схожими алгоритмами та зробити висновки

**Апробація результатів.** Роботу було апробовано на 3-ій Міжнародній науково-практичній конференції "ІННОВАЦІЙНИЙ РОЗВИТОК У ГЛОБАЛЬНІЙ НАУЦІ", опубліковано статтю у матеріалах конференції під назвою “Агентна модель майнінгу прав доступу в хмарних середовищах”

**Новизна результатів** полягає у розробці працездатного рішення у вигляді алгоритму для знаходження, виявлення та усунення надлишкових прав доступу користувачів на основі їх атрибутів в реальному часі

# 1 Аналіз предметної області

## 1.1 Ідея хмарних технологій

Хмарні технології (cloud technologies) - це сукупність інформаційних технологій, які дозволяють доступатися до різних ресурсів, таких як обчислювальна потужність, зберігання даних чи програмне забезпечення, через Інтернет. Замість того, щоб мати власні фізичні сервери або обладнання, користувачі можуть використовувати ресурси, які знаходяться в "хмарі" - доступ до яких забезпечується через мережу Інтернет. Це забезпечує більшу мобільність, гнучкість і зручність у використанні ресурсів.

Хмарні обчислення (cloud computing) - це надання різноманітних обчислювальних послуг, таких як сервери, сховища даних, бази даних, мережі, програмне забезпечення, аналітика та штучний інтелект, через Інтернет, аби забезпечити швидкість впровадження інновацій, гнучкість використання ресурсів та економію завдяки масштабованості.

Хмарні сервіси (cloud services) – це інфраструктура, платформи або програмне забезпечення, які розміщуються сторонніми постачальниками та надаються користувачам через Інтернет. На основі вище вказаних можливостей надання послуг використання комп’ютерних сервісів можна виділити переваги такого підходу:

1. **Скасування географічних обмежень:** Хмарні сервіси створюють можливість будь-яким організаціям працювати над проєктами в режимі реального часу, подолавши обмеження відстані та часові різниці.

2. **Миттєва масштабованість:** Гнучкість в горизонтальному або вертикальному масштабуванні інфраструктури за допомогою хмарних ресурсів дозволяє ефективно впоратися з великими обсягами даних та високими завантаженнями.
3. **Інновації та оновлення:** Завдяки хмарним оновленням, компанії можуть оперативно впроваджувати нові функції та вдосконалення без затримок, що сприяє постійному розвитку.
4. **Економія енергоресурсів:** Централізоване зберігання та обробка даних може зменшити енергоспоживання порівняно з декількома розсіяними серверами.
5. **Мінімізація ризику втрати даних:** Резервне копіювання на різних географічно розташованих серверах зменшує ймовірність втрати даних внаслідок природних катастроф чи інших подій.
6. **Зниження інфраструктурних витрат:** Використання інфраструктури хмарних сервісів дозволяє уникати великих капіталовкладень у серверні парки.
7. **Гарантована доступність:** Забезпечення високої доступності через резервні сервери та системи забезпечення неперервної роботи, що знижує ризик відмов та перерв в роботі.

Але під час використання хмарних сервісів також потрібно знати, з якими проблемами можна зіштовхнутись:

- **Залежність від Інтернету:** Без доступу до Інтернету користувачі можуть втратити можливість отримувати доступ до своїх даних та використовувати хмарні сервіси.
- **Приватність та безпека даних**: Зберігання важливих даних в хмарі може породжувати питання стосовно конфіденційності та безпеки, особливо при виникненні кіберзагроз та порушень безпеки.

- **Вартість на довгостроковий період:** Витрати на використання хмарних сервісів можуть зростати з часом, особливо при великому обсязі даних та великих вимогах до обчислювальних ресурсів.
- **Можливість відмови служби:** Якщо хмарний провайдер має проблеми або вирішує припинити свою діяльність, це може призвести до втрати доступу до даних та послуг.
- **Проблеми з сумісністю та інтеграцією:** Існують можливість проблем при інтеграції хмарних сервісів з наявними системами та програмами.

## 1.2 Аналіз хмарних сервісів

### Аналіз ринку хмарних сервісів

За даними від Євростату станом на 2021 рік 10 з 28 країн Європи використовують хмарні технології в більше ніж 50% бізнесів, у 15 країнах частка бізнесів, які взаємодіють з хмарними технологіями сягає від 20% до 40%. Порівняно з минулими роками[9] частка бізнесів, в яких так чи інакше залучення хмарних технологій збільшується

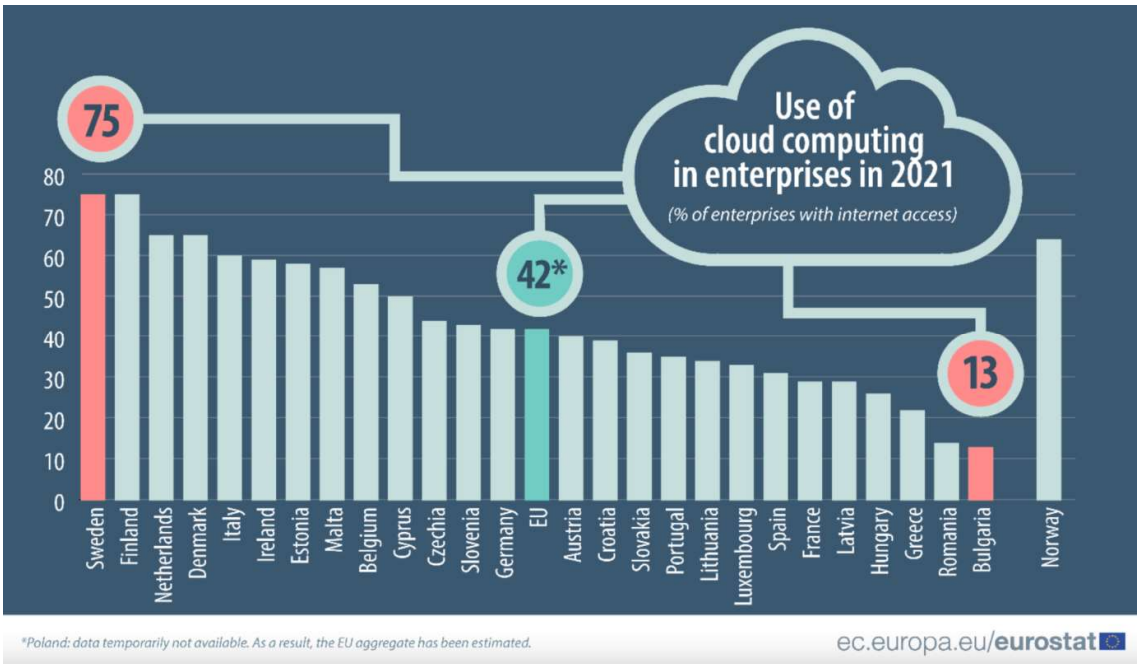


Рисунок 1.1 - Поширеність хмарних технологій в країнах ЄС. 2021 рік. [7]

Завдяки простоті та відносній дешевизні використання хмарних технологій можна побачити суттєві зміни в капіталізації хмарних сервісів з 2010 рік по 2020 рік Рисунок 1.2 та на 2022 рік Рисунок 1.3, що спонукає розвивати та вдосконалювати даних напрямок діяльності

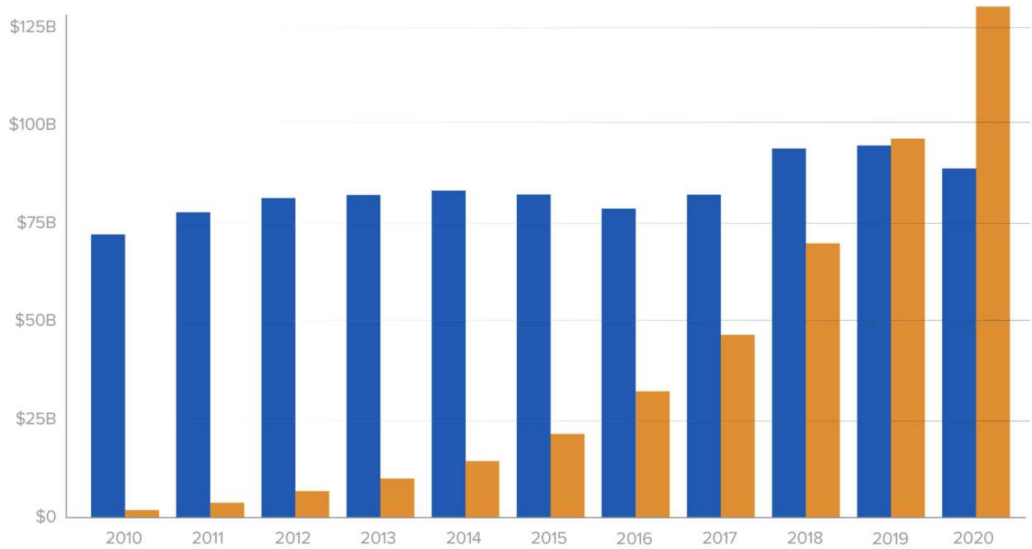

Рисунок 1.2 - Капіталізація хмарних сервісів 2010 р. - 2020 р. [10]

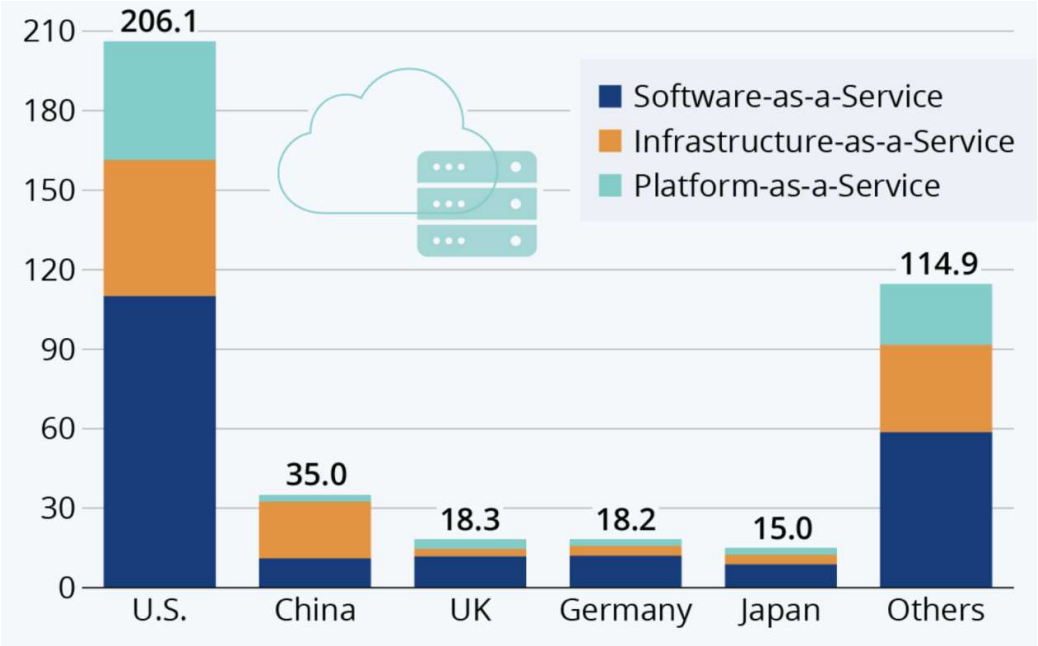


Рисунок 1.3 - Капіталізація  за допомогою хмарних сервісів на 2022 р.[28]

**Класифікація хмарних середовищ**

Хмарні сервіси можна розділити за різними критеріями. Найпоширеніший спосіб поділу - за рівнем абстракції:

- Інфраструктура як послуга (IaaS): IaaS надає користувачам доступ до базової хмарної інфраструктури, включаючи сервери, мережі, сховища та операційні системи. Користувачі IaaS відповідають за розгортання та управління власним програмним забезпеченням і даними.

- Платформа як послуга (PaaS): PaaS надає користувачам доступ до готової хмарної платформи, яка включає інструменти та служби для розробки, розгортання та управління вебдодатками та мобільними додатками. Користувачі PaaS відповідають за розробку та управління своїм власним програмним забезпеченням, але не за управління базовою хмарною інфраструктурою.
- Програмне забезпечення як послуга (SaaS): SaaS надає користувачам доступ до готового програмного забезпечення, яке розгорнуто в хмарі. Користувачі SaaS не відповідають за розробку, розгортання або управління програмним забезпеченням.

  Також кожен з вище перерахованих хмарних сервісів може бути як проєкт відкритого програмного забезпечення

**Відкрите програмне забезпечення** (FOSS) — програмне забезпечення, код якого є відкритим для перегляду, модифікації та розповсюдження будь-якою особою. Відкрите програмне забезпечення часто розробляється спільнотою розробників, які роблять свій внесок у код, використовуючи модель спільного розвитку.

Існують різні типи відкритого програмного забезпечення, які можна використовувати для задоволення конкретних потреб організації або користувача. До них належать:

1. Вільне програмне забезпечення (FOSS): Вільне програмне забезпечення відповідає чотирьом принципам свободи, встановленим Фондом Вільного ПЗ(FSF[1]): свободі використовувати програмне забезпечення будь-яким способом, свободі вивчати, змінювати та поширювати програмне забезпечення,

свободі поширювати модифіковані версії програмного забезпечення, свободі поширювати програмне забезпечення безплатно або за плату.

2. Програмне забезпечення з відкритим кодом (OSS): Програмне забезпечення з відкритим кодом має відкритий для перегляду та модифікації код, але може не відповідати принципам свободи, встановленим FSF.
3. Програмне забезпечення з відкритим вихідним кодом (OSS): Програмне забезпечення з відкритим вихідним кодом має відкритий для перегляду та модифікації код, але не може бути розповсюджено безплатно.

Інші способи поділу хмарних сервісів включають:

4. За галуззю застосування: хмарні сервіси можна розділити на галузеві та універсальні. Галузеві хмарні сервіси розроблені для конкретних галузей, таких як фінанси, охорона здоров'я або роздрібна торгівля. Універсальні хмарні сервіси можуть використовуватися в будь-якій галузі.
5. За рівнем доступу: хмарні сервіси можна розділити на публічні, приватні та гібридні. Публічні хмарні сервіси доступні для широкого загалу. Приватні хмарні сервіси надаються лише певним організаціям. Гібридні хмарні сервіси поєднують публічні та приватні хмари.
6. За рівнем управління: хмарні сервіси можна розділити на керовані та некеровані. Керовані хмарні сервіси надаються провайдером хмарної інфраструктури, який відповідає за обслуговування та управління сервісом. Некеровані хмарні сервіси надаються провайдером

хмарної інфраструктури, але користувачі відповідають за обслуговування та управління сервісом.

Існує кілька моделей розгортання хмарних сервісів, які визначають, де і як саме знаходяться та управляються ресурси. Публічна хмара - це модель хмарних обчислень, де обчислювальні ресурси (сервери, сховища даних, програмне забезпечення) надаються стороннім постачальником хмарних послуг і стають доступними через Інтернет. Такий підхід робить ресурси доступними для широкого кола користувачів та організацій, забезпечуючи гнучкість у використанні, ефективне масштабування, плату лише за фактичне використання та швидкість розгортання нових інфраструктурних ресурсів. Противагою публічної хмари існує приватна хмара - це модель хмарних обчислень, де обчислювальні ресурси розгортані власником організації на їх власних серверах або віртуальних середовищах. Ця модель надає більший контроль і безпеку, оскільки інфраструктура знаходиться під власником. Вона може бути фізичним дата-центром або віртуальним приватним хмарним середовищем, але ресурси фізично ізольовані від інших організацій. Також можлива комбінація вище описаних моделей в гібридну хмару - це модель хмарних обчислень, що комбінує елементи приватної та публічної хмар для створення єдиної інфраструктури. Організація може використовувати приватну хмару для чутливих даних та завдань, а публічну хмару - для більшого кола людей та можливостей отримати дані. Окрім даних основних моделей розгортання існують ще такі типи як: спільна хмара та мультихмара

Спільна хмара - це модель хмарних обчислень, де кілька організацій з одного галузі чи області об'єднують свої ресурси для створення спільного хмарного середовища. Це спільне використання інфраструктури

дозволяє організаціям спільно використовувати ресурси, сприяючи більш ефективному використанню та розподілу витрат. Приватність та безпека можуть бути краще контрольовані в порівнянні з публічними хмарами

Мультихмара - це стратегія використання різних хмарних сервісів або провайдерів для покращення гнучкості, надійності та оптимізації хмарної інфраструктури.

Хмарні сервіси надають можливість зберігати дані  за допомогою хмарних сховищ

**Хмарні сховища** — це послуги, які надаються провайдерами хмарних сервісів, які дозволяють користувачам зберігати, керувати та отримувати доступ до своїх даних у віддаленому центрі обробки даних. Існує кілька різних типів хмарних сховищ, які можна класифікувати за різними критеріями. Залежно від типу даних, які можна зберігати, можна вибрати один або кілька типів хмарних сховищ.

З найпоширеніших типів хмарних сховищ:

1. **Об’ємне сховище:** Цей тип хмарного сховища призначений для зберігання великих обсягів даних, таких як відео, фотографії та архіви.
2. **Об’єктне сховище:** хмарне сховище схоже на об’ємне сховище, але пропонує додаткові функції, такі як масштабованість та доступність.
3. **Бази даних:** хмарне сховище призначене для зберігання даних, які використовуються для роботи баз даних.
4. **Файлове сховище:** хмарне сховище призначене для зберігання файлів, які можна легко використовувати та обробляти.

5. **Спільне сховище:** хмарне сховище дозволяє користувачам спільно використовувати дані.
6. **Сховища обробки даних:** призначені для зберігання даних, які використовуються для обробки даних, таких як дані для машинного навчання та штучного інтелекту.
7. **Сховища резервного копіювання:** призначені для зберігання резервних копій даних.

Взаємодія сервісів хмарних сервісів з кінцевим користувачем може відбуватися за допомогою трьох ключових механізмів: UI (інтерфейс користувача), API (інтерфейс програмування застосунків) та SDK (набір інструментів для розробки).

**UI (Інтерфейс користувача):** UI являє собою графічний або текстовий інтерфейс, через який користувачі можуть взаємодіяти з хмарними послугами. Це може бути вебінтерфейс у вигляді панелі керування або мобільний додаток, який дозволяє легко керувати та моніторити різні аспекти хмарних ресурсів.

**API (Інтерфейс програмування застосунків):** API надає користувачам можливість взаємодії з хмарними послугами через програми та скрипти. Це дає розробникам можливість автоматизувати операції, створювати власні застосунки або інтегрувати хмарні послуги з іншими системами. API може включати різні кінцеві точки для роботи з обчислювальними ресурсами, зберіганням, мережевими службами та іншими хмарними функціями. Використання API дозволяє користувачам максимально використовувати потужності хмари з власних застосунків.

**SDK (Набір інструментів для розробки):** SDK - це збірка інструментів, бібліотек та прикладів коду, яка допомагає розробникам швидше та легше інтегрувати та розробляти застосунки для хмарних послуг. Використання SDK спрощує процес розробки, оскільки надає готові компоненти для взаємодії з API та іншими ресурсами.

## 1.3 Провайдери хмарних сервісів

**CSP** - це компанія, яка надає користувачам доступ до обчислювальних ресурсів, хмарного зберігання, хмарного аналізу даних та інших хмарних послуг

**Аспекти CSP**

1. Інфраструктура: CSP має великі дата-центри, обладнані тисячами серверів, забезпечуючи високу доступність та масштабованість. Це дозволяє користувачам отримувати доступ до ресурсів за потребою, не маючи власних фізичних серверів.
2. Моделі обслуговування: CSP пропонує різні моделі обслуговування, такі як Інфраструктура як сервіс (IaaS), Платформа як сервіс (PaaS) та Програмне забезпечення як сервіс (SaaS). Це дає користувачам можливість вибирати та налаштовувати рівень управління та контролю над інфраструктурою.
3. Безпека та відповідність: Безпека є важливим аспектом для CSP, оскільки вони забезпечують безпеку даних користувачів, які зберігаються в хмарі. Це включає заходи з шифрування, аутентифікації та управління доступом. Відповідність із законодавством щодо захисту даних також є критичною.

4. Ціноутворення: CSP пропонують різні моделі ціноутворення, включаючи pay-as-you-go, зарезервовані інстанси та інші. Це дозволяє користувачам оптимізувати витрати в залежності від їхніх конкретних потреб та використання ресурсів.
5. Інновації та нові технології: CSP постійно впроваджують нові технології, такі як штучний інтелект, машинне навчання, блокчейн та інші, щоб надавати користувачам доступ до передових рішень без необхідності власних великих IT-інфраструктур.
6. Супровід та підтримка: CSP забезпечують технічну підтримку, моніторинг та управління, щоб гарантувати стабільність та продуктивність послуг для своїх користувачів.

Найбільш відомими провайдерами хмарних послуг є:

1. **Amazon Web Services (AWS):** AWS є найбільшим провайдером хмарних сервісів у світі та пропонує широкий спектр послуг, включаючи хмарне обчислення, хмарне зберігання, хмарний аналіз даних та хмарне машинне навчання
2. **Microsoft Azure**: Великий конкурент AWS, Azure володіє широким функціоналом для побудови, розгортання та управління застосунками через глобальну мережу даних Microsoft.
3. **Google Cloud Platform (GCP):** провайдер володіє разючим потенціалом у сферах штучного інтелекту, машинного навчання та аналітики, а також пропонує інструменти для розробки та управління застосунками.
4. **IBM Cloud:** Інтегрує в себе різноманітні сервіси, включаючи обчислення, зберігання, розумний аналіз та блокчейн. IBM також акцентує на безпеці та високій доступності.

5. **Alibaba Cloud:** Цей провайдер особливо сильний в Китаї та Азії, пропонуючи аналогічний набір послуг, що та AWS, але з фокусом на регіональні потреби.
6. **Oracle Cloud:** Спеціалізується на корпоративних клієнтах, пропонуючи послуги управління базами даних, обчислення та аплікацій.
7. **DigitalOcean:** Орієнтований на стартапи та розробників, DigitalOcean пропонує простий інтерфейс та широкий спектр послуг, включаючи віртуальні сервери та зберігання.

Кожен хмарний провайдер має власну модель ціноутворення. Їх можна поділити на:

- **Плати, скільки використовуєш (pay-as-you-go):** є однією з основних моделей ціноутворення в хмарних послугах. Ця модель дозволяє користувачам платити лише за фактично використані ресурси та послуги, без необхідності передплати або заздалегідь визначених обсягів
- **Зарезервовані інстанси:** Користувачі можуть зобов'язатися платити за фіксовану кількість ресурсів на довгий термін, що дозволяє знизити загальні витрати в порівнянні з pay-as-you-go. Це особливо ефективно для стабільних або передбачуваних навантажень.
- **Спот-інстанси:** Користувачі можуть придбати невикористані ресурси за значно меншу ціну, але тут є ризик, оскільки ціни можуть змінюватися в залежності від попиту. Ця модель особливо корисна для тих, хто готовий пристосуватися до коливань цін.
- **Фіксоване ціноутворення (Fixed Pricing):** Деякі провайдери можуть пропонувати фіксовану ціну для конкретних пакетів послуг, що може бути зручно для тих, хто бажає більш прогнозовані витрати.

- **Модель витрат (Usage-based model):** Користувачі платять лише за конкретні характеристики або функції, які вони використовують, а не за загальний обсяг ресурсів. Це може бути вигідним для тих, хто використовує обмежений набір функцій.
- **Безплатні рівні обслуговування (Freemium model):** Багато провайдерів пропонують безплатні пакети для обмежених об'ємів ресурсів, дозволяючи користувачам випробувати їхні послуги перед переходом до платних планів.

## 1.4 Архітектура хмарних технологій

**Хмарна архітектура** — це сукупність компонентів, які працюють разом для забезпечення хмарних послуг. Ці компоненти можуть бути фізичними, такими як сервери, системи зберігання даних та мережеві пристрої, або вони можуть бути програмними, такими як операційні системи, програмне забезпечення для управління та програмне забезпечення для розробки додатків.

Хмарна архітектура зазвичай поділяється на три основні рівні:

- Інфраструктурний рівень: Цей рівень відповідає за фізичні компоненти, які підтримують хмарну інфраструктуру. Він включає сервери, що є основою хмарної інфраструктури системи зберігання даних використовуються для зберігання даних, які обробляються або зберігаються в хмарі. Системи зберігання даних можуть бути фізичними або віртуальними, мережеві пристрої для з'єднання компонентів хмарної інфраструктури та забезпечення доступу до хмарних послуг. Мережеві пристрої, які можуть включати маршрутизатори, комутатори, брандмауери та інші фізичні компоненти.

- Платформний рівень: Цей рівень відповідає за надання базових послуг, які використовуються для розробки та розгортання хмарних додатків. Він включає операційні системи які забезпечують функціональність, необхідну для розробки та розгортання хмарних додатків. Операційні системи платформного рівня, як правило, надають такі функції, як управління процесами, управління пам'яттю та управління мережею, програмне забезпечення для управління використовується для управління хмарною платформою та забезпечення її надійності та безпеки. Програмне забезпечення для управління платформою включає інструменти для моніторингу, управління ресурсами та забезпечення безпеки та інші платформні послуги можуть включати такі послуги, як вебсервери, бази даних та інструменти розробки.
- Прикладний рівень: Цей рівень відповідає за надання конкретних хмарних послуг, таких як хмарні обчислення, хмарне зберігання, бази даних та аналіз даних.

Побудова безпеки хмарних технологій будується на трьох пунктах: авторизація, автентифікація та захист даних. Ці три основні аспекти визначають фундамент безпеки хмарних технологій, забезпечуючи високий рівень захисту і конфіденційності Рисунок 1.4. Автентифікація гарантує, що лише вповноважені користувачі мають доступ до системи, перевіряючи їхню ідентичність через різноманітні методи, такі як паролі, біометричні дані чи аутентифікація з двома чинниками. Авторизація визначає права доступу кожного користувача чи системи, регулюючи, до яких ресурсів вони мають доступ. Це важливий елемент управління доступом, який забезпечує принцип "принцип найменшого доступу" для запобігання несанкціонованому доступу. Захист даних забезпечує шифрування та інші методи, щоб убезпечити конфіденційність і цілісність

інформації. Це важливо, особливо в умовах хмарного середовища, де дані можуть передаватися через мережу та зберігатися на віддалених серверах. Разом ці пункти створюють комплексний підхід до безпеки хмарних технологій, що дозволяє підприємствам впевнено впроваджувати та використовувати сучасні хмарні рішення з високим рівнем захисту.

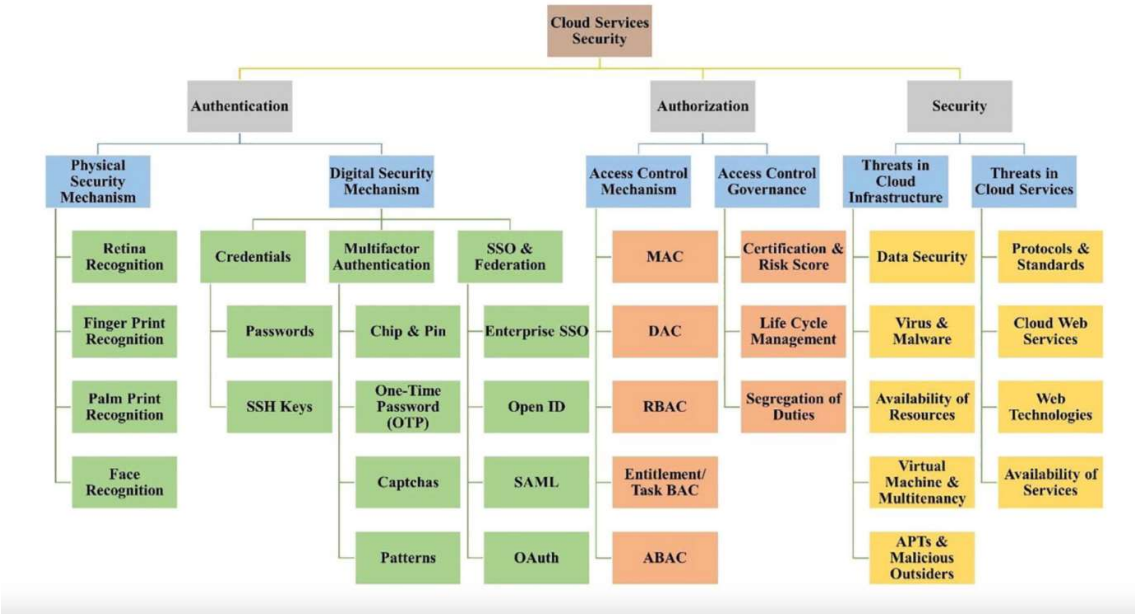


Рисунок 1.4 - Модель захисту хмарних технологій[27]

## 1.5 Механізми контролю доступу

Кожний хмарний провайдер надає інтерфейс для взаємодії з системою керуванням та управлінням доступом (Identity and access management). Identity and access management (IAM) - це набір процесів, політик і інструментів для визначення та управління ролями та правами доступу окремих мережевих об'єктів (користувачів та пристроїв) до різноманітних хмарних та локальних застосунків. Основна мета систем

IAM - це одна цифрова ідентичність на кожну особу чи предмет. Зазвичай IAM побудований на основі рольової моделі доступу.

Модель контролю доступу - це система правил та механізмів, яка визначає, як користувачі чи комп'ютерні системи отримують доступ до ресурсів чи інформації в комп'ютерній системі. Вона визначає, які користувачі чи системи мають право виконувати певні дії, а які - ні, забезпечуючи тим самим безпеку та конфіденційність інформації. Основними моделями розмежування доступу є мандатна модель доступу (Mandatory Access Control), дискреційна модель доступу (Discretionary access control), рольова модель доступу (Role based access control) та модель доступу на основі атрибутів (Attribute based access control). Взаємодія суб’єктів та об’єктів показана на Рисунок 1.5

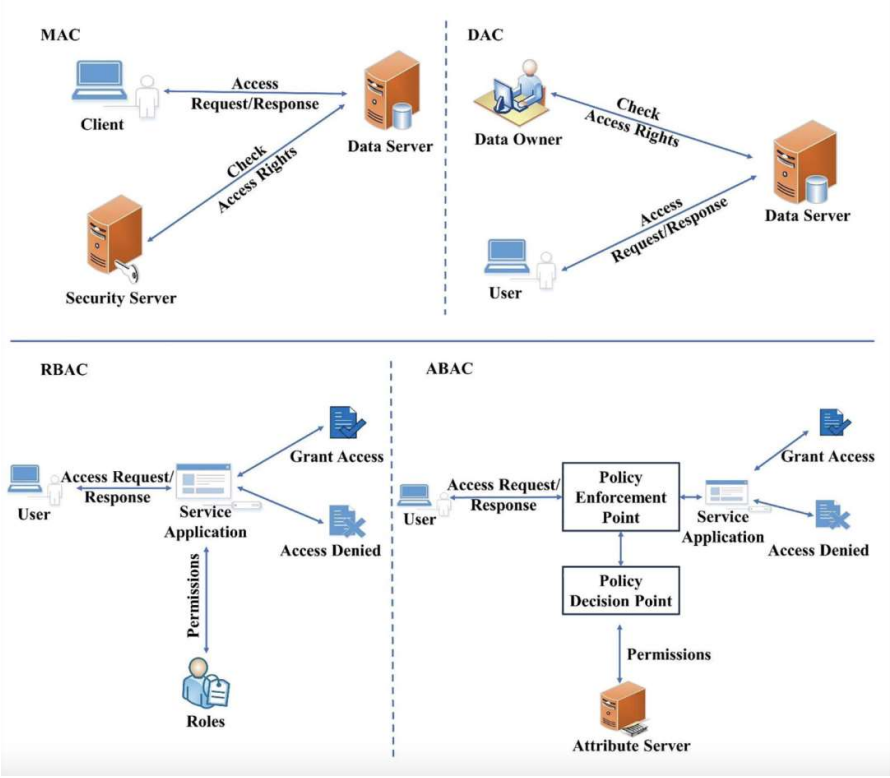


Рисунок 1.5 - Моделі доступу[26]

**MAC (мандатна модель доступу)** — це модель доступу, яка базується на повноваженнях. У MAC-системі ресурси мають повноваження, які визначають, хто може до них отримати доступ і що вони можуть робити з ними. Користувачі також мають повноваження, які визначають, до яких ресурсів вони мають доступ і що вони можуть робити з ними. Доступ до ресурсу дозволено, якщо повноваження користувача відповідають повноваженням ресурсу.

**DAC (дискреційна модель доступу)** — це модель контролю доступу, яка базується на розмежуванні. У DAC-системі ресурси мають власника, який відповідає за визначення того, хто має доступ до ресурсу і що вони можуть робити з ним. Власник ресурсу може надавати або забороняти доступ до ресурсу іншим користувачам.

**RBAC (рольова модель доступу)** — це модель доступу, яка базується на ролях. У RBAC-системі ресурси мають ролі, які визначають, хто може до них отримати доступ і що вони можуть робити з ними. Користувачі мають ролі, які визначають, до яких ресурсів вони мають доступ і що вони можуть робити з ними. Доступ до ресурсу дозволено, якщо роль користувача відповідає ролі ресурсу. Дана модель доступу є основною у головних постачальних хмарних сервісів: Amazon Web Services[2], Google Cloud Platform[3] та Microsoft Azure[4]

**ABAC (модель доступу на основі атрибутів)** — це модель доступу, яка базується на атрибутах. У ABAC-системі ресурси мають атрибути, які визначають, хто може до них отримати доступ і що вони можуть робити з ними. Користувачі також мають атрибути, які визначають, до яких ресурсів вони мають доступ і що вони можуть робити з ними. Доступ до ресурсу дозволено, якщо атрибути користувача відповідають атрибутам ресурсу.

Таблиця 1.1 - Порівняння моделей розмежування доступу

| Модель | Основа | Призначення |
|---|---|---|
| MAC | Повноваження | Забезпечення конфіденційності та цілісності даних |
| DAC | Розмежування | Контроль доступу до ресурсів |
| RBAC | Ролі | Забезпечення гнучкості та ефективності |
| ABAC | Атрибути | Забезпечення точності та адаптивності |

## 1.6 Механізми аутентифікації

Існує кілька різних механізмів автентифікації, які можна використовувати у хмарних сервісах:

- Логін та пароль: класичний метод, де користувачі вказують ідентифікатори (логін) та паролі для підтвердження своєї особи. Паролі є найпростішим і найзручнішим способом автентифікації. Даний метод доречно використовувати коли потрібна проста і зручна аутентифікація з мінімальною шкодою для системи у випадку несанкціонованого доступу або для аутентифікації користувачів з обмеженими правами, наприклад, користувачів, які мають доступ лише до певних даних або ресурсів. Їх можна ввести за допомогою будь-якого пристрою через мережу Інтернет
- Двофакторна аутентифікація (2FA)[5]: 2FA додає додатковий рівень безпеки до автентифікації за допомогою паролів. Крім пароля, користувач також повинен ввести код, який надсилається на його пристрій. 2FA може допомогти запобігти несанкціонованому доступу, навіть якщо пароль буде вкрадений.

- Аутентифікація за допомогою біометричних даних: Аутентифікація за допомогою біометричних даних використовує фізичні характеристики користувача, такі як відбиток пальця або обличчя, для автентифікації. Біометрична аутентифікація є дуже безпечною, оскільки її важко підробити.
- Аутентифікація за допомогою PKI[6]: PKI використовує асиметричні методи шифрування для забезпечення конфіденційності повідомлень та аутентифікації пристрою чи користувача, що відправляє передачу. Асиметричне шифрування включає використання відкритого та приватного ключа Рисунок 1.6. Відкритий ключ доступний для будь-якого, хто його запитує, і виданий довіреною сертифікаційною владою. Другий компонент криптографічної пари ключів - це приватний ключ. Цей ключ залишається приватним в одержувача зашифрованого повідомлення і використовується для розшифрування передачі. Відкритий ключ аутентифікує відправника цифрового повідомлення, тоді як приватний ключ гарантує, що тільки отримувач може відкрити та прочитати його.

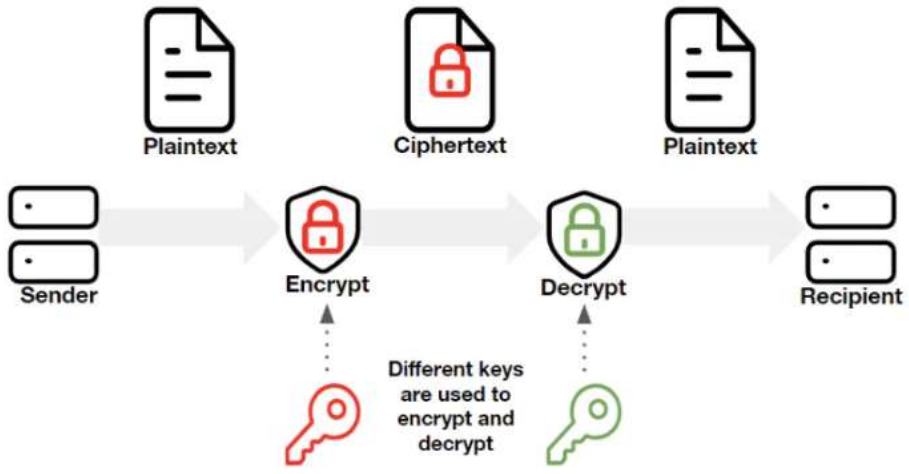


Рисунок 1.6 - Схема автентифікації PKI[25]

## 1.7 Кібербезпека хмарних сервісів

### Аналіз інформаційної безпеки

Дослідження (CPR)[11] оприлюднює нові дані про тенденції кібератак на 2022 рік Рисунок 1.7. Дані поділені за глобальним обсягом, галуззю та географією. Глобальні кібератаки зросли на 38% у 2022 році порівняно з 2021 роком. Ці цифри кібератак були викликані меншими, більш гнучкими групами хакерів і розбійників, які фокусувалися на використанні засобів співпраці, що використовуються в умовах роботи з дому, а також на націленні на навчальні установи, які перейшли до електронного навчання після COVID-19. Цей зріст глобальних кібератак також випливає із зацікавленості хакерів у медичних установах, які спостерігали за найбільшим зростанням кібератак у 2022 році порівняно із всіма іншими галузями Рисунок 1.8

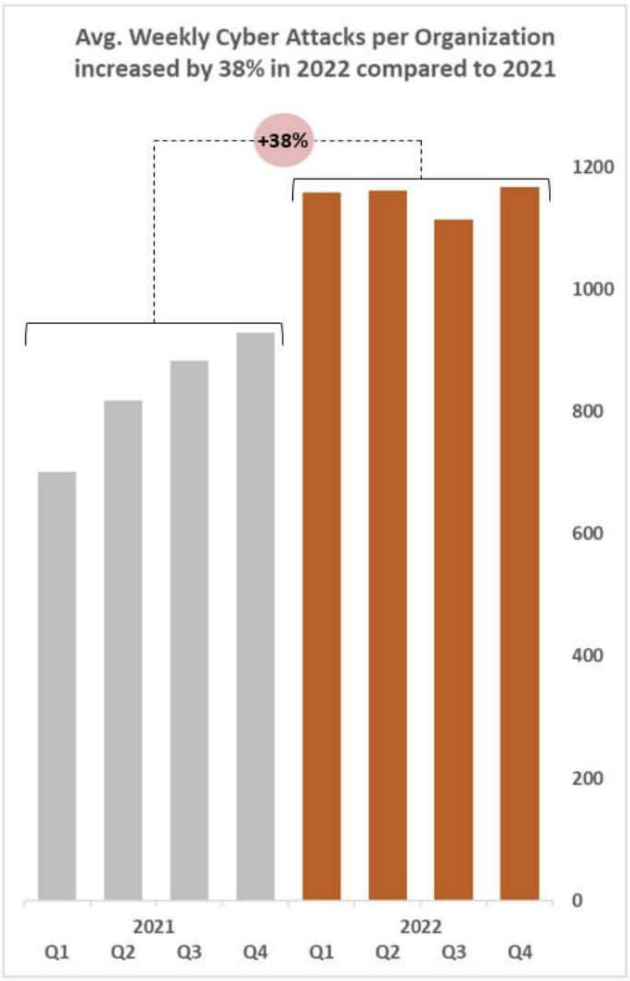


Рисунок 1.7 - Порівняння кібератак 2021 рік та 2022 рік.[24]

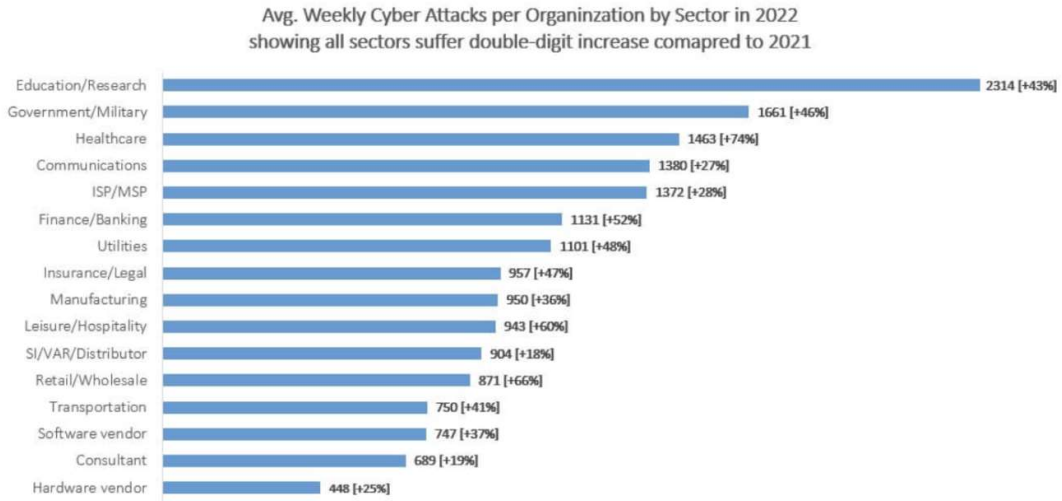

Рисунок 1.8 - Статистика кібератак по секторах[23]

У сфері кібербезпеки хмарних сервісів існує багато потенційних загроз та вразливостей, з якими можуть зіткнутися організації та індивідуальні користувачі. Ось деякі з них:

1. Неавторизований доступ: Хмарні сервіси доступні через Інтернет, що робить їх вразливими до атак на рівні аутентифікації та авторизації[17]. Це може включати крадіжку облікових даних, фішинг та інші методи отримання несанкціонованого доступу до хмарних ресурсів.
2. Конфігураційні помилки: Неправильно налаштовані хмарні сервіси можуть ненавмисно відкривати вразливі точки доступу, що може призвести до витоку даних або зловмисних атак.
3. Вразливості API: Хмарні сервіси часто використовують API для взаємодії з іншими сервісами та клієнтами. Вразливості в API можуть призвести до витоку даних, несанкціонованого доступу та інших кіберзагроз.
4. Внутрішні загрози: Співробітники організації можуть ненавмисно або умисно спричинити витік даних або інші безпекові інциденти у хмарі.
5. Шифрування даних: Хоча багато хмарних сервісів пропонують шифрування даних, існують ризики, пов'язані з управлінням ключами шифрування та можливістю їх компрометації.
6. Розподілені відмови в обслуговуванні (DDoS[18]): Хмарні сервіси можуть стати мішенями для DDoS-атак, що може призвести до перебоїв у роботі сервісу та втрати доступності для кінцевих користувачів.

7. Витік даних: Хмарні сервіси зберігають великі обсяги даних, які можуть містити конфіденційну інформацію. Витоки даних можуть відбуватися через багато векторів, включаючи крадіжку даних зловмисниками або через треті сторони, наприклад, постачальників хмарних послуг.
8. Хмарні віруси: Зловмисники можуть завантажити шкідливе ПЗ в хмарні системи, яке може поширюватися між користувачами або навіть цілими хмарними середовищами.
9. Спільне використання ресурсів: В хмарних середовищах ресурси часто спільно використовуються багатьма користувачами, що може призвести до ризиків ізоляції та розмежування ресурсів.

Ці загрози вимагають комплексного підходу до кібербезпеки, включаючи сильну політику безпеки, регулярні аудити та оцінки ризиків, а також використання інструментів безпеки для моніторингу та захисту хмарних сервісів.

**Кібератаки на українську інфраструктуру**

З початку повномасштабного вторгнення росії в Україну, Україна стала мішенню для численних кібератак, які були спрямовані на дестабілізацію кібернетичного простору держави та вплив на її здатність вести оборону. Найбільш розповсюдженими атаками були:

- Руйнівні атаки Були здійснені спроби знищення критичної інфраструктури через використання вірусів-знищувачів, таких як "wiper[19]" віруси, які видаляють важливі дані та перешкоджають роботі критичних систем.

- DDoS атаки: Українські урядові сайти, їх сервіси та бізнес зіткнулися з масштабними DDoS атаками, що перешкоджали доступу користувачів до важливої інформації та послуг.
- Фішинг та шпигунство: Кібератаки також включали запуск фішингових кампаній та розгортання шпигунського ПЗ для збору інформації, крадіжки даних або навіть спроби впливу на високопосадовців.
- Пропаганда та дезінформація: Через кіберпростір також розповсюджувались неправдиві новини та дезінформація з метою створення паніки серед населення та підриву довіри до уряду та армії.
- Атаки на енергетичну інфраструктуру: Відзначалися спроби атак на енергетичні системи, які могли мати на меті зупинити постачання електроенергії та інших життєво важливих послуг. Однією з таких атак була спрямована проти електроенергетичних компаній, яка використовувала шкідливе ПЗ яке було призначене для видалення та перезапису важливих файлів, що могло призвести до серйозних збоїв в енергосистемі та мала на меті відключити електропостачання та спричинити додатковий хаос та нестабільність у країні.

Український уряд та міжнародна спільнота активно працювали над зміцненням кіберзахисту країни, включаючи впровадження заходів безпеки, підвищення обізнаності громадян та співпрацю з міжнародними партнерами для виявлення та відбиття кібератак Рисунок 1.9.


Рисунок 1.9 - Статистика кібератак на українську інфраструктуру[22]

**Методи захисту від кібератак**

У світі кібербезпеки використовується широкий спектр методів захисту для запобігання, виявлення та реагування на кіберзагрози:

- Фаєрволи (Firewalls): Фаєрволи контролюють вхідний та вихідний мережевий трафік на основі визначених правил безпеки, ефективно функціонуючи як бар'єр між внутрішньою мережею та зовнішнім світом.
- Антивірусне ПЗ (Antivirus Software): Антивірусні програми виявляють, запобігають і видаляють шкідливе програмне забезпечення, таке як віруси, трояни та шпигунське ПЗ.
- Інтрузійні Системи Виявлення та Попередження (Intrusion Detection and Prevention Systems - IDS/IPS): Ці системи моніторять мережевий трафік на ознаки підозрілої активності та блокують атаки в реальному часі.

- Менеджмент Ідентифікації та Доступу (Identity and Access Management - IAM): IAM забезпечує, що лише уповноважені користувачі мають доступ до ресурсів IT-систем. Це може включати багатофакторну аутентифікацію, управління обліковими записами та контроль доступу.
- Системи Захисту від Розповсюдження DDoS-Атак (DDoS Mitigation): Ці системи захищають вебсайти та мережі від атак, які прагнуть перевантажити систему трафіком, ефективно блокуючи її роботу.
- Безпека Кінцевих Точок (Endpoint Security): Забезпечує захист кінцевих точок, таких як комп'ютери, мобільні пристрої та сервери, від різноманітних загроз.
- Системи Управління Безпекою Інформації та Подій (Security Information and Event Management - SIEM[20]): SIEM системи збирають і аналізують дані про безпеку з різних джерел, допомагаючи ідентифікувати, моніторити та реагувати на підозрілу діяльність.
- Політики Безпеки та Свідомість Користувачів[21]: Важливим елементом захисту є розробка та впровадження політик безпеки, а також навчання співробітників основам кібербезпеки.

Ці методи часто використовуються разом для створення багаторівневої стратегії захисту, яка може ефективно протистояти широкому спектру кіберзагроз.

**ABAC та його імплементація**

Науковий аналіз потенціалу інтеграції (ABAC) у сучасні системи кібербезпеки вказує на значний потенціал цього методу. ABAC може бути інтегрований у різноманітні системи захисту, використовуючи його гнучкість і динамічність для підвищення загального рівня безпеки. Вивчення ABAC з наукової точки зору висвітлює його переваги у контексті адаптивності та масштабованості. ABAC дозволяє створювати складні правила доступу, які можуть динамічно змінюватися в залежності від змінних атрибутів користувачів та ресурсів. Такий підхід може забезпечити вищий рівень безпеки в порівнянні з традиційними статичними методами контролю доступу. З огляду на сучасні виклики кібербезпеки, такі як внутрішні загрози, розширення доступу до хмарних сервісів та зростання загроз, пов'язаних з Інтернетом речей (IoT), ABAC пропонує підхід, що відповідає цим викликам. Його здатність до деталізації правил доступу на основі багатофакторного аналізу атрибутів робить його ідеальним для складних мережевих середовищ, де потрібен високий рівень індивідуалізації доступу. Крім того, з точки зору відповідності нормативним вимогам, таким як Загальний регламент захисту даних (GDPR) та інші регуляторні стандарти, ABAC може забезпечити більш точне та контрольоване управління доступом до даних, що є критично важливим для дотримання цих стандартів. Наукове дослідження потенціалу ABAC також підкреслює необхідність його інтеграції з іншими системами безпеки, такими як SIEM, IDS/IPS та системи управління ідентифікацією, щоб максимізувати ефективність захисту. Інтеграція ABAC з цими системами може створити більш зв'язну та взаємодоповнюючу структуру безпеки. У підсумку, науковий погляд на ABAC в контексті кібербезпеки підкреслює його потенціал як цінного інструменту, здатного значно покращити безпеку в різноманітних IT-

середовищах. Його універсальність і адаптивність роблять його привабливим вибором для інтеграції в сучасні системи захисту.

У контексті ABAC, можливість майнінгу прав являє собою важливу перспективу, яка дозволяє поглибити розуміння та оптимізувати системи контролю доступу. Майнінг прав на основі ABAC може включати аналіз чинних моделей доступу та поведінки користувачів для виявлення та встановлення оптимальних політик доступу. Цей процес може виявити неочевидні зв'язки та залежності між атрибутами та правами доступу, що допомагає у формуванні більш точних та безпечних правил доступу.

Але до цього часу, не існує широко відомих випадків, де метод майнінгу прав на основі ABAC був би конкретно використаний для реагування на кіберінциденти. Хоча ABAC є дуже потенційно потужним інструментом для покращення кібербезпеки, його практичне застосування в контексті кіберінцидентів залишається відносно недослідженим. Основна сила ABAC полягає в його гнучкості та здатності адаптуватися до складних та динамічних умов доступу. Використання різноманітних атрибутів для визначення прав доступу може значно збільшити безпеку, зокрема у складних або великих організаційних середовищах. Однак, використання ABAC як засобу реагування на кіберінциденти потребує подальшого вивчення та експериментування. Реальні випробування та імплементація ABAC в різних сценаріях кіберінцидентів можуть надати цінні дані щодо його ефективності та практичності. Також важливим є розуміння, як цей метод може бути інтегрований з іншими засобами кібербезпеки, щоб створити комплексний та ефективний захист від кіберзагроз. В цьому контексті, дослідження та пілотні проєкти, які використовують ABAC для управління доступом і реагування на кіберінциденти, можуть відіграти ключову роль у розробці нових стратегій кібербезпеки. Особливо важливо, щоб такі проєкти включали оцінку

ефективності, масштабованості та інтеграції з теперішніми системами кіберзахисту.

# Висновки до розділу 1

В даному розділі визначено поняття хмарних технологій, їх видів, переваги та недоліки використання. Проаналізовано динаміку розвитку та використання хмарних сервісів та атаки на хмарну інфраструктуру. Також було розглянуто можливості використання та розподіл хмарних сервісів

Розглянуто традиційні методи розподілу прав доступу, моделі захисту та виявлення порушень. Проаналізовано традиційні механізми авторизації та авторизації під час роботи з хмарним середовищем.

Виокремлено проблеми безпеки, що постають під час використання хмарних сервісів, як вони впливають на ринок та з якими викликами зіштовхнула Україна

Для розв'язання проблеми виявлення можливих надлишкових доступів у користувачів запропоновано розробити модель, що буде враховувати поточний стан прав користувачів до бажаного на основі політики безпеки організації на базі хмарного середовища та його інструментів

# 2 Модель майнінгу прав на основі ABAC

## 2.1 Модель доступу в хмарному середовищі (AWS) за допомогою ABAC

В AWS існують серверні та безсерверні ресурси, які будуть використовуватись в подальшому аналізі. Безсерверні сервіси - це набір хмарних послуг, які дозволяють створювати та запускати додатки без необхідності займатися управлінням серверами. Основні характеристики безсерверних сервісів включають:

1. Автоматичне масштабування: Сервіси автоматично масштабуються відповідно до потреби, зменшуючи або збільшуючи ресурси, необхідні для обробки запитів.
2. Оплата за використання: Оплата тільки за ті ресурси та час, які використовувались для обробки ваших запитів.
3. Управління інфраструктурою: AWS керує інфраструктурою, автоматично забезпечуючи безпеку, моніторинг та підтримку.

Деякі з популярних безсерверних сервісів в AWS включають AWS Lambda (для виконання коду в відповідь на події), Amazon S3 (для зберігання файлів), Amazon DynamoDB (для баз даних) та Amazon API Gateway (для створення та управління API).

У порівнянні з серверними сервісами, головні відмінності безсерверних сервісів полягають у:

- Управління серверами: В традиційних серверних сервісах користувачі повинні управляти серверами (фізичними або віртуальними), включаючи їх налаштування, масштабування та обслуговування. У безсерверному середовищі, AWS відповідає за всю інфраструктуру.
- Масштабування: У серверних сервісах масштабування часто є ручним процесом або потребує автоматизованих рішень для масштабування. У безсерверних сервісах масштабування відбувається автоматично і миттєво.

Таким чином, безсерверні сервіси пропонують гнучкіше і більш ефективне середовище для розробки та запуску додатків.

Перед побудовою власної моделі доступу необхідно розглянути класичну модель за допомогою діаграми класів та послідовностей. Основними класами хмарного середовища будуть:

- IAM Service (IAM сервіс) – основний клас сервісу в хмарному середовищу. Саме даний сервіс відповідає за реалізацію підсистем аутентифікації (**authenticate(Account, Resource)**) в системі. Містить в собі всі акаунти хмарного середовища accounts. Перевіряє кожну дію суб’єкта системи відносно відповідної політики безпеки об’єкта та забороняє неавторизовані дії. Дозволяє дії відносно акаунтів середовища (**delete_account(Account)**, **add_account(Account)**), політики безпеки хмари(**edit_policy(Policy)**, **add_policy(Policy)**, **remove_policy(Account, Policy)**) та можливість присвоювати політику безпеки до акаунту **add_policy_account(Account, Policy)**
- AWS Console (консоль доступу AWS) - об’єкт, який є інтерфейсом взаємодії кінцевого користувача з хмарним середовищем. Основним завданням консолі є авторизація користувача**(authorize(username,**

**password, has_mfa)**) та має на меті отримати від користувача його логін(**username**), пароль(**password**) та за встановленням мультифакторну автентифікацію(**has_mfa**)

- Policy (Політика безпеки) - це об’єкт, що характеризує набір принципів які визначаються та впроваджуються власником хмарного середовища. Містить в собі унікальний ідентифікатор політики безпеки(**policy_id**), набір правил(**rules**) та додаткові метатеги, які будуть виконувати роль атрибутів(**tags**)
- Rule (Правило) - це компонент системи безпеки в хмарному сервісі, який визначає конкретні умови доступу та управління для ресурсів системи. Містить в собі перелік дій(**action**), які політика дозволяє або забороняє(**effec**t) у відношення до ресурсу(**resource**)
- Account (Акаунт) - об’єкт “хмари”, що характеризує можливість доступу до інших об’єктів системи(користувацький, системний, сервісний), який виступає в ролі суб’єкта в системі хмарного середовища. Кожен акаунт має унікальний ідентифікатор**(account_id**), атрибути(**tags**) та можливість взаємодії з іншими об’єктами(**getAccess(Resource)**)
- Resource (Ресурс) - абстрактний клас об'єкта “хмари”. Містить унікальний ідентифікатор(**id**) та атрибути(**tags**) та є частиною усіх об’єктів середовища
- EC2 (Обчислювальний сервер) - об’єкт хмарного середовища, котрий надає обчислювальні потужності в хмарі, наприклад, використання віртуальних машин для обчислень. Імплементує Ресурс з власним ідентифікатором**(id**), збирає інформацію про надання доступу акаунтам(**logs**) та надсилає інформацію про спроби отримання доступу до централізованого місця зберігання аудиту**(push_logs(ResourceLog, CloudTrailId**)

- CloudTrail (журналювання подій та моніторинг) - об’єкт хмарного середовища який призначений для моніторингу та журналювання подій та імплементує Ресурс. Кожен екземпляр має унікальний ідентифікатор(**id**), журнал події та моніторингу(**logs**) та можливість збереження отриманої інформації**(save_logs(S3Id, logs))**
- S3 (об’єктне сховище) - об’єкт хмарного середовища, який надає інтерфейс збереження файлових об’єктів. Для коректного збереження екземпляр має унікальний ідентифікатор(**id**) та самі файлові об’єкти(**objects**). Створення кожного нового файлового об'єкта має на меті відсилання події про створення файлового екземпляра**(push_event(eventBridgeId, objectId**)
- EventBridge (шина подій) - об’єкт хмарного середовища, який імплементує безсерверну шину подій, в які знаходяться події, згенеровані іншими об’єктами системи. Має унікальний ідентифікатор**(id**), отримані події**(events**) та можливість запустити інший ресурс хмарного середовища, на основі отриманої події(**activate_lambda(LambdaId, event)**)
- Lambda (безсерверна функція) - об’єкт хмарного середовища, який виконує зміну будь-якого об'єкта “хмари” за допомогою API. Має унікальний ідентифікатор(**id**) та єдину функції виклику на основі отриманої події(**invoke_function(event)**)

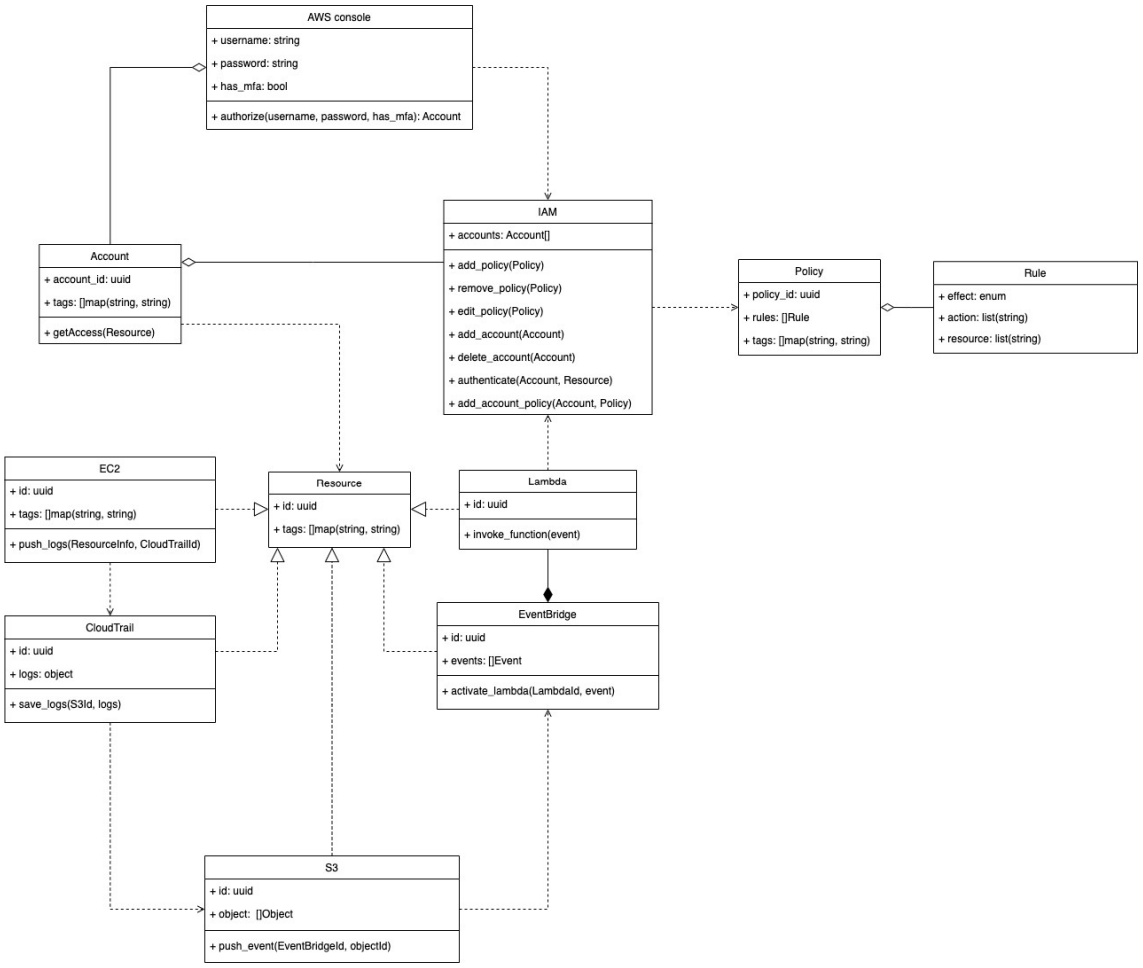


Рисунок 2.1 - Діаграма класів моделі безпеки доступу в AWS за допомогою ABAC

У наступній діаграмі буде розглянуто послідовність взаємодії класів на доступ до об’єктів та модифікації даних користувача в системі:

В моделі безпеки доступу показано взаємодію між користувачем, різними компонентами AWS і процес авторизації, логування та зміна атрибутів в залежності від дії Account

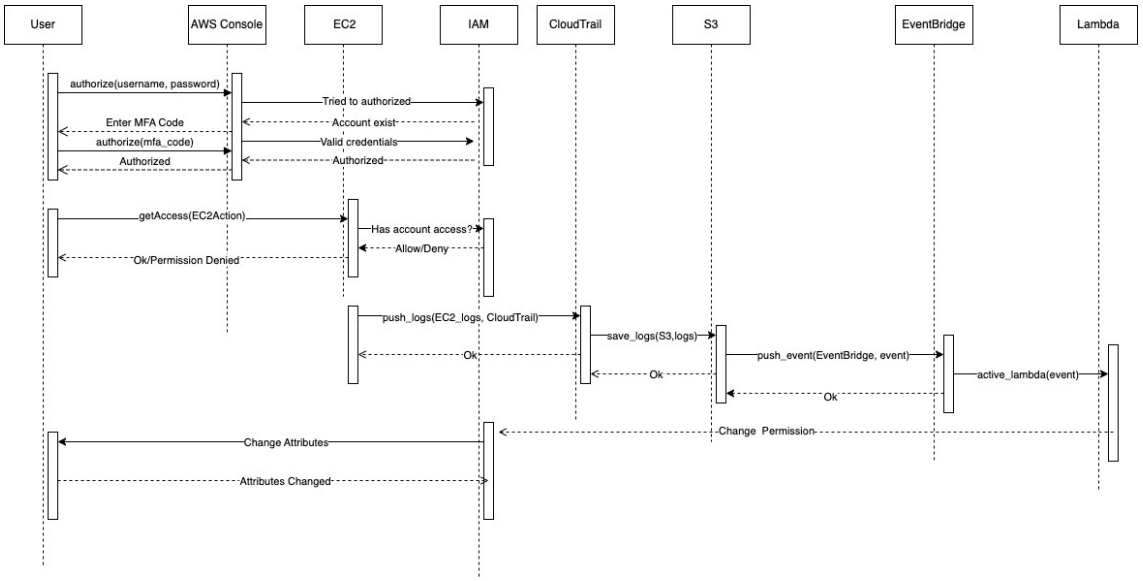


Рисунок 2.2 - Діаграма послідовності для моделі безпеки на основі ABAC

## 2.2 Модель майнінгу прав на основі ABAC

Attribute-Based Access Control (ABAC) є підходом до системи управління доступом, який використовує атрибути, а не ролі для прийняття рішень щодо доступу. Атрибути можуть бути пов'язані з користувачем, ресурсом, дією або середовищем.

Attribute-based access rule mining — це процес аналізу шаблонів доступу для автоматичного генерування правил доступу, які враховують різні атрибути. Метою цього процесу є зменшення потреби в ручному визначенні політик доступу, яке може бути часомістким і схильним до помилок, особливо в великих і складних системах

До діаграми класів моделі безпеки на основі ABAC додамо наступні класи:

- Mined Rule(Генероване правило) - надбудова над класом Rule, яка дозволяє додавати перевірки приналежності бажаних параметрів до суб'єкта системи(**conditions**)
- Rule Miner Agent (Агент Механізму генерації правил) - абстрактний клас механізму генерації правил. Містить унікальний ідентифікатор(id), інформацію про інцидент(incident_info) та віртуальний метод (**run_mining_process**()) який буде використовуватись у наступних екземплярах механізму генерації правил
- Rule Miner (Механізм генерації правил) - об’єкт , який дозволяє автоматично визначати загрози від дій акаунтів та генерувати правила для запобігання схожих дій. Під час роботи з інцидентами в екземпляра класу є можливі акаунти, які є підозрілими(**accounts**), всі їх атрибути(**attribute_set**), можливі шляхи розв'язання проблеми(**candidate_rules**) та місце зберігання інциденту(**s3_bucket_name**). Для роботи з інцидентами екземпляра потрібно систематизувати події зі сховища(**parse_logs()**), згенерувати забороняюче доступ правило(**generate_deny_rule()**), перевірити, чи у всіх акаунтів однакові права на дане правило(**check_access_consistency(accounts)**), у випадку конфлікту додати додатковий керуючий атрибут для акаунтів, який вирішить конфлікт(**add_deny_rule_attribute**(**account**, attribute)), видалити суперечливий атрибут з алгоритму, додати додатковий атрибут в **attribute_set** (**add_attributes_to_set**(**attributes**)) та згенерувати фінальне правило доступу та модифікувати акаунти на основі правила доступу(**run_mining_process**())

  Та було модифіковано наявний об’єкт середовища:

- Lambda - об’єкт також зберігає інформацію про інцидент безпеки(**incident_info**) та надсилає дану інформацію до агента майнінгу прав(**send_incident_info(incident_info)**)

Отже, тепер система виглядає наступним чином: під час зміни атрибутів за допомогою lambda виконується додатковий крок для пошуку схожих ідентифікаційних портретів акаунтів, та встановлюється заборона на проведення несанкціонованих дій для всіх учасників

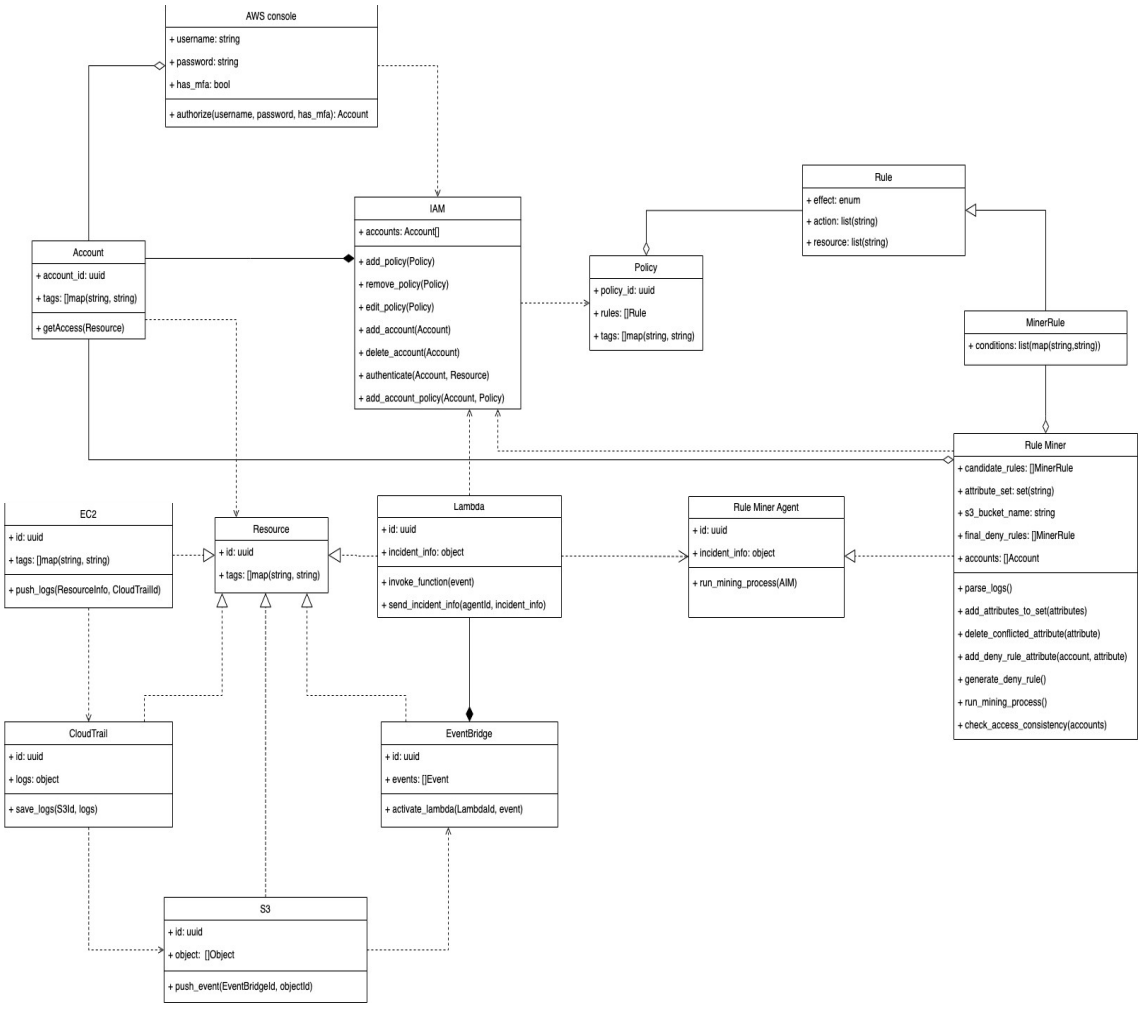


Рисунок 2.3 - Діаграма класів моделі майнінгу прав в хмарному середовищі

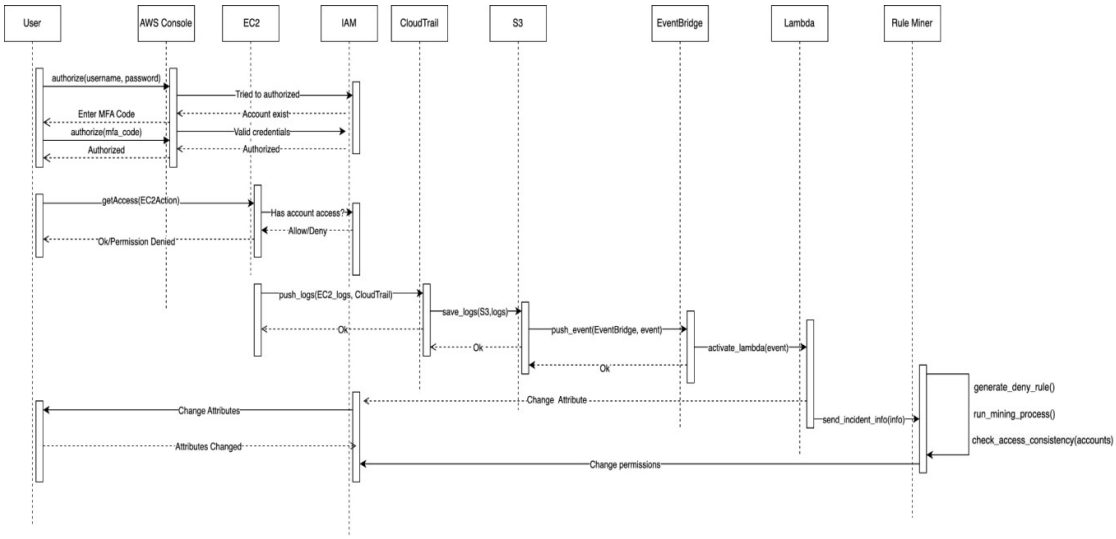


Рисунок 2.4 - Діаграма послідовності моделі майнінгу прав в хмарному середовищі

## 2.3 Запропонована модель майнінгу прав

Mining rules, або майнінг правил[16], у контексті управління доступом, це процес автоматичного виявлення та створення політик доступу на основі аналізу поведінки користувачів і спроб доступу до ресурсів. Цей процес базується на декількох ключових підставах:

1. **Історичні Дані Аудиту (Логи)**: Майнінг правил використовує логи аудиту для аналізу попередніх спроб доступу до ресурсів. Ці логи можуть включати інформацію про успішні та невдалі спроби доступу, включно з деталями, як-от роль користувача, час спроби, місцеперебування, та інші відповідні атрибути.
2. **Атрибути Користувачів і Ресурсів**: Атрибути, які можуть включати ролі, географічні регіони, персональна інформація користувачів, теги ресурсів, часові мітки тощо, використовуються для визначення поведінкових шаблонів і створення правил.

3. **Моделі Поведінки**: За допомогою аналізу логів можна виявити звичні моделі поведінки користувачів або груп і визначити, які дії зазвичай дозволені або заборонені.
4. **Правила доступу**: На основі аналізу створюються правила, які визначають дозволені або заборонені дії для користувачів з певними атрибутами. Ці правила можуть бути загальними або дуже специфічними, залежно від виявлених шаблонів.
5. **Політика Безпеки Організації**: Всі майнінг правил повинні відповідати загальній політиці безпеки організації, враховуючи вимоги конфіденційності, регуляторні стандарти та бізнес-вимоги.
6. **Автоматизація**: Майнінг правил часто включає використання автоматизованих інструментів та алгоритмів для ефективного аналізу великих обсягів даних, виявлення шаблонів та генерації правил без безпосередньої людської участі.
7. **Валідація та Тестування**: Перед застосуванням нових правил вони проходять процес валідації та тестування, щоб переконатися у їхній ефективності та відсутності непередбачених конфліктів або вразливостей.

Розглянемо детальніше, як саме буде забезпечуватись майнінг прав за допомогою алгоритму Рисунок 2.5

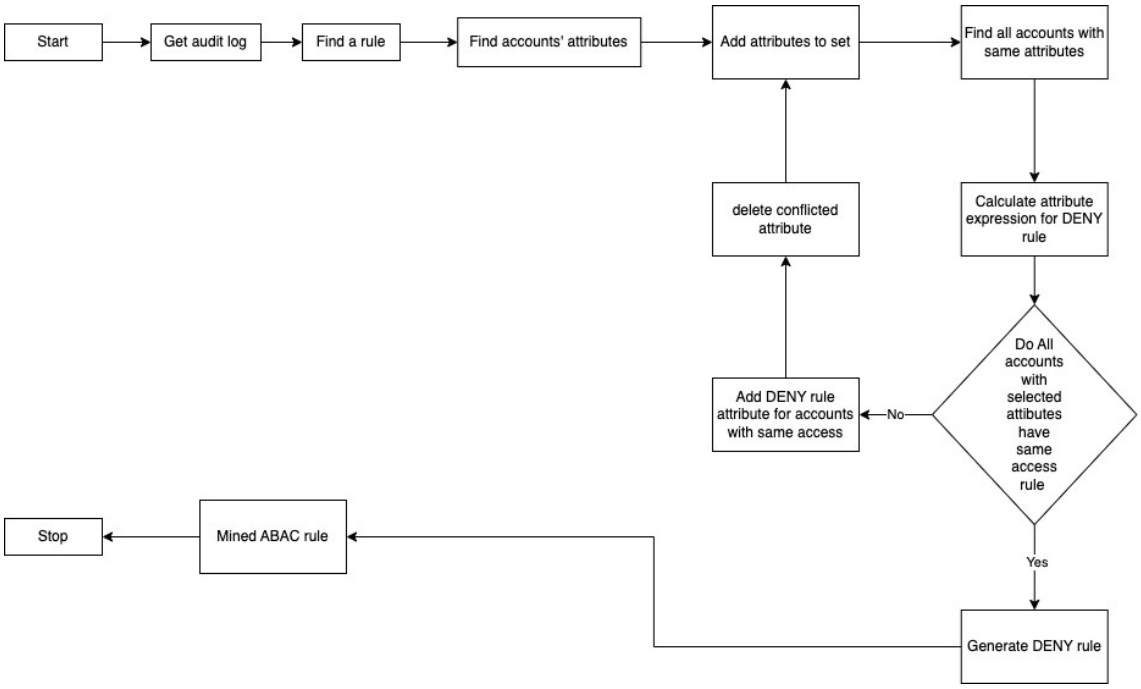


Рисунок 2.5 - Алгоритм майнінгу прав

Процес починається зі збору аудит-логів та закінчується створенням виведеного правила доступу. Детальний опис кожного кроку алгоритму:

1. **Get audit log**: Алгоритм починається з отримання аудит-логів, які містять інформацію про дії користувачів із системою.
2. **Find a rule**: Проводиться пошук конкретних правил або шаблонів у цих логах, що можуть вказувати на необхідність створення нових правил доступу.
3. **Find accounts' attributes**: Визначаються атрибути акаунта, який брав участь у подіях, записаних у логах.
4. **Add attributes to set**: Виявлені атрибути додаються до набору, який буде використовуватися для формування нового правила доступу.
5. **Find all accounts with same attributes**: Пошук всіх акаунтів, які мають такі самі атрибути. Це необхідно для забезпечення консистентності правил.

6. **Calculate attribute expression for DENY rule**: Розрахунок виразу атрибутів для правила `DENY`, яке буде використане для обмеження доступу.
7. **Do All accounts with selected attributes have same access rule**: Перевірка, чи всі акаунти з вибраними атрибутами підпадають під однакове правило доступу.
8. **Add DENY rule attribute for accounts with same access**: Якщо виявлено, що не всі акаунти мають однакове правило доступу, алгоритм додає атрибути для правила `DENY` для цих акаунтів, щоб уніфікувати доступ.
9. **Delete conflicted attribute**: Якщо під час процесу виявляється конфлікт атрибутів (наприклад, акаунт з роллю `admin` не має підпадати під правило `DENY`), конфліктні атрибути видаляються з набору.
10. **Generate DENY rule:** Якщо всі акаунти з вибраними атрибутами мають однакове правило доступу, генерується остаточне правило `DENY`.
11. **Mined ABAC rule**: В результаті ми отримуємо правило, яке було "видобуто" з аналізу логів і готове до застосування в системі управління доступом.

Розглянемо детальніше створення правил або шаблону:

- **Визначення Цілей**: Система починає з визначення цільових дій, які повинні бути контрольовані правилами доступу. Це можуть бути операції, такі як читання, запис, видалення або модифікація ресурсів.

- **Ідентифікація Атрибутів**: Визначаються атрибути, які є важливими для встановлення політик доступу. Атрибути можуть стосуватися користувачів, ресурсів, дій та контексту.
- **Логіка Правил**: Встановлюється логіка правил, яка визначає, як атрибути взаємодіють для надання або заборони доступу. Ця логіка може бути простою (якщо атрибут A є істинним, дозволити дію) або складною (якщо атрибути A і B є істинними, але C є неправдивим, заборонити дію).
- **Формулювання Умов**: На основі логіки правил формулюються конкретні умови, які будуть вбудовані в правила доступу. Умови можуть включати логічні вирази та порівняння атрибутів.
- **Створення Правила**: Правила формулюються у відповідному форматі, що підтримується системою управління доступом. Вони повинні бути чітко структуровані, щоб система могла їх інтерпретувати та застосовувати.

Під час застосування процесу майнінгу правил можуть виникнути різні проблеми, наприклад:

1. **Комплексність Даних**: Логи можуть містити величезну кількість даних, які можуть бути складними для аналізу. Розуміння та інтерпретація цих даних вимагає спеціалізованого знання та інструментів.
2. **Шум у Даних**: Логи можуть містити "шум" - неістотні або випадкові дії, які не повинні впливати на правила доступу.
3. **Виявлення Патернів**: Автоматичне виявлення шаблонів та аномалій у поведінці користувачів може бути складним.

Неправильне виявлення може призвести до невірного створення правил.

4. **Конфлікти Правил**: Можуть виникнути конфлікти між ново сформованими правилами та наявними політиками, що може призвести до непередбачуваного поведінки доступу.
5. **Дотримання Політики**: Нові правила повинні відповідати корпоративним політикам безпеки та стандартам дотримання. Встановлення правил, які не відповідають цим вимогам, може призвести до юридичних або регуляторних проблем.
6. **Застарілі Правила**: Потреба в постійному оновленні правил для забезпечення їх актуальності з огляду на зміну бізнес-процесів та технологічних умов.
7. **Продуктивність Системи**: Велика кількість деталізованих правил може вплинути на продуктивність системи, сповільнюючи процеси перевірки та виконання доступу.
8. **Управління Змінами**: Необхідність управління змінами в політиках та тренінгів для користувачів, щоб забезпечити розуміння та дотримання нових правил.

Для розв'язання цих проблем може знадобитися поглиблений аналіз, включаючи використання машинного навчання для фільтрації шуму та ідентифікації вірних патернів, регулярний перегляд політик для забезпечення їхньої актуальності, а також ретельне планування та тестування перед впровадженням будь-яких нових правил.

# Висновки до розділу 2

Для розв'язання проблеми безпеки, розглянутих в розділі 1, буде побудована модель майнінгу прав доступу, що буде генерувати та надавати права на основі атрибутів користувачів та поточного стану безпеки системи

Розглянуто базову об’єктну модель безпеки на основі атрибутів та визначено місце інтеграції та взаємодії компонентів, з якими буде взаємодіяти запропонована модель.

Виокремлено дві додаткові елементи системи: ЛФА та ЛФМ, які будуть виступати генерацією та присвоєнням нових прав доступу відповідно. Побудовано відповідні діаграми класів та послідовностей, що описують статичну та динамічні взаємодії компонентів між собою. За допомогою діаграми послідовностей розглянуто взаємодію нових об’єктів системи та хмарного середовища

Запропоновано детальну реалізацію побудови інфраструктури для моделі розділення доступу ABAC, описано алгоритм майнінгу прав та впровадження його в інфраструктуру, що існує на основі опису діаграм класів та послідовностей

# 3 Тестування програмної реалізації агентної моделі та порівняльний аналіз

## 3.1 Огляд програми та структури взаємодії сервісів

Реалізація моделі виконана мовою програмування Python 3.11 та відповідає діаграмі класів, представленій в розділі 2. Реалізація зосереджена на управлінні політиками доступу користувачів AWS IAM (Identity and Access Management) на основі їх атрибутів та зміні їх за допомогою майнінгу прав.

Програма складається з двох частин: Lambda-функція агент (ЛФА) та Lambda-функція майнер (ЛФМ).

ЛФМ використовується для обробки S3 подій. Вона читає дані з об'єкта S3, який є файлом журналу у форматі gzip та аналізує логи з цього файлу, перетворюючи їх у JSON формат для подальшої роботи. Далі ЛФМ програма використовує IAM API для отримання інформації про користувачів і їх політики доступу. Основні дії включають: 1) Перевірка та виведення тегів користувачів 2) Аналіз політик доступу користувачів. 3)Виявлення та вирішення конфліктів у політиках. Наступним кроком є обробка подій AWS сервісу: програма шукає конкретні події, для аналізу та майнінгу нових правил. У разі виявлення такої події, вона виконує наступні кроки:

- Створення нових IAM політик, заснованих на тегах користувачів.
- Аналіз політик, що існують у користувачів.
- Виявлення та вирішення конфліктів між політиками.
- Оновлення політик користувачів з урахуванням нових правил.

Управління конфліктами політик: Одна з ключових функцій - знаходження та вирішення конфліктів між 'Allow' та 'Deny' політиками, застосовуючи специфічні умови засновані на тегах користувачів.

На кожному етапі роботи програми відбувається логування для відстеження дій та обробка помилок

ЛФА займається зміною поточних прав доступу на згенеровані від ЛФМ

**Налаштування інфраструктури для взаємодії**

1. Створення користувачів в системі адміністратором з належними атрибутами. В системі є наступні пари ключів та значень атрибутів: role - qa, pm, developer project - feel, davinci, loyal
2. Налаштувати базові права на дії стосовно сервісів AWS
3. Створити сервер для маніпуляції його станом


Рисунок 3.1 - Успішне створення віртуального сервера

4. Створити журнал подій про усі маніпуляції з AWS акаунтом

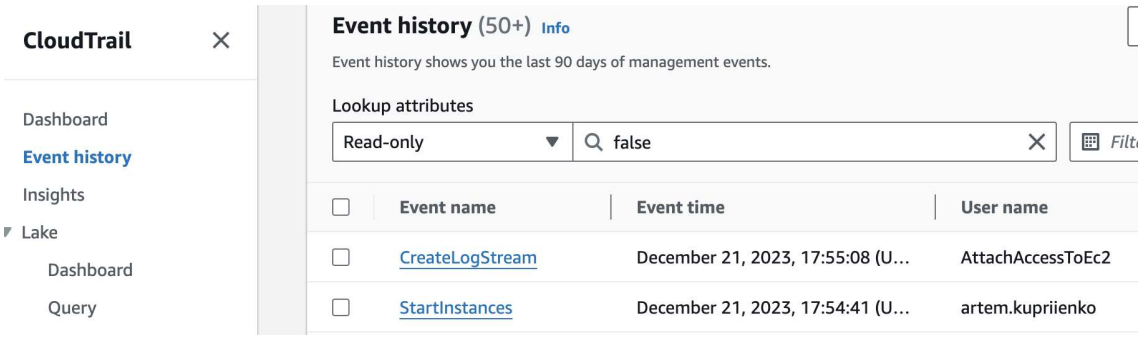


Рисунок 3.2 - Збереження створення події в CloudTrail

5. Створення файлового сховища AWS S3 для зберігання об’єктів даних від CloudTrail

6. Налаштування збереження аудит файлів та перевірка їх доступності

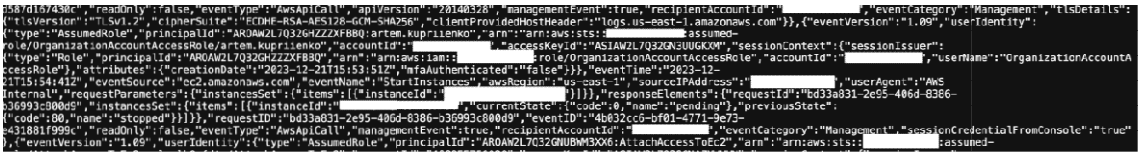

Рисунок 3.3 - Перевірка аудит логу на файловому сховищу

7. Створення шини подій, яка буде реагувати на створення нового об’єкту в S3

8. Створення ЛФМ та ЛФА

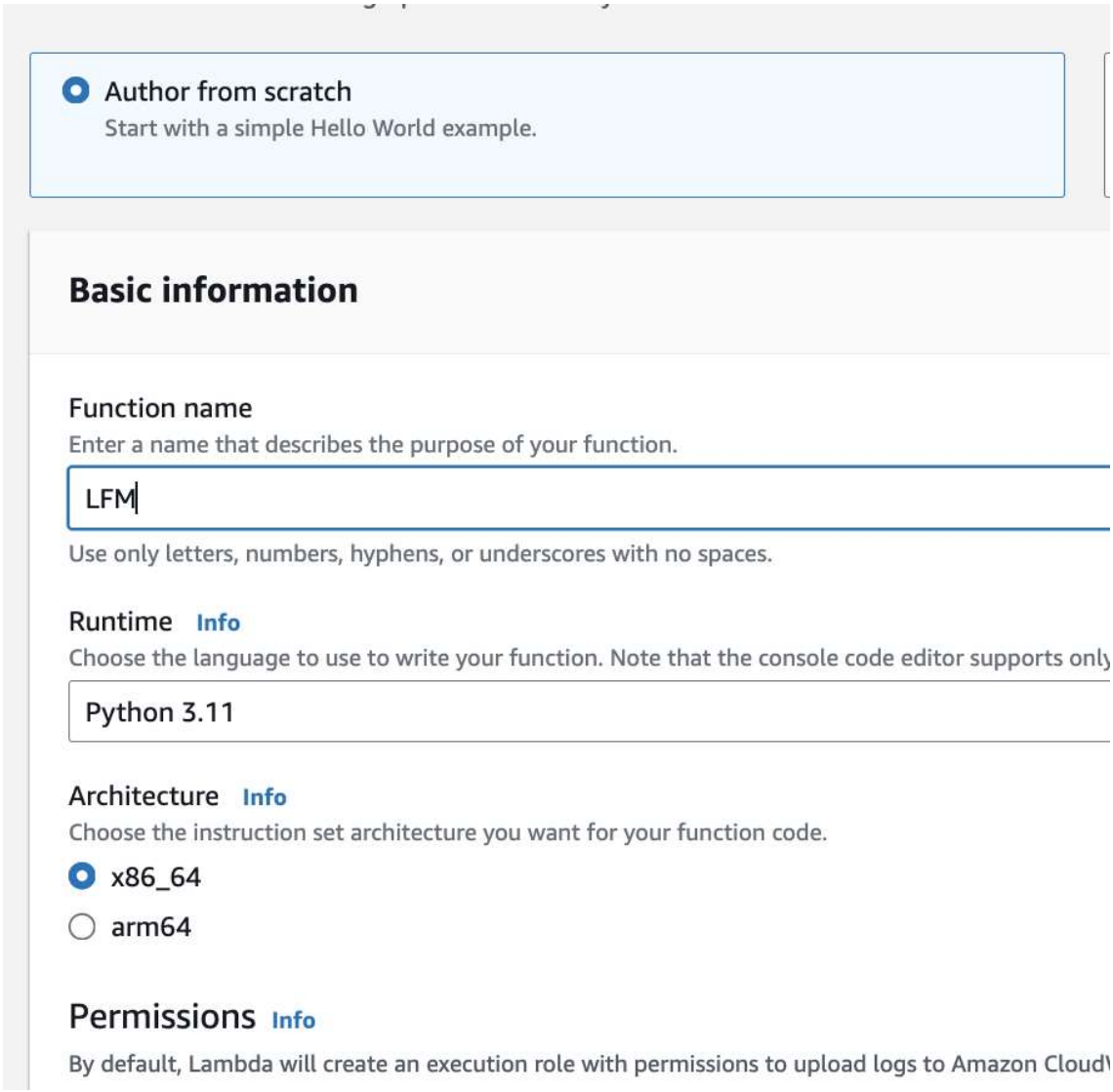


Рис 3.4 - Створення ЛФМ на основі Python 3.11

9. Додати до ЛФМ реагування на створення нового об’єкту в S3

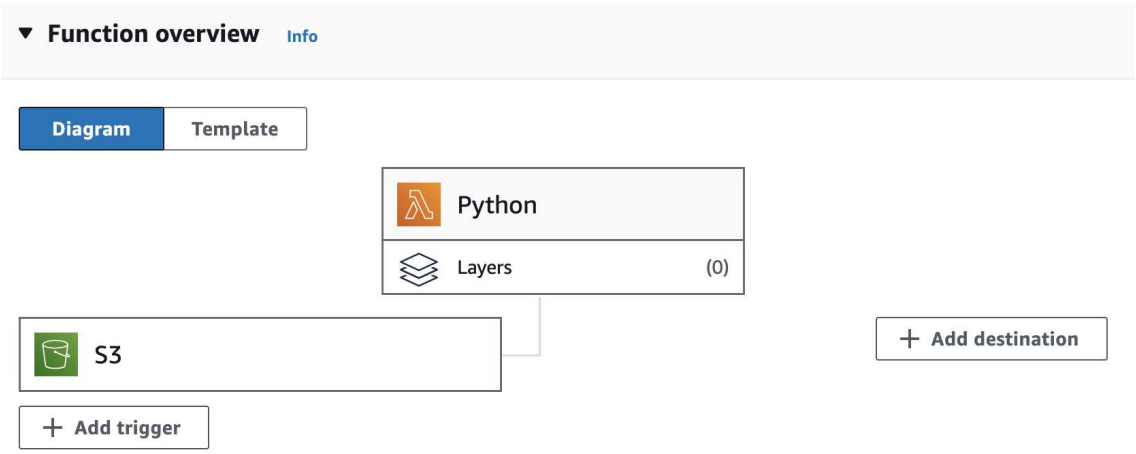


Рисунок 3.5 - ЛФМ з тригером на новий об’єкт

10. Дати ЛФА дозволи на маніпуляцію з користувацькими атрибутами та їх доступами

Програма має декілька вихідних кодових файлів:

- LFA.py - опис інтерфейсу для генерації правил доступу
- LFM.py - реалізація майнінгу нових правил доступу
- LFM_config.json - JSON документ опису політики безпеки в AWS, що буде надана ЛФМ
- Users1-5_config.json - JSON документ опису політики безпеки в AWS, що буде надана першій п’ятірці користувачів
- Users6-10_config.json - JSON документ опису політики безпеки в AWS, що буде надана другій п’ятірці користувачів
- DenyStopInstancesForTaggedUsers.json - документ опису політики безпеки в AWS, яка згенерована під час майнінгу

## 3.2 Приклад роботи майнінгу прав

Після успішної авторизації користувача User5 з атрибутами role=pm та project=feel, даний користувач успішно зупинив створений вище сервер

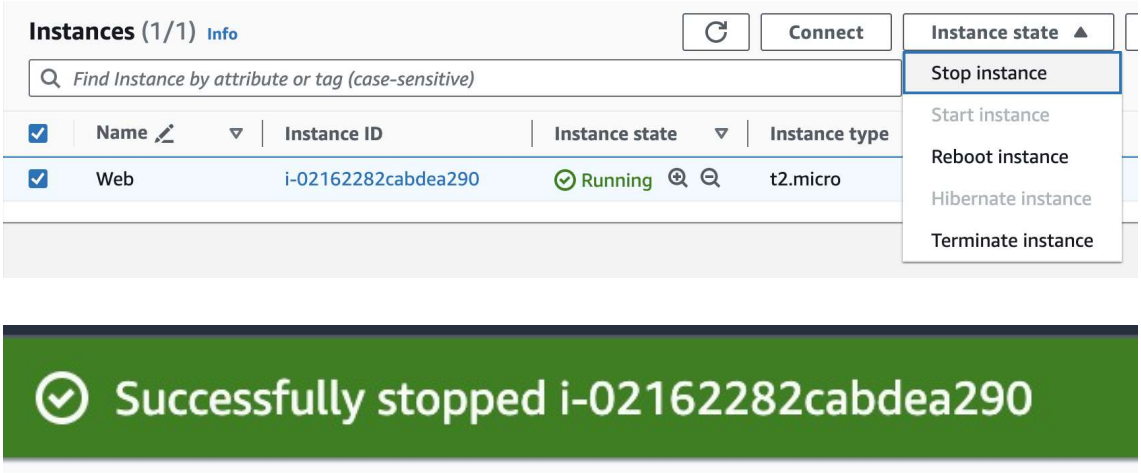


Рисунок 3.6 - Зупинка сервера

Після чого розпочався процес аналізу даної події. Зазвичай, з деяким проміжком часу, сервер надсилає аудит по своєму стану. Адміністратором було встановлено наступні критерії майнінгу прав:

- Користувач з атрибутом role=pm більше не має права робити вище зроблену дію
- Користувач з атрибутом project=feel більше не має права робити вище зроблену дію

Приклад роботи ЛФМ:

Зібрано інформацію про користувача

```
================================================
Користувач User5 має наступні атрибути: 1) role - pm, 2) project - feel
```

Рисунок 3.7 - Атрибути користувача, який зупинив сервер

Створено політику безпеки, яка заборонить дію користувача

```
Створено нову політику DenyStopInstancesForTaggedUsers на заборону дії ec2:StopInstances
```

Рисунок 3.8 - Створена політика

Знайдено усіх користувачів, які мають схожі атрибути

```
3 користувачем User3 однакові наступні атрибути: role - pm
3 користувачем User4 однакові наступні атрибути: project - feel
3 користувачем User8 однакові наступні атрибути: project - feel
```

Рисунок 3.9 - Знайдені користувачі зі спільними атрибутами

Знайдено та проаналізовано поточку політику доступу знайдених користувачів

```
Користувач User3 має наступні права:
- ec2:StartInstances: ['Allow']
- ec2:DescribeInstances: ['Allow']
Користувач User4 має наступні права:
- ec2:StartInstances: ['Allow']
- ec2:DescribeInstances: ['Allow']
Користувач User5 має наступні права:
- ec2:StopInstances: ['Allow']
- ec2:DescribeInstances: ['Allow']
Користувач User8 має наступні права:
- ec2:StopInstances: ['Allow']
- ec2:DescribeInstances: ['Allow']
```

Рисунок 3.10 - Наявні дозволи користувачів

З аналізу поточних прав доступу, можна побачити, що у користувачів User5 та User8 є дозвіл на обрану дію для заборони надалі, про що повідомляє програма. Також аналізуючи поточні атрибути програма додала до користувача User5 додатковий атрибут, що буде контролювати подальші дії по зміні прав доступу

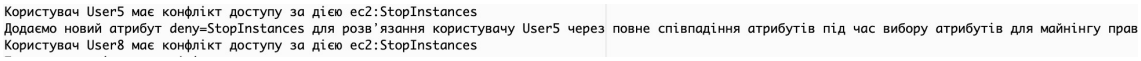


Рисунок 3.11 - Конфлікт прав доступу

Наступним кроком є вирішення конфлікту прав доступу на основі доданих нових атрибутів.

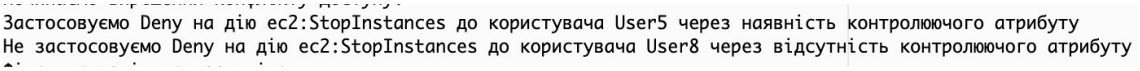


Рисунок 3.12 - Вирішення конфлікту доступу

Генерація остаточних правил доступу після вирішення конфлікту та відправлення даних до ЛФА

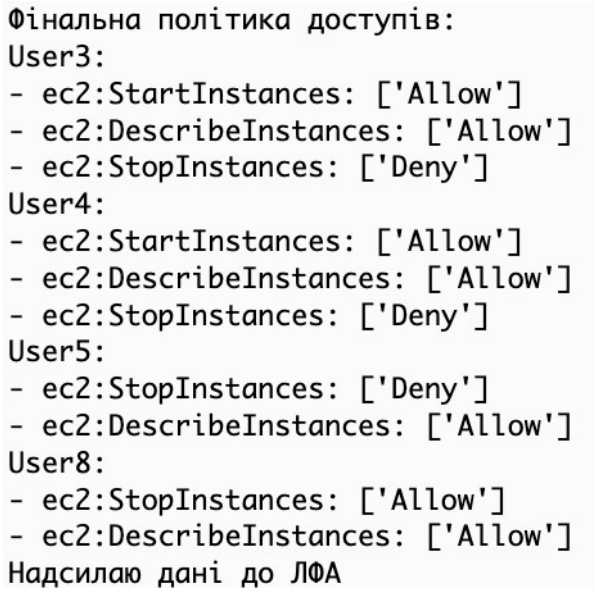


Рисунок 3.13 - Повторна генерація правил доступу

Після чого можемо побачити результат виконання ЛФА та перевірити права доступу користувачів

```
Нові політики успішно присвоєні
```

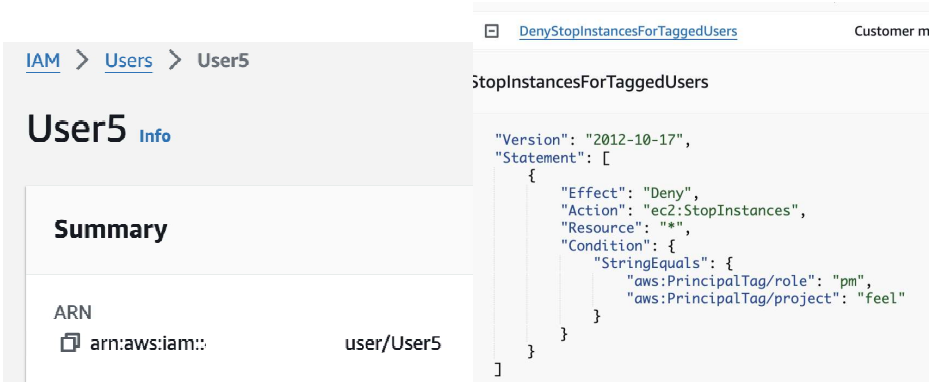


Рисунок 3.14 - Результат виконання ЛФА

## 3.3 Перевірка доступу

Для перевірки доступу користувачів в середовищі AWS можна використати їхній симулятор політик[12]. В секції Global Setting можна підставити атрибути користувача для отримання результатів

Зробимо перевірку політик для 3-х користувачів:

- User3 та User4 не брали участі під час перегенерації прав, тому оберемо будь-якого з ним, наприклад, User3
- User5 мав конфлікт доступу та має отримати заборону на виконання
- User8 мав конфлікт доступу та має отримати дозвіл на виконання

Отримання доступу на дію ec2:StopInstances для користувача з атрибутами role=pm та project=loyal є неуспішно. Майнінг прав спрацював за встановленим правилом

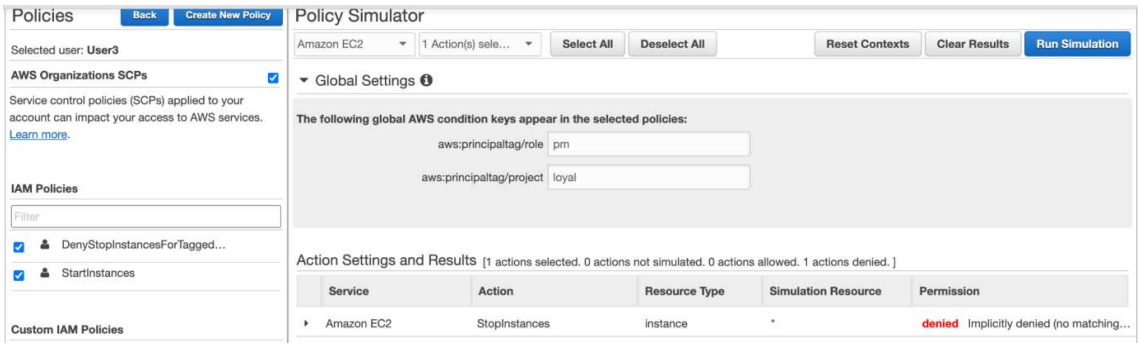


Рисунок 3.15 - Перевірка доступу для User3

Аналогічно для User5

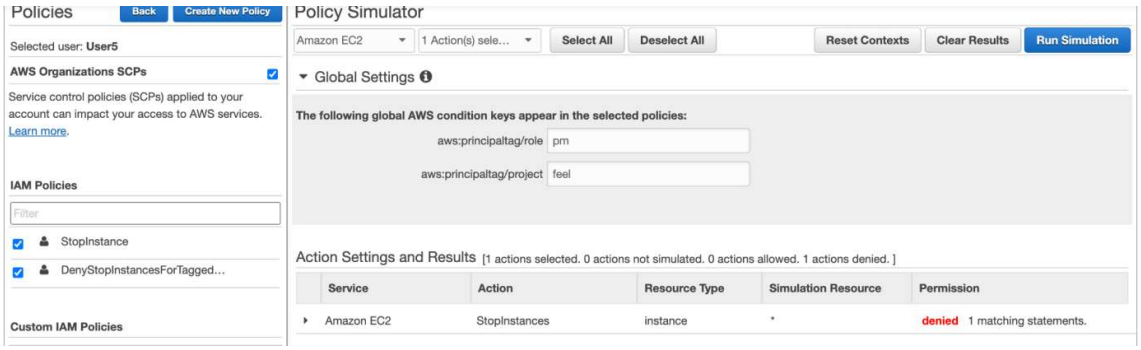


Рисунок 3.16 - Перевірка доступу для User5

Користувач User8 отримав дану політику доступу, але можливість виконання дії у нього залишилась, як і було вказано під час виконання майнінгу.

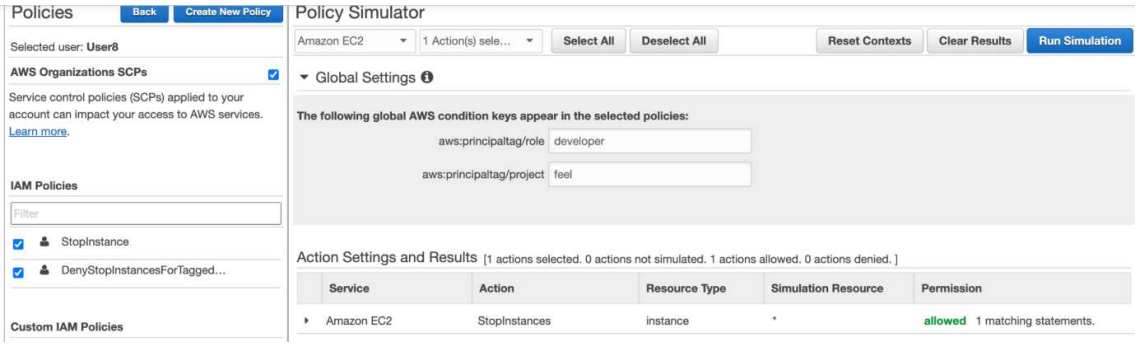


Рисунок 3.17 - Перевірка доступу для User8

## 3.4 Результати

Алгоритм було протестовано на наступній кількості користувачів: 10, 100 та 500. Для отримання даних про використання ОЗП, часу виконання будуть використовуватись вбудовані виміри використання в AWS Lambda. Для виміру витрачених коштів буде використовуватись формула вирахунку з офіційної документації Lambda[13]


Рисунок 3.18 - Діаграма кількості часу виконання алгоритму

З графіку видно, що з ростом кількості користувачів кожна наступна сотня користувачів потребує на 5 секунд більше часу, що є в середньому буде 13%. Для зменшення часу можна використовувати алгоритми швидкого пошуку або паралельне виконання алгоритму.

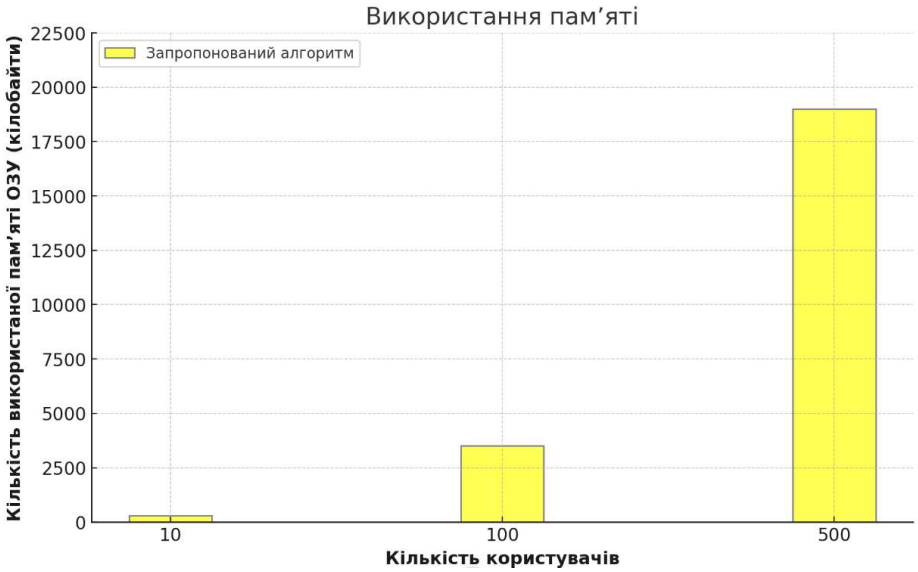


Рисунок 3.19 - Діаграма використаної пам’яті ОЗП алгоритму

Аналізуючи діаграму використаної пам’яті ОЗП можна побачити, що кожна сотня користувачів в середньому додає 8% використання ОЗП. Приріст є меншим порівняно з часом виконання. Ріст використання ОЗП обумовлений збереженням усіх пошуком атрибутів користувачів, збереженням їх та призначенням нових політик користувачам.

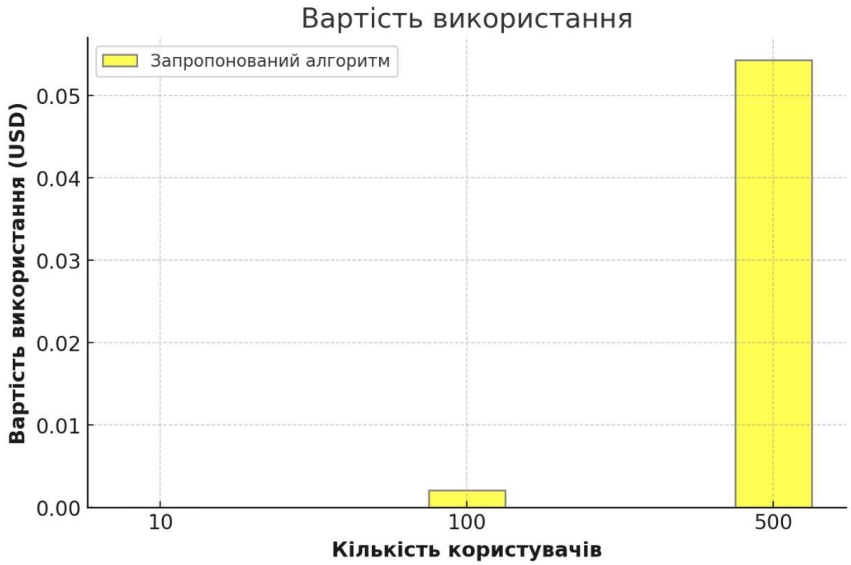


Рисунок 3.20 - Діаграма вартості використання алгоритму

З діаграми видно, що використання коштів виростає з більшим навантаженням на виконання алгоритму. На 10 користувачах витрата коштів має 0.33*10^-3 центів, що є у 1800 разів менше, ніж при аналізі в 500 користувачів. З діаграми можна сказати, що вигідніше та продуктивніше запускати алгоритм різними Lambda функціями з меншим навантаженням.

Порівняти запропоновану модель доступу з традиційними системами розмежування доступу, а саме: мандатною, дискреційною та рольовою - майже неможливо. Хоча в основі лежить атрибутна система розмежування доступу, але майнінг прав на основі ABAC - це алгоритм майнінгу правил доступу, а не система розмежування, тому порівняння може відбуватись зі схожими алгоритмами.

## 3.5 Порівняльний аналіз зі схожими алгоритмами

**Алгоритм Mining Positive and Negative Attribute-Based Access Control Policy Rules[14]**

Схожим аналогом до програмної реалізації майнінгу прав є алгоритм, який презентовано в даній публікації. Даний алгоритм націлений на аналіз великої кількості аудит-логів, розв'язанню проблеми надлишкового доступу користувачів, створенню Allow та Deny правил доступу, розширенню поточних прав доступу.

Як і в запропонованій моделі, даний алгоритм враховує поточні доступи користувачів, аналізує можливі заборони та робить перевірку вірності призначених прав доступу.

**Критерії для порівняльного аналізу**

Основними критеріями при порівняльному аналізі алгоритмічних рішень для майнінгу прав доступу є швидкість виконання алгоритму(O[15]), аналіз великої кількості даних, можливість працювати в реальному часі та вирішенню конфліктів прав доступу.

**Порівняльний аналіз на основі вказаних критеріїв**

У наступній таблиці наведено порівняння критеріїв запропонованої моделі з (MPNABAC). Хоча наразі порівнювальний алгоритм не має взаємодії зі хмарними технологіями, але принципи побудови алгоритму можливо імплементувати в хмарні середовища. Швидкість роботи окремих частин буде враховуватись у найгірших ситуаціях при пошуку. Дані про час виконання окремих частин алгоритму отримані від авторів алгоритму

Таблиця 3.1 - Порівняння характеристик запропонованого алгоритму та MPNABAC

| **Алгоритм / Характеристика** | MPNABAC | Запропонований |
|---|---|---|

| | | |
|---|---|---|
| Швидкість пошуку сервісів в аудит-логів | O(n) | O(1) |
| Швидкість пошуку усіх пар можливих атрибутів | $O(n^2)$ | O(1) |
| Визначення найкращих пар атрибутів для майнінгу | $O(n^2)$ | - |
| Швидкість визначення конфліктів для Deny, та користувачів, які їх мають | $O(n^2)$ | $O(n^2)$ |
| Генерація Deny правила для користувачів | $O(n^2)$ | O(n) |
| Вирішення конфліктів доступу | Так | Так |
| Робота в реальному часі | Ні | Так |
| Підтримуване хмарне середовище | - | AWS |

За результатом порівняння даних алгоритмів можна сказати, що запропонований алгоритм гарно працює для конкретних дій та для контрактних атрибутів, в той час, як MPNABAC - більше на все одночасно

# Висновки до розділу 3

У розділі наведено опис реалізованої програми моделі майнінгу прав на основі атрибутів користувачів, запропонованої в розділі 2. Детально описана побудова та взаємодія суб’єктів хмарного середовища AWS та спосіб її використання.

Розглянуто найближчий схожий алгоритмічний підхід до майнінгу прав. Виокремлено основні критерії порівняння для порівняння алгоритмів. Побудовано таблицю порівняння даних алгоритмів на швидкодію та можливість експлуатації. Визначено, що запропонований алгоритм націлений на роботу з конкретним запитом на дію та атрибути користувачів, в той час, як MPNABAC охоплює весь спектр дій та атрибутів

Проведено аналіз ефективності використання ресурсів хмарного середовища за використанням ОЗП, процесорного часу, загальним часом виконання та грошових витрат. Встановлено, що ефективнішим використанням запропонованої моделі буде розділення кількості користувачів на окреме виконання задля продуктивнішого та менш затратного виконання

# Висновки

У даній роботі було розглянуто проблематику класичної моделі безпеки, основою якої є ABAC в ХС та запропоновано модель майнінгу прав, що зробить можливим аналізувати та покращувати рівень доступів у користувачів. У ході роботи було:

1. Проаналізовано дані з відкритих джерел про атаки на хмарні середовища та IT інфраструктуру України. Виокремлено найбільш поширені серед зловмисників вектори атак та найменш захищені компоненти системи. Побудовано діаграми класів та послідовностей для класичної моделі безпеки на основі атрибутів користувачів у ХС. Визначено основні компоненти системи та проаналізовано їх взаємодію на прикладі відповідних діаграм.
2. Побудовано діаграми класів та послідовностей для моделі майнінгу прав на основі ABAC в ХС, що має бути надбудовою поверх класичної імплементації ABAC в ХС. Визначено ролі агенту та майнера у системі, детально описано взаємодію об’єктів на прикладі діаграм послідовностей. Виокремлено підсистеми безпеки та задано детальний алгоритм їх роботи у відповідних блок-схемах. Новостворена інфраструктура відповідатиме усім актуальним вимогам до моделей безпеки ХС, розглянутих у розділі 1 та буде цілком задовольняти базові критерії кібербезпеки: цілісність, доступність та конфіденційність.

3. Розроблено прикладний алгоритм, що реалізовує запропоновану модель майнінгу прав на основі ABAC на практиці. Оглянуто етап налаштування інфраструктуру на Amazon Web Services та алгоритму, наведено приклади роботи. Здійснено порівняльний аналіз, з алгоритмом що існує на схожому підґрунті – MPNABAC. За результатами аналізу встановлено, що запропонований алгоритм кращий у використанні на конкретні цілі під конкретні користувацькі атрибути та проведено порівняння часу роботи, як запропонованого алгоритму, так і MPNABAC

# Перелік джерел посилань


1. FSR [Електронний ресурс] - Режим доступу: https://www.fsf.org
2. Using role-based access control [Електронний ресурс] - Режим доступу: https://docs.aws.amazon.com/cognito/latest/developerguide/role-based-access-control.html
3. Role-based access control [Електронний ресурс] - Режим доступу: https://cloud.google.com/data-fusion/docs/concepts/rbac
4. What is Azure RBAC? [Елекронний ресурс] - Режим доступу: https://learn.microsoft.com/uk-ua/azure/role-based-access-control/
5. Multi-Factor Authentication [Електронний ресурс] - Режим доступу: https://auth0.com/docs/secure/multi-factor-authentication
6. Understanding How PKI Works [Електронний ресурс] - Режим доступу: https://www.okta.com/uk/identity-101/public-key-infrastructure/
7. Cloud computing used by 42% of enterprises [Елекронний ресурс] - режим доступу: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20211209-2
8. The U.S. Rules the Cloud: [Електронний ресурс] - Режим доступу: https://www.statista.com/chart/22253/estimated-revenue-public-cloud-computing-top-markets/
9. Cloud computing for business yet to go mainstream in the EU [Електронний ресурс] — Режим доступу: https://ec.europa.eu/eurostat/web/products-eurostatnews/-/ddn-20210121-1

10. The Cost of Cloud, a Trillion Dollar Paradox [Електронний ресурс] - Режим доступу: https://a16z.com/the-cost-of-cloud-a-trillion-dollar-paradox
11. CPR [електронний ресурс] - Режим доступу: https://www.checkpoint.com/
12. Policy Simulator [електронний ресурс] - Режим доступу: https://policysim.aws.amazon.com/
13. Lambda Pricing [електронний ресурс] - Режим доступу: https://aws.amazon.com/lambda/pricing/
14. Padmavathi Iyer and Amirreza Masoumzadeh. 2018. Mining Positive and Negative Attribute-Based Access Control Policy Rules. In Proceedings of The 23rd ACM Symposium on Access Control Models & Technologies (SACMAT), Indianapolis, IN, USA, June 13–15, 2018 (SACMAT '18), 12 pages. https://doi.org/10.1145/3205977.3205988
15. Big O notation [електронний ресурс] - Режим доступу: https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/
16. Купрієнко, А., & Гальчинський, Л. (2023). Агентна модель майнінгу прав доступу в хмарних середовищах. Scientific Collection «InterConf», (184), 482–490. Retrieved from https://archive.interconf.center/index.php/conference-proceeding/article/view/5128
17. Galchynsky, L., & Murtazina, A. (2022). Vulnerability detection in the network traffic flow of the RADIUS protocol based on the object-oriented model. Theoretical and Applied Cybersecurity, 4(1).
18. What does DDoS Mean [електронний ресурс] - Ресурс доступу: https://www.imperva.com/learn/ddos/denial-of-service/
19. What is Wiper Malware [електронний ресурс] - Ресурс доступу: https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-malware/what-is-wiper-malware/

20. What is SIEM [електронний ресурс] - Ресурс доступу: https://www.ibm.com/topics/siem
21. What is “Social Engineering” [електронний ресурс] - Режим доступу: https://www.enisa.europa.eu/topics/incident-response/glossary/what-is-social-engineering
22. Кібератаки на інфраструктуру [електронний ресурс] - Режим доступу: https://ck-oda.gov.ua/novyny-cherkaskoyi-oblasti/z-pochatku-vijni-sbu-nejtralizuvala-majzhe-35-tis-kiberatak-na-organi-vladi-ta-obyekti-infrastrukturi/
23. Average Weekly Global Cyberattacks [електронний ресурс] - Режим доступу: https://blog.checkpoint.com/2023/01/05/38-increase-in-2022-global-cyberattacks/
24. Attacks between 2021 and 2022 [електронний ресурс] - Режим доступу: https://blog.checkpoint.com/2023/01/05/38-increase-in-2022-global-cyberattacks/
25. Cloud-Based PKI [ електронний ресурс] - Режим доступу: https://www.hashicorp.com/blog/pki-hosting-cloud-based-pki-vs-self-managed
26. MAC, DAC, RBAC, ABAC [електронний ресурс] - Режим доступу: https://upload.wikimedia.org/wikipedia/commons/thumb/a/a2/Comparison_of_different_Access_Control_Mechanisms_in_a_cloud_environment.svg/1200px-Comparison_of_different_Access_Control_Mechanisms_in_a_cloud_environment
27. Vidhyacharan Bhaskar, Indu I, Rubesh Anand. Engineering Science and Technology An International Journal, Pages 574-588, May 2018 doi:10.1016/j.jestch.2018.05.010

28. The U.S. Rules The Cloud [електронний ресурс] - Режим доступу: https://www.statista.com/chart/22253/estimated-revenue-public-cloud-computing-top-markets/

# ДОДАТОК А

## ПРОГРАМНИЙ КОД

### lfm.py

```
import json
import gzip
import boto3


def lambda_handler(event, context):
   s3_event = event['Records'][0]['s3']
   bucket = s3_event['bucket']['name']
   key = s3_event['object']['key']


   s3 = boto3.client('s3')
   response = s3.get_object(Bucket=bucket, Key=key)
   with gzip.open(response['Body'], 'rt', encoding='utf-8') as file:
       log_content = file.read()
   logs = [json.loads(log) for log in log_content.strip().split('\n')]


   iam_client = boto3.client('iam')


   eventsToCatch = ['StopInstances']
   for log in logs:
           log_data = log
           if 'Records' in log_data:
               for record in log_data['Records']:
                   try:
                       print(json.dumps(record, indent=2))
```

```
                        print("=" * 50)



                        userName = record['userIdentity']['userName']



                        response =
iam_client.list_user_tags(UserName=userName)

                        tags = response.get('Tags', [])

                        tags_description = ", ".join([f"{i+1}) {tag['Key']}
- {tag['Value']}" for i, tag in enumerate(tags)])

                        print(f"Користувач {userName} має наступні атрибути:
{tags_description}")

                        eventName = record['eventName']

                        print(tags)

                        if eventName in eventsToCatch:

                            tags_to_deny = {'role':'pm', 'project':'feel'}

                            arn =
create_policy_for_tagged_users(iam_client,eventName, tags_to_deny)

                            matching_users =
find_users_with_matching_tags(tags)

                            user_tags_and_policies =
get_user_tags_and_policies(iam_client, matching_users, arn)

                            conficted, tag_set =
find_conflicting_policies_for_users(user_tags_and_policies, tags,
eventName)

                            users_with_resolved_conflicts =
resolve_conflicts_based_on_tags(conficted, tag_set)

                            final =
update_user_policies(user_tags_and_policies, users_with_resolved_conflicts)

                            return {'statusCode': 200, 'body': final, 'arn':
arn, 'eventName': eventName}



                    except (KeyError, json.JSONDecodeError) as e:

                        print(f"Помилка обробки логу: {e}")

                        continue



    return {'statusCode': 500, 'error': error }
```

```
def find_users_with_matching_tags(target_tags):
    iam_client = boto3.client('iam')
    paginator = iam_client.get_paginator('list_users')
    matching_users = []

    for page in paginator.paginate():
        for user in page['Users']:
            user_tags =
iam_client.list_user_tags(UserName=user['UserName']).get('Tags', [])
            match_result = has_matching_tags(user_tags, target_tags)
            if match_result['matches']:
                matching_users.append(user['UserName'])
                format_matching_tags(user['UserName'],
match_result['matching_tags'])

    return matching_users

def has_matching_tags(user_tags, target_tags):
    user_tags_dict = {tag['Key']: tag['Value'] for tag in user_tags}
    matching_tags = []

    for tag in target_tags:
        if tag['Key'] in user_tags_dict and user_tags_dict[tag['Key']] ==
tag['Value']:
            matching_tags.append(tag)

    return {'matches': len(matching_tags) > 0, 'matching_tags':
matching_tags}

def format_matching_tags(user_name, matching_tags):
    if matching_tags:
```

```
        matching_tags_str = ", ".join([f"{tag['Key']} - {tag['Value']}" for
tag in matching_tags])

        print(f"З користувачем {user_name} однакові наступні атрибути:
{matching_tags_str}")



def analyze_user_policies(iam_client, user):

    user_policy_details = {'Allow': [], 'Deny': []}



    attached_policies =
iam_client.list_attached_user_policies(UserName=user)['AttachedPolicies']

    for policy in attached_policies:

        policy_version =
iam_client.get_policy(PolicyArn=policy['PolicyArn'])['Policy']['DefaultVers
ionId']

        policy_document =
iam_client.get_policy_version(PolicyArn=policy['PolicyArn'],
VersionId=policy_version)['PolicyVersion']['Document']



        for statement in policy_document['Statement']:

            if 'Action' in statement:

                actions = statement['Action']

                actions = actions if isinstance(actions, list) else
[actions]



                if statement['Effect'] == 'Allow':

                    if statement['Resource'] == '*' or
isinstance(statement['Resource'], list) and '*' in statement['Resource']:

                        user_policy_details['Allow'].extend(actions)

                elif statement['Effect'] == 'Deny':

                    if statement['Resource'] == '*' or
isinstance(statement['Resource'], list) and '*' in statement['Resource']:

                        user_policy_details['Deny'].extend(actions)



    user_policy_details['Allow'] = list(set(user_policy_details['Allow']))

    user_policy_details['Deny'] = list(set(user_policy_details['Deny']))
```

```
    return user_policy_details


def find_users_with_tag(iam_client, tag_key, tag_value):
    paginator = iam_client.get_paginator('list_users')
    matching_users = []

    for page in paginator.paginate():
        for user in page['Users']:
            user_tags =
iam_client.list_user_tags(UserName=user['UserName']).get('Tags', [])
            for tag in user_tags:
                if tag['Key'] == tag_key and tag['Value'] == tag_value:
                    matching_users.append(user['UserName'])
                    break

    return matching_users



def get_users(iam_client):
    paginator = iam_client.get_paginator('list_users')
    return [user['UserName'] for page in paginator.paginate() for user in
page['Users']]


def get_user_tags_and_policies(iam_client, users, new_policy_arn):
    user_details = {}
    for user in users:
        user_details[user] = {
            'tags': get_user_tags(iam_client, user),
            'policies': get_user_policies(iam_client, user, new_policy_arn)
        }

    return user_details
```

```
def get_user_tags(iam_client, user):
    response = iam_client.list_user_tags(UserName=user)
    return {tag['Key']: tag['Value'] for tag in response['Tags']}


def get_user_policies(iam_client, user, new_policy_arn):
    attached_policies =
iam_client.list_attached_user_policies(UserName=user)['AttachedPolicies']
    user_tags = get_user_tags(iam_client, user)
    policies = {}


    policy_conditions = get_policy_conditions(iam_client, new_policy_arn)


    for policy in attached_policies:
        add_policy_details(iam_client, policy['PolicyArn'], policies)
    print(f"Користувач {user} має наступні права:")
    for action, effects in policies.items():
        print(f" - {action}: {effects}")


    if check_user_meets_policy_conditions(user_tags, policy_conditions):
        add_policy_details(iam_client, new_policy_arn, policies)




    return policies


def get_policy_conditions(iam_client, policy_arn):
    policy_version =
iam_client.get_policy(PolicyArn=policy_arn)['Policy']['DefaultVersionId']
    policy_doc = iam_client.get_policy_version(PolicyArn=policy_arn,
VersionId=policy_version)['PolicyVersion']['Document']


    conditions = {}
    for statement in policy_doc['Statement']:
```

```
        if 'Condition' in statement and 'StringEquals' in
statement['Condition']:
            conditions.update(statement['Condition']['StringEquals'])
    return conditions


def check_user_meets_policy_conditions(user_tags, policy_conditions):
    for condition_key, condition_value in policy_conditions.items():
        _, tag_key = condition_key.split('/', 1)

        if tag_key in user_tags and user_tags[tag_key] == condition_value:
            return True

    return False




def add_policy_details(iam_client, policy_arn, policies):
    default_version_id =
iam_client.get_policy(PolicyArn=policy_arn)['Policy']['DefaultVersionId']
    policy_doc = iam_client.get_policy_version(
        PolicyArn=policy_arn,
        VersionId=default_version_id
    )['PolicyVersion']['Document']

    for statement in policy_doc['Statement']:
        if 'Action' in statement:
            actions = statement['Action']
            actions = actions if isinstance(actions, list) else [actions]
            effect = statement['Effect']

            for action in actions:
                if action in policies:
```

```
                    if effect not in policies[action]:
                        policies[action].append(effect)
                else:
                    policies[action] = [effect]




def create_policy(iam_client, policy_name, policy_document):
    try:
        response = iam_client.create_policy(
            PolicyName=policy_name,
            PolicyDocument=policy_document
        )
        policy_arn = response['Policy']['Arn']

        return policy_arn
    except iam_client.exceptions.EntityAlreadyExistsException:

        return f'arn:aws:iam:::policy/{policy_name}'



def create_policy_for_tagged_users(iam_client, action, tags):
    policy_name = f"Deny{action}ForTaggedUsers"
    action_with_prefix = f"ec2:{action}"
    conditions = {"StringEquals": {}}

    for tag_key, tag_value in tags.items():
        conditions["StringEquals"][f"aws:PrincipalTag/{tag_key}"] =
tag_value

    policy_document = json.dumps({
        "Version": "2012-10-17",
```

```
        "Statement": [

            {

                "Effect": "Deny",

                "Action": action_with_prefix,

                "Resource": "*",

                "Condition": conditions

            }

        ]

    })



    print(f"Створено нову політику {policy_name} на заборону дії
{action_with_prefix}")

    policy_arn = create_policy(iam_client, policy_name, policy_document)

    return policy_arn



def find_policies_for_users(user_tags_and_policies):

    for user, details in user_tags_and_policies.items():

        policies = details['policies']

        print(f"Користувач {user} має наступні права:")

        for action, effects in policies.items():

            print(f" - {action}: {', '.join(effects)}")



def find_conflicting_policies_for_users(user_tags_and_policies,
search_tags_list, eventName):

    search_tags = {tag['Key']: tag['Value'] for tag in search_tags_list}

    users_with_conflicts = {}

    user_attributes_set = set()

    users_with_deny = {}



    for user, details in user_tags_and_policies.items():

        user_tags = details['tags']

        policies = details['policies']
```

```
        conflicting_policies = {}


        for action, effects in policies.items():
            if len(effects) > 1 and 'Allow' in effects and 'Deny' in
effects:
                conflicting_policies[action] = list(effects)


        if conflicting_policies:
            users_with_conflicts[user] = {
                'tags': user_tags,
                'conflicting_policies': conflicting_policies
            }


            for conflict_policy_name in conflicting_policies:
                print(f"Користувач {user} має конфлікт доступу за дією
{conflict_policy_name}")


            if all(user_tags.get(tag) == value for tag, value in
search_tags.items()):
                users_with_conflicts[user]['tags']['deny'] = eventName
                print(f"Додаємо новий атрибут deny={eventName} для
розв'язання користувачу {user} через повне співпадіння атрибутів під час
вибору атрибутів для майнінгу прав")
                users_with_deny['deny'] = eventName


    return users_with_conflicts, users_with_deny




def resolve_conflicts_based_on_tags(conflicting_policies_data,
search_tags):
    resolved_policies = {}
    print('Починаємо вирішення конфлікту доступу:')
    for user, details in conflicting_policies_data.items():
        user_tags = details['tags']
```

```
        conflicting_policies = details['conflicting_policies']



        resolved_actions = {}

        for action, effects in conflicting_policies.items():

            if 'Allow' in effects and 'Deny' in effects:

                if all(tag in user_tags and user_tags[tag] ==
search_tags.get(tag) for tag in search_tags):

                    resolved_actions[action] = ['Deny']

                    print(f"Застосовуємо Deny на дію {action} до користувача
{user} через наявність контролюючого атрибуту")

                else:

                    resolved_actions[action] = ['Allow']

                    print(f"Не застосовуємо Deny на дію {action} до
користувача {user} через відсутність контролюючого атрибуту")




        if resolved_actions:

            resolved_policies[user] = {'policies': resolved_actions}

    return resolved_policies


def update_user_policies(existing_data, resolved_conflicts):

    print("Фінальна політика доступів:")



    for user, details in existing_data.items():

        if user in resolved_conflicts and 'policies' in details:

            resolved_user_policies = resolved_conflicts[user]['policies']

            for action, effect in resolved_user_policies.items():

                if action in details['policies']:

                    details['policies'][action] = effect

        formatted_policies = "\n  - ".join([f"{action}:
{details['policies'][action]}" for action in details['policies']])

        print(f"{user}:\n  - {formatted_policies}")

    print("Надсилаю дані до ЛФА")
```

```
    return existing_data


def attach_policy_to_users_with_deny(existing_data, policy_arn, action):

    iam_client = boto3.client('iam')

    action_with_prefix = f"ec2:{action}"


    for user, details in existing_data.items():

        if 'policies' in details and action_with_prefix in
details['policies'] and 'Deny' in details['policies'][action_with_prefix]:

            iam_client.attach_user_policy(UserName=user,
PolicyArn=policy_arn)
```

lfa.py

```
import json

import boto3

def lambda_handler(event, context):

    body = event['responsePayload']['body']

    arn = event['responsePayload']['arn']

    eventName = event['responsePayload']['eventName']

    attach_policy_to_users_with_deny(body, arn, eventName)

    return {

        'statusCode': 200,

        'body': 'OK'

    }

def attach_policy_to_users_with_deny(existing_data, policy_arn, action):

    iam_client = boto3.client('iam')

    action_with_prefix = f"ec2:{action}"


    for user, details in existing_data.items():

        if 'policies' in details and action_with_prefix in
details['policies'] and 'Deny' in details['policies'][action_with_prefix]:

            iam_client.attach_user_policy(UserName=user,
PolicyArn=policy_arn)
```

```
  print("Нові політики успішно присвоєні")
```

**LFM_config.json**

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Action": "logs:CreateLogGroup",
     "Resource": "arn:aws:logs:us-east-1::*"
   },
   {
     "Effect": "Allow",
     "Action": [
       "logs:CreateLogStream",
       "logs:PutLogEvents"
     ],
     "Resource": [
       "arn:aws:logs:us-east-1::log-group:/aws/lambda/:*"
     ]
   },
```

```
{
  "Effect": "Allow",
  "Action": [
    "iam:ListUserTags",
    "iam:ListAttachedUserPolicies"
  ],
  "Resource": "arn:aws:iam:::user/*"
},
{
  "Effect": "Allow",
  "Action": "iam:ListUsers",
  "Resource": "arn:aws:iam:::*"
},
{
  "Effect": "Allow",
  "Action": [
    "iam:GetPolicy",
    "iam:GetPolicyVersion",
    "iam:CreatePolicy",
    "iam:AttachUserPolicy",
    "iam:ListEntitiesForPolicy",
    "iam:DetachUserPolicy"
  ],
  "Resource": "*"
}
```

```
 ]

}
```

**Users1-5_config.json**

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Action": [
       "ec2:StartInstances"
     ],
     "Resource": "*"
   }
 ]
}
```

**Users6-10_config.json**

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Action": [
       "ec2:StopInstances"
```

```
      ],
      "Resource": "*"
    }
  ]
}
```

**DenyStopInstancesForTaggedUsers.json**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "ec2:StopInstances",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/role": "pm",
          "aws:PrincipalTag/project": "feel"
        }
      }
    }
  ]
}
```