

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

НАУКОВО-НАВЧАЛЬНИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«До захисту допущено»

Завідувач кафедри ІБ

_____Д.В.Ланде

“___” 2022 р.

**Дипломна робота
на здобуття ступеня бакалавра**

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

на тему: Агентна модель безпеки хмарних середовищ

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-84
(шифр групи)

Вацик Максим Мирославович

(прізвище, ім'я, по батькові)

(підпис)

Керівник: _____**к.т.н., доцент Гальчинський Леонід Юрійович**

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Здобувач вищої освіти _____

(підпис)

Київ – 2022 року

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

До захисту допущено
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

« _____ » _____ 2022 р.

**Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи, технології та математичні
методи кібербезпеки»
спеціальності 125 «Кібербезпека»**

на тему: Агентна модель безпеки хмарних середовищ

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-84
(шифр групи)

Вацик Максим Мирославович
(прізвище, ім'я, по батькові)

(підпис)

Керівник доцент, к.т.н., доцент Гальчинський Леонід Юрійович _____
(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 – 2022 року

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

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 125 «Кібербезпека»

Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Дмитро ЛАНДЕ
 (підпис)

«___» _____ 2022 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Вацiku Максимy Мирославовичу
 (прізвище, ім'я, по батькові)

1. Тема роботи: Агентна модель безпеки хмарних середовищ,
 керівник роботи: Гальчинський Леонід Юрійович, доцент, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом по університету від «___» _____ 2022 р. №
2. Термін подання здобувачем вищої освіти роботи ____ червня 2022 р.
3. Вихідні дані до роботи _____

4. Зміст роботи _____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація
6. Дата видачі завдання 1 грудня 2021 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання	01.12.2021	виконано
2	Огляд та опрацювання інформаційних джерел	01.01.2022-15.01.2022	виконано
3	Ознайомлення з концепцією хмарних середовищ	15.01.2022 – 15.02.2022	виконано
4	Аналіз трендів атак на хмарні середовища	15.02.2022 – 05.03.2022	виконано
5	Виокремлення найменш захищених компонентів хмарних середовищ	05.03.2022 – 10.03.2022	виконано
6	Постановка актуальних проблем в традиційній моделі безпеки хмарних середовищ	10.03.2022 – 17.03.2022	виконано
7	Постановка вимог до моделі безпеки хмарних середовищ	17.03.2022 – 25.03.2022	виконано
8	Побудова діаграм класів та послідовностей для класичної моделі безпеки хмарних середовищ	25.03.2022 – 03.04.2022	виконано
9	Розробка агентної моделі безпеки та опис її компонентів. Побудова відповідних діаграм класів та послідовностей	03.04.2022 – 17.04.2022	виконано
11	Розробка прикладної програми згідно з агентною моделлю	17.04.2022 – 17.05.2022	виконано
12	Налаштування розробленої прикладної програми та її аналогів з метою порівняльного аналізу	17.05.2022 – 24.05.2022	виконано
13	Тестування прикладної програми та порівняльний аналіз з аналогами	24.05.2022 – 01.06.2022	виконано

Здобувач вищої освіти

(підпис)

Керівник роботи

(підпис)

Максим ВАЦИК
(Власне ім'я, ПРІЗВИЩЕ)Леонід ГАЛЬЧИНСЬКИЙ
(Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Робота обсягом 92 сторінки, містить 30 ілюстрацій, 1 таблицю, 34 літературні посилання та 1 додаток.

Мета роботи – створення агентної моделі безпеки для хмарних середовищ для уніфікації підходу до побудови інфраструктур між багатьма хмарними середовищами та покращення їх безпеки.

Методи дослідження – ознайомлення з тематичною літературою, аналіз даних з відкритих джерел про атаки на хмарні обчислення та недоліки існуючих систем захисту. Виокремлення найменш захищених компонентів хмарних середовищ та визначення актуальних вимог до моделі безпеки. Побудова агентної моделі хмарних середовищ та порівняльний аналіз прикладної програми, що реалізує запропоновану модель, з наявними аналогами.

Наукова новизна дослідження полягає в побудові агентної моделі безпеки, що узагальнює підхід до побудови безпеки між багатьма хмарними середовищами за допомогою програмних агентів та дозволяє легко масштабувати ресурси в новій інфраструктурі.

Прикладна програма була розроблена у програмному середовищі Visual Studio Code 1.68.0 з використанням мови програмування Python 3.10.4.

Ключові слова: хмарне середовище, безпека хмарних середовищ, модель безпеки, агентна модель безпеки, атака на хмарне середовище, автентифікація, авторизація, аудит подій.

ABSTRACT

The work consists of 3 sections, contains 30 illustrations, 1 table, 1 appendix and 34 references, the volume of work is 88 pages.

The aim of this work is to create an agent-based security model for cloud environments with the purpose of unifying the approach of designing infrastructures in-between multiple different cloud environments and improve the overall security level.

The research method involves studying of related scientific works, open-source data analysis on cloud computing attack trends and weak points in the classical cloud security model. Identification of the least protected assets in the cloud environments and defying modern demands for a cloud security model. Development of an agent-based security model for cloud environments as well as an application that implements the proposed model. Extensive analysis of the resulting application with the existing counterparts.

The scientific novelty of the study resides in the development of an agent-based model that allows for constructing a security model in-between multiple different cloud environments in a unified manner with the help of program agents, and can be heavily modifiable and expandable with new assets.

The application was developed with the help of Visual Studio Code 1.68.0 IDE in Python 3.10.4 programming language.

Keywords: cloud environment, cloud environment security, security model, agent-based security model, attacks on cloud environment, authentication, authorization, accounting.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів.....	8
Вступ.....	9
1 Проблеми безпеки хмарних середовищ	11
1.1 Хмарні середовища	11
1.2 Проблеми інформаційної безпеки в хмарних середовищах	14
1.3 Принципи безпеки.....	18
1.4 Компоненти моделі безпеки в хмарному середовищі	18
1.5 Проблема розмежування хмарних середовищ	20
1.6 Проблема централізованого керування доступом	20
1.7 Моделі автентифікації	25
1.8 Моделі розмежування прав доступу	27
Висновки за розділом 1	29
2 Агентна модель безпеки хмарних середовищ	30
2.1 Класична модель безпеки хмарного середовища	30
2.2 Агентна модель безпеки	36
2.3 Запропонована модель автентифікації.....	41
2.4 Запропонована модель авторизації	45
2.5 Запропонована модель аудиту подій системи.....	49
2.6 Блокування агентів та позбавлення доступу	49
Висновки за розділом 2.....	50
3 Порівняльний аналіз та тестування програмної реалізації агентої моделі.....	51
3.1 Огляд прикладної програми-реалізації агентної моделі	51
3.2 Порівняльний аналіз програмних агентів з наявними альтернативами	57
Висновки за розділом 3	63
Висновки.....	64
Перелік джерел посилань	66
Додатки.....	Error! Bookmark not defined.
Додаток А Програмний код	71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACE – Access Control Entry

ACL – Access Control List

DAC – Discretionary Access Control

ECDSA – Elliptic Curve Digital Signature Algorithm, алгоритм підпису асиметричної криптографії

Federated authentication – автентифікація, делегована третій стороні

First party – перша сторона, ресурс під безпосереднім контролем користувача

IAM – Identity and Access Management

MAC – Mandatory Access Control

MitM – Man-in-the-Middle attack

PKI – Public Key Infrastructure

ProtoBuf – формат серіалізації даних для їх подальшої передачі

RBAC – Role-Based Access Control

Third party – третя сторона, сторонній ресурс, що не контролюється користувачем

AAA – автентифікація, авторизація, аудит подій

KI – конфіденційна інформація

MA – мобільний агент

OR – обчислювальні ресурси

ПЗ – програмне забезпечення

ПМА – програма-мобільний агент

ПЦА – програма-центральний агент

ТЗ – технічні засоби

ХС – хмарне середовище

ЦА – центральний агент

ВСТУП

У зв'язку з швидким розвитком мережеских та комп'ютерних технологій в минулому десятилітті, і появою принципово нових підходів до використання обчислювальних ресурсів, сьогодні гостро постала проблема захисту цілісності, конфіденційності та доступності даних користувачів в інформаційно-комунікаційних системах з поділюваним доступом до обчислювальних ресурсів (ОР).

Метою даної роботи є вирішення даної проблеми, що потребує розробки комплексної моделі доступу до налаштувань поділюваних ОР та конфіденційної інформації (КІ) користувачів, що ними оброблюється, передається та зберігається. Важливою складовою даної моделі доступу є повне розмежування окремих користувачів на поділюваних ОР для унеможливлення доступу до чужої КІ.

Хмарні середовища (ХС), стають дедалі більш поширеними серед системних інженерів. Тому розробка гнучкої та безпечної моделі доступу до ресурсів у ХС привертає особливу увагу в останні роки.

Модель доступу має уніфікувати керування доступом до усіх наявних типів ОР, ефективно використовувати мережескі та обчислювальні ресурси при високих навантаженнях для мінімізації витрат, та також надавати гнучкий інтерфейс керування інженерів.

Об'єктом дослідження є хмарні середовища та організація безпеки в них.

Предметом дослідження є недоліки класичної моделі безпеки хмарних середовищ, що породжують вразливості системи.

Методи дослідження – ознайомлення з тематичною літературою, аналіз даних з відкритих джерел про атаки на хмарні обчислення та недоліки існуючих систем захисту. Виокремлення найменш захищених компонентів хмарних

середовищ та визначення актуальних вимог до моделі безпеки. Побудова діаграм моделей безпеки хмарних середовищ з метою аналізу взаємодії їх компонентів.

Наукова новизна дослідження полягає в побудові агентної моделі безпеки, що узагальнює підхід до побудови безпеки між багатьма хмарними середовищами за допомогою програмних агентів та дозволяє легко масштабувати ресурси в новій інфраструктурі.

Прикладна програма-реалізація агентної моделі безпеки, що є результатом даної роботи, була подана на реєстрацію авторського права в ДП «Український інститут інтелектуальної власності» Укрпатент.

Завданням цієї роботи є:

1. Аналіз даних про атаки на хмарні середовища, визначення проблем та недоліків існуючої моделі безпеки.
2. Постановка актуальних вимог до моделі безпеки.
3. Розробка агентної моделі доступу, що передбачає запровадження централізованого керування доступом за допомогою програмних агентів для кожного суб'єкта в ХС, які інкапсулюватимуть усі обчислення прав та взаємодію з об'єктами в собі.
4. Програмна реалізація агентної моделі безпеки

Перевага агентної моделі в тому, що вона може легко масштабуватися при додаванні нових ресурсів до ХС та надає великий простір для гнучкого налаштування прав доступу за допомогою створення нових політик до агентів.

Запропонований підхід дозволить побудувати рівень абстракції над усіма суб'єктами в системі та узагальнити керування доступом за допомогою визначених в агентах політик доступу.

1 ПРОБЛЕМИ БЕЗПЕКИ ХМАРНИХ СЕРЕДОВИЩ

1.1 Хмарні середовища

Хмарні середовища (ХС) – тип інфраструктури з поділюваним доступом до ОР, що надає користувачеві зручну платформу для швидкого розгортання та масштабування власного програмного забезпечення (ПЗ), однак не надає прямого контролю над технічними засобами (ТЗ), що лежать в основі [1]. Забезпеченням основних ТЗ (налаштування мережі, ОС, тощо) займається надавач послуг ХС, третя сторона. Забезпечення додаткових ТЗ, а також їх відповідне налаштування покладається на користувача ХС. Даний підхід дозволяє пришвидшити робочі процеси щодо розгортання ПЗ та зменшити відповідні витрати.

1.1.1 Класифікація хмарних середовищ

Найбільш поширеними способами класифікації ХС є класифікація за власником ОР та класифікація за видом наданих послуг [2].

За власником обчислювальних ресурсів хмарні середовища поділяють на:

- Приватні ХС. Користувач хмарного середовища є власником обчислювальних ресурсів, на яких розгорнуто ХС.
- Публічні ХС. Власником обчислювальних ресурсів є компанія-посередник, що здає їх в оренду клієнтам за грошову плату. Приклади компаній-посередників: Google, Amazon, Microsoft, IBM, тощо.
- Гібридні ХС. Хмарне середовище побудоване з використанням як приватних ОР клієнта, так і орендованих у компанії-посередника.

Зазвичай, гібридні ХС є етапом переходу від приватного ХС до публічного.

За типом наданих послуг, ХС поділяють на:

- Infrastructure-as-a-Service (IaaS)

Надавач послуг ХС бере відповідальність за налаштування та технічну підтримку лише фізичного обладнання. Користувачеві надається можливість власноруч налаштувати ОС та ПЗ.

- Platform-as-a-Service (PaaS)

Надавач послуг ХС бере відповідальність за налаштування та підтримку роботи серверного обладнання та операційної системи. Користувачеві надається ОС, на яку він може ставити власне ПЗ.

- Service-as-a-Service (SaaS)

Надавач послуг ХС повністю контролює серверне обладнання, ОС та ПЗ. Користувачеві надається можливість використовувати ПЗ, що пропонує надавач послуг.

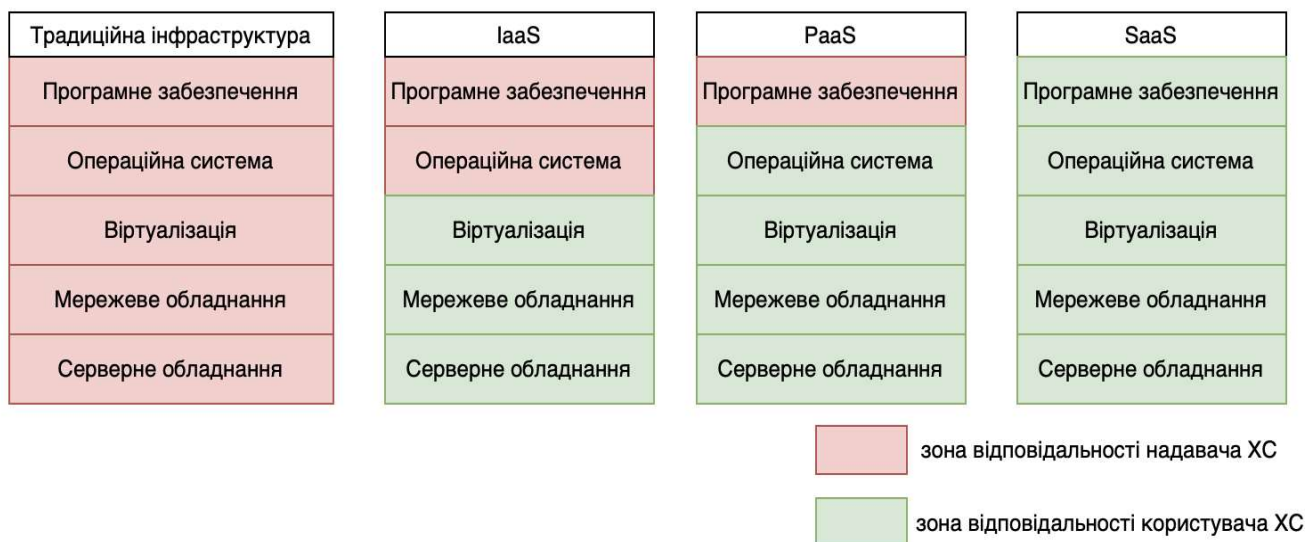


Рисунок 1.1 – Зона відповідальності сторін у різних типах інфраструктур.

1.1.2 Поширеність хмарних середовищ

Простота та дешевизна хмарних середовищ зумовили стрімкий перехід більшості компаній до їх використання для швидкої побудови нових інфраструктур на базі ХС. Так, капіталізація ринку у період з 2008 по 2020й рік виросла у 27 разів з 5.82 млрд. дол. США до 159.28 млрд. дол. США.

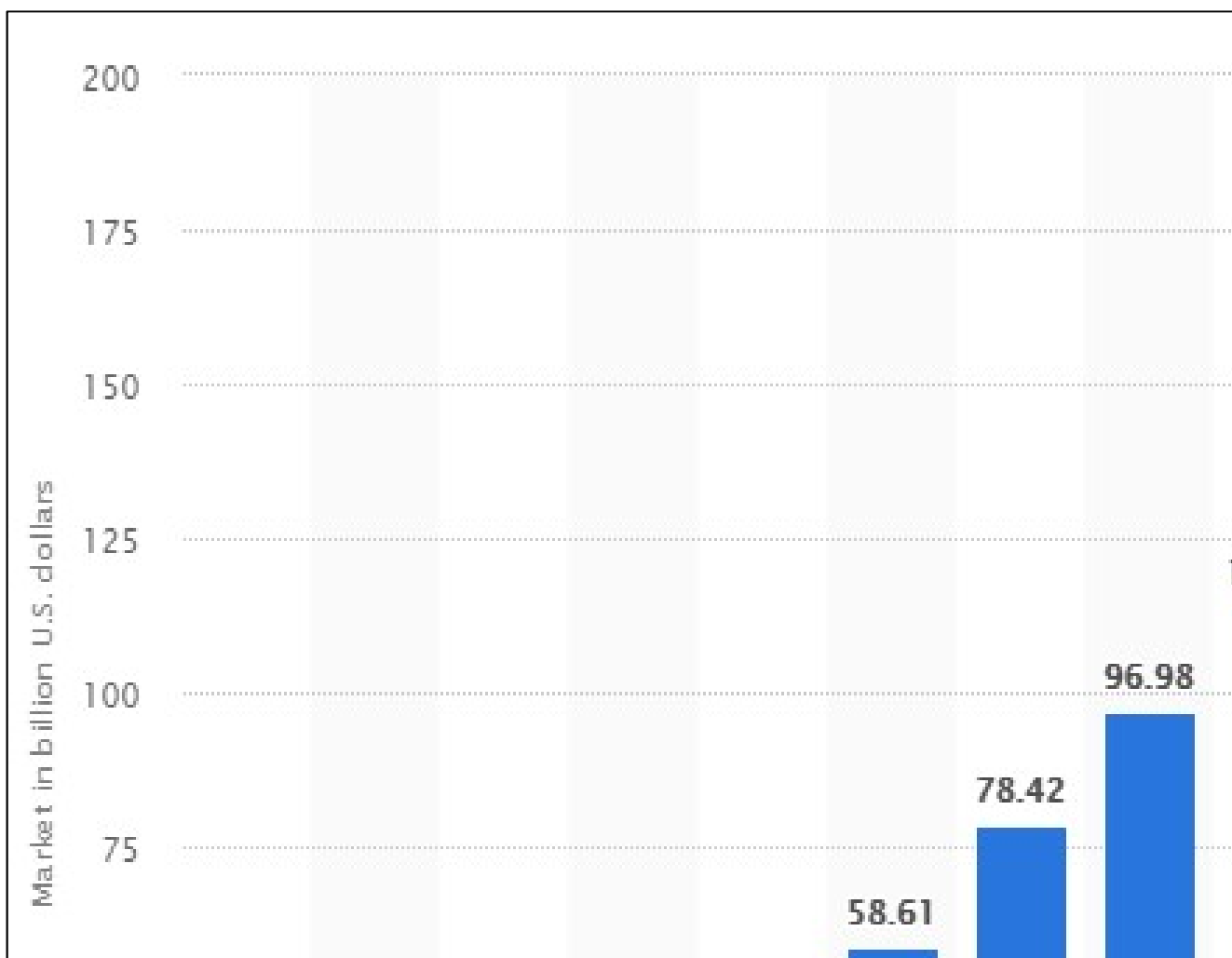


Рисунок 1.2 – Капіталізація ринку хмарних середовищ за 12 років. [16]

Більш того, за даними Євростату станом на 2020й рік, 50% бізнесів 9 з 28 країн ЄС використовували ХС у своїй інфраструктурі. У 16 країнах частка рішень на базі ХС становила від 20% до 40%. Найбільш використовуваними ХС є у Фінляндії, Швеції та Данії, де їх частка перевищує 60%.

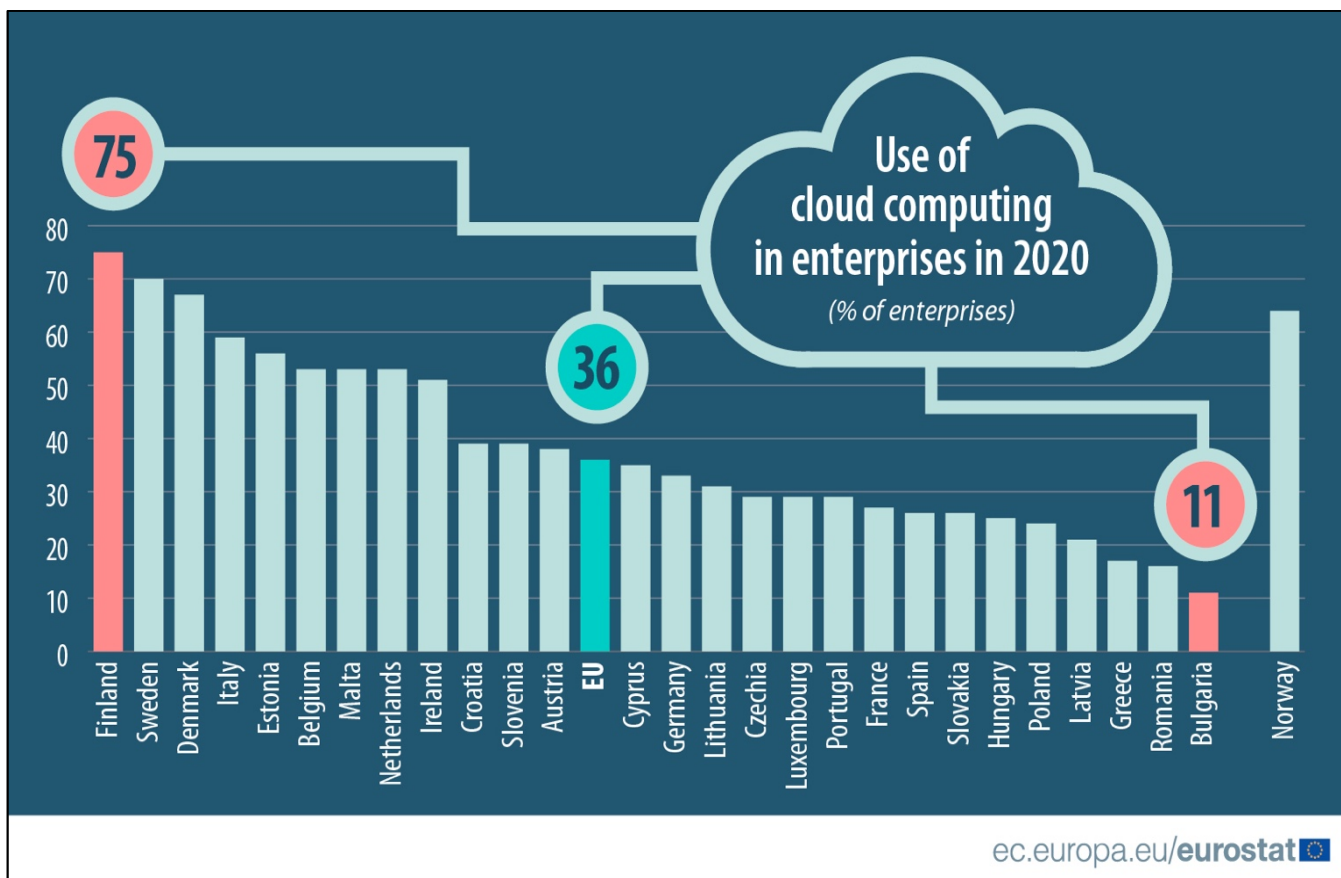


Рисунок 1.3 – Поширеність хмарних середовищ в ІТ секторах країн ЄС.

2020 рік. [17]

Швидке прийняття нових підходів до побудови ІТ інфраструктур, таких, як делегування відповідальності за основні системні ресурси і використання спільних обчислювальних ресурсів, що фізично знаходяться у власності третьої сторони, породив принципово нові потенційні загрози безпеки даних користувача.

1.2 Проблеми інформаційної безпеки в хмарних середовищах

До найбільших проблем безпеки в хмарних середовищах відносять [4]:

- Розмежування окремих користувачів. Без ізоляції ресурсів користувачів існує загроза компрометації даних, що оброблюються, зберігаються, або передаються, іншими користувачами, що перехоплять контроль над

ресурсом. Позаяк сама концепція хмарного середовища припускає постійну ротацію ОР між користувачами в залежності від їх потреб, повна ізоляція на фізичному рівні не є можливою.

- Організацію контролю доступу до ресурсів. Без надання інструментів користувачеві для побудови власної моделі доступу всередині інфраструктури та обмеження доступу до наявних ресурсів, що знаходиться в хмарі, результатом будь-якої успішної атаки буде компрометація усіх даних в системі. Сучасні вимоги до хмарних середовищ передбачають існування двох моделей доступу: доступу до ресурсів всередині інфраструктури користувача та доступу до ресурсів між інфраструктурами декількох користувачів.
- Організацію контролю доступу до інтерфейсу керування. Кількість атак на інтерфейси керування хмарних середовищ з роками дедалі більшає, адже адміністративний доступ гарантує успішну компрометацію інфраструктури та дозволяє обійти існуючі механізми захисту всередині неї [5].
- Узгодження компонентів безпеки між хмарними середовищами. Однією з основних вимог до моделі безпеки є можливість роботи між основними видами хмарних середовищ, як з точки зору бізнес вимог для побудови гнучких інфраструктур (гібридні ХС, приватні ХС), так і з точки зору вимог безпеки – відсутність уніфікованого стандарту для взаємодії між декількома ХС [14] несе в собі суттєві загрози компрометації даних та вимагає розробки додаткових систем безпеки [15].

Відсутність загальноприйнятих стандартів безпеки у хмарних обчисленнях, новий підхід до контролю доступу до ОР, та широкого досвіду побудови нових інфраструктур у системних інженерів зробили ХС привабливою ціллю для

зловмисників. Більш ніж 70% усіх інфраструктур у хмарі зазнали атак з боку хакерів станом на 2019й рік.

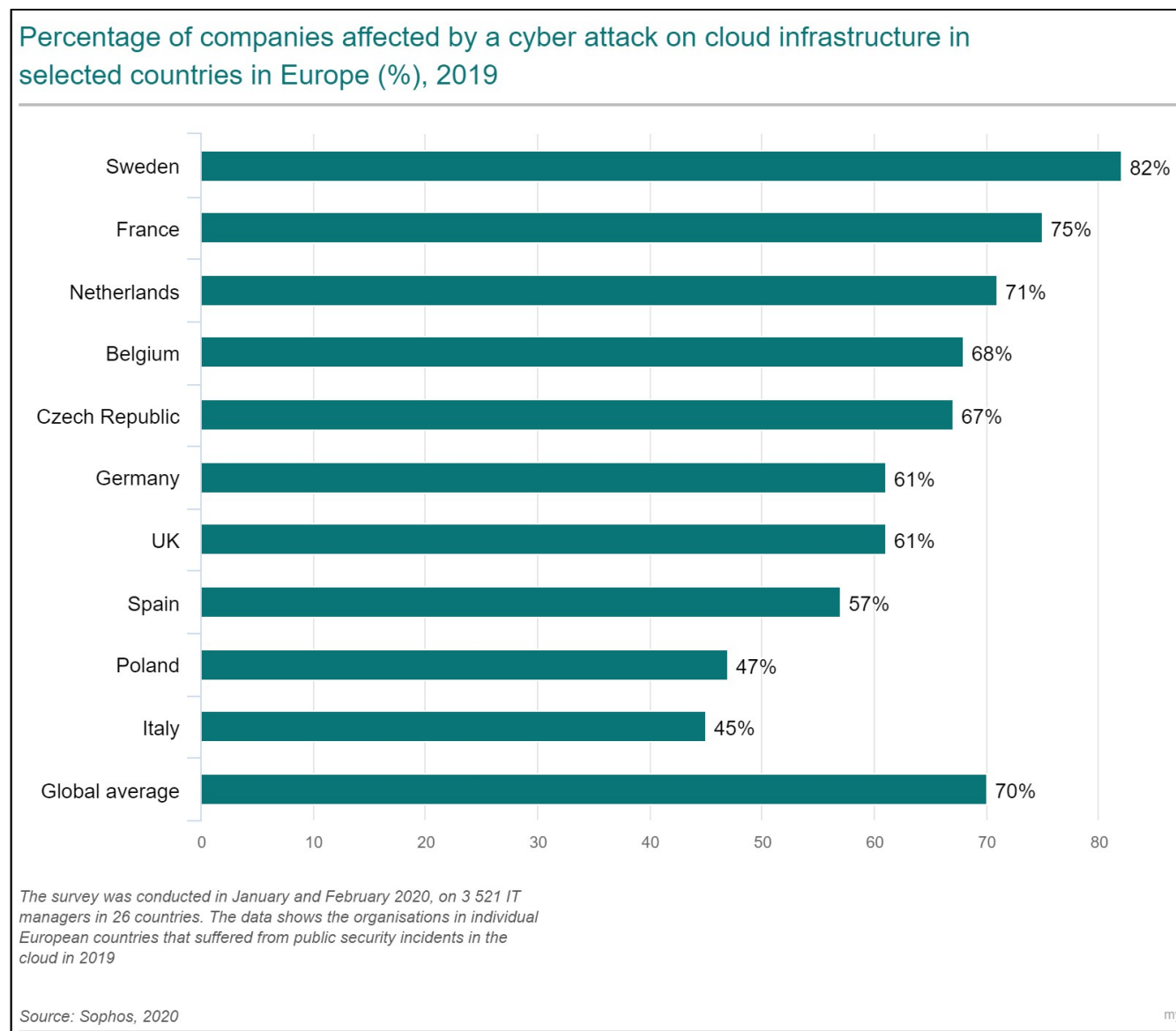


Рисунок 1.4 – Відсоток хмарних інфраструктур, що зазнавали атак за країнами ЄС. 2019 рік. [18]

Серед найпоширеніших векторів атак на інфраструктури були саме ті вектори, що є особливо критичними для користувацьких інфраструктур у хмарі, а саме: неправильне налаштування сервісів ХС (68%), погано розподілений доступ до ресурсів ХС (58%), компрометація акаунтів ХС (50%), погане розмежування даних в ХС, що зумовило витік даних (43%).

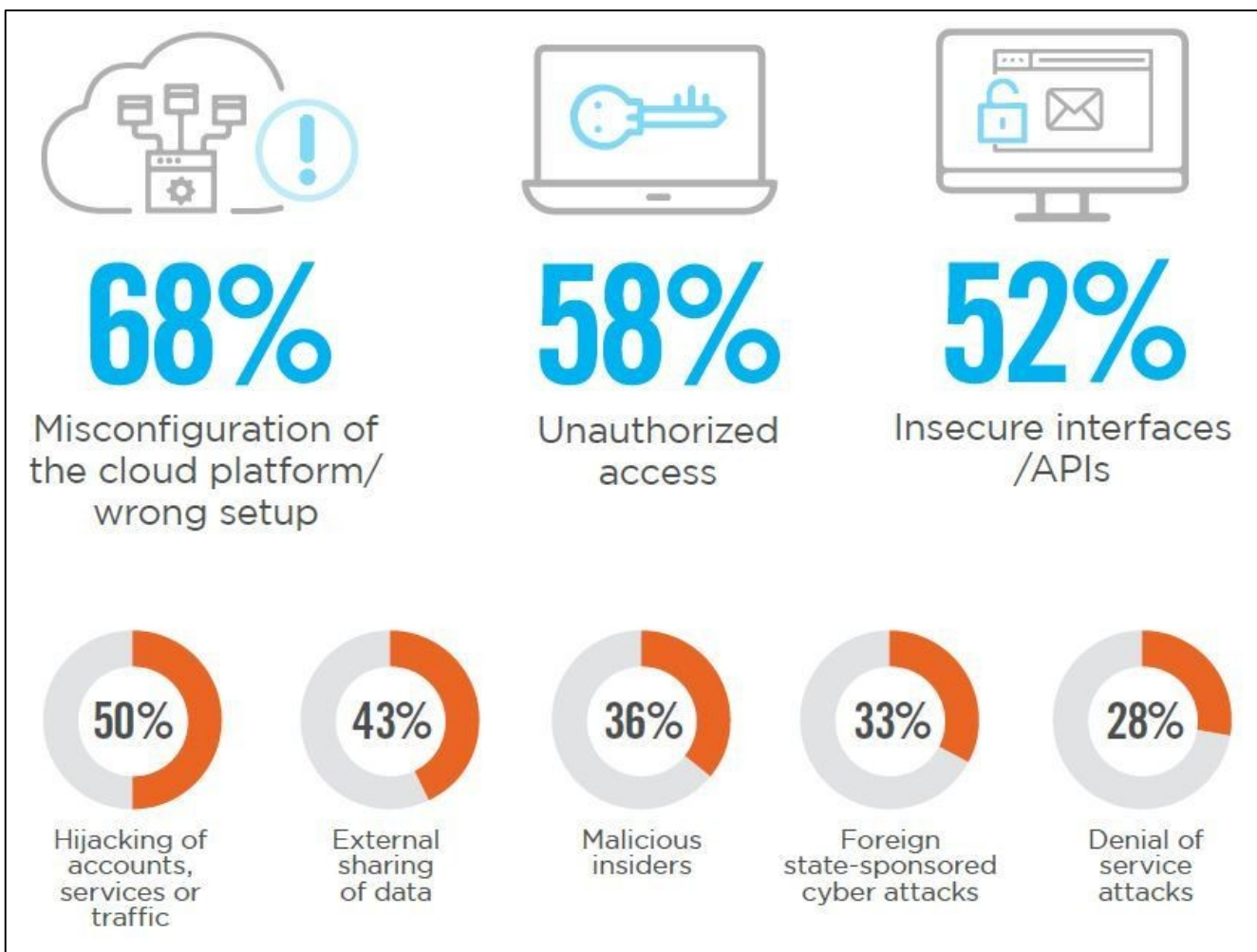


Рисунок 1.5 – Найбільш поширені вектори атак на хмарні інфраструктури. [19]

Згідно з дослідженням Centrify [20], 90% успішних атак на користувацькі інфраструктури у ХС дозволяли зловмиснику отримати облікові дані користувача ХС з привілейованим доступом до ресурсів, що, власне, гарантувало йому повний контроль над інфраструктурою.

Більш того, 89% успішних атак на інфраструктуру не ідентифікуються користувачами ХС вчасно [21], що дає зловмисникам додатковий час зловмисникам на закріплення в системі, збір користувацьких даних та їх ексфільтрацію.

Отже, одними з основних проблем безпеки в хмарних середовищах постають недоліки існуючих систем автентифікації, авторизації, аудиту подій та їх конфігурація.

1.3 Принципи безпеки

Для вирішення даних проблем, запропонована модель має попереджувати неавторизований доступ до хмарних ресурсів і даних з метою їх зчитування, модифікації, або видалення, таким чином забезпечуючи наступні базові принципи безпеки [6]:

- Конфіденційність – лише авторизовані користувачі мають мати доступ до зчитування даних з метою попередження їх витоку.
- Цілісність – лише авторизовані користувачі мають мати доступ до модифікації даних з метою попередження їх спотворення.
- Доступність – дані та сервіси мають бути доступні постійно. Лише авторизовані користувачі матимуть можливість змінювати доступність об'єктів системи.
- Спостережливість – модель має вести журнали аудиту усіх подій, що відбуваються у системі та сповіщати користувача у разі порушення політик безпеки. Це інформація необхідна для встановлення усіх дій зломисника у системі в разі її компрометації.

1.4 Компоненти моделі безпеки в хмарному середовищі

Користуючись існуючими концепціями публічних ХС [7], під які буде розроблено модель, можна виділити окремі типи суб'єктів та об'єктів системи, яких необхідно врахувати при розробці.

1.4.1 Суб'єкти в хмарному середовищі

Суб'єктом в ХС є будь-яка сутність, що здатна виконувати дії над іншими сутностями. Суб'єкти поділяються на 3 типи:

- Системний акаунт. Системний акаунт в хмарному середовищі є кореневою точкою ХС окремого користувача [8]. Саме до нього прив'язані всі ресурси в середовища.
- Акаунт користувача. Акаунти користувачів використовуються ними на повсякденній основі для доступу до ресурсів.
- Сервісний акаунт. Сервісні акаунти використовуються налаштованими користувачем сервісами хмарного середовища для доступу до ресурсів в хмарі.

Системний акаунт має найвищі привілеї та здатен обходити встановлені політики безпеки. Власне, він є вбудованим акаунтом адміністратора системи. Користувацьким та системним акаунтам можуть бути надані привілеї та додаткові права для здійснення визначених дій в системі.

1.4.2 Об'єкти в хмарному середовищі

Об'єктом в ХС є будь-яка сутність, над якою можна виконувати дію. Суб'єкти також відіграють роль об'єктів при виконанні дій над ними [9]. Таким чином, об'єктом в хмарному середовищі є:

- Обчислювальні ресурси, що знаходяться в тимчасовому володінні користувача
 - Віртуальні ОС
 - Накопичувачі даних
 - Мережеві ресурси
- Програмне забезпечення та сервіси, що надаються в тимчасове володіння користувачеві
- Суб'єкти створені користувачем

1.5 Проблема розмежування хмарних середовищ

Позаяк всі обчислення окремих користувачів здійснюються на поділюваних ресурсах, вкрай необхідно за замовчуванням ізолювати окремі хмарні ресурси, що належать різним користувачам. Більш того, необхідно групувати окремі ресурси, що належать одному користувачеві для створення віртуальної інфраструктури.

Для однозначної ідентифікації ресурсу в системі, надавачі послуг ХС використовують глобальні імена [10]. Позаяк кожна інфраструктура має в собі один системний акаунт, його ідентифікатор і використовується для прив'язки ресурсу до інфраструктури [8].

Усі ресурси в хмарному середовищі є публічно видимими за їх глобальним іменем. Тому, необхідно впровадити відповідні механізми захисту для попередження атак зловмисників на підконтрольні користувачеві ресурси.

1.6 Проблема централізованого керування доступом

Позаяк системні ресурси окремо взятої інфраструктури користувача можуть знаходитися в різних підмережах та регіонах ХС та не можуть бути згруповані фізично, гостро постає проблема організації централізованого керування доступом до ресурсів хмари, що має бути окремою підсистемою ХС та реалізувати основні функції управління доступом, а саме: автентифікацію, авторизацію користувачів і аудит усіх подій в хмарі. Будь-яка дія суб'єкта має насамперед бути погодженою підсистемою керування доступу до ресурсів.

1.6.1 Вимоги до автентифікації суб'єктів в системі

Автентифікація має бути уніфікованою для всіх типів суб'єктів ХС [12]. Позаяк суб'єктам надаватимуться права доступу до глобальних ресурсів з міткою певного користувача, процедуру автентифікації має здійснювати надавач послуг ХС. Таким чином, автентифікація відбуватиметься по мережі. Підсистема автентифікації має однозначно визначати суб'єкт хмарного середовища та підтверджувати його за допомогою перевірки секрету, наданого суб'єктом. Процедура передачі секрету має бути стійкою від атак, що спрямовані на перехоплення мережових пакетів та їх інспекцію, або повторне надсилання підсистемі (replay attack). Підсистема автентифікації має попереджувати можливі атаки перебору секретів суб'єкта.

1.6.2 Вимоги до авторизації суб'єктів в системі

Кожна дія суб'єкта в системі спершу має бути погоджена підсистемою авторизації. Підсистема має за замовчуванням забороняти доступ до будь-яких

ресурсів суб'єктам. Кожному суб'єкту потрібно ставити у відповідність список контролю доступу, що описує його права на об'єкти системи. Має існувати чіткий алгоритм прийняття рішень для уникнення конфліктних ситуацій при обрахунку прав доступу.

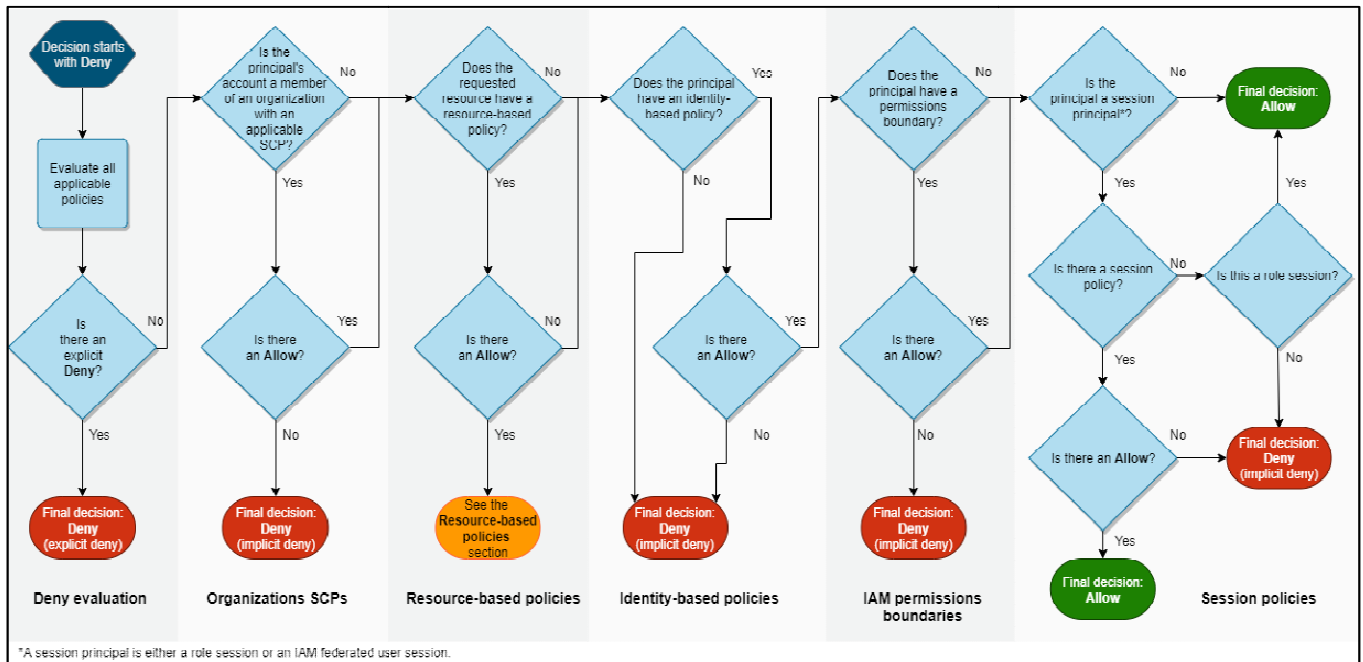


Рисунок 1.6 – Поведінка підсистеми авторизації AWS при обрахуванні прав доступу [13]

1.6.3 Вимоги до аудиту подій в системі

Підсистема аудиту має нотувати будь-яку подію, пов'язану з підсистемами авторизації та автентифікації. Записи мають містити необхідну інформацію, щоб однозначно ідентифікувати суб'єкта, об'єкта, дію, та відповідні метрики, як час та мережеву адресу, тощо. Доступ до підсистеми аудиту має бути обмежений та контролюватися підсистемами автентифікації та авторизації. Журнали аудиту

мають знаходитися за межею інфраструктури користувача для уникнення їх модифікації.

1.6.4 Проблема керування суб'єктами і об'єктами в системі

Усі суб'єкти та об'єкти системи мають бути синхронізовані з підсистемами авторизації та автентифікації. При видаленні окремого об'єкту, суб'єкту, або ж інфраструктури користувача, система має забезпечити повне видалення відповідних списків контролю доступу для упередження можливого перехоплення контролю над видаленими ресурсами іншим користувачем.

1.6.5 Проблема сумісності між хмарними середовищами

Між хмарними середовищами на даний момент не існує єдиного стандарту по найменуванню послуг та уніфікованому розподілу доступу до ресурсів. Так, можна розглянути формат політик безпеки найбільших ХС – GCP, AWS, Azure, що надають аналогічні права суб'єкту (п.р. отримання всіх ролей в системі):

Опис ролі в Azure Cloud:

```
{
  "properties": {
    "roleName": "Azure-Custom-Role",
    "description": "Azure custom role to list IAM roles",
    "assignableScopes": [
      "/subscriptions/11111111-1111-1111-1111-111111111111"
    ],
    "permissions": [
      {
```

```

        "actions": [
            "Microsoft.Insights/ActionGroups/Read"
        ],
        "notActions": [],
        "dataActions": [],
        "notDataActions": []
    }
}

```

Опис ролі в Google Cloud Platform:

```

description: Google Cloud Platform custom role to list IAM roles
etag: BwVkBX0sQD0=
includedPermissions:
- iam.roles.list
name: organizations/111111111111/roles/GPCustomRole
stage: ALPHA
title: GCP Custom Role

```

Опис ролі в Amazon Web Services:

```

{
  "RoleId": "AROAJ520TH4H7LEXAMPLE",
  "RoleName": "AWSCustomRole",
  "Arn": "arn:aws:iam::123456789012:role/AWSCustomRole",
  "Description": "AWS custom role to list IAM roles",
  "AttachedPolicies": [
    {
      "PolicyName": "AWSCustomPolicy",
      "PolicyArn": "arn:aws:iam::123456789012:policy/AWSCustomPolicy",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "iam:ListRoles",
          "Resource": "*"
        }
      ]
    }
  ]
}

```

Хоча формат ролей (JSON – AWS, Azure; YAML – GCP), спосіб надання прав (через роль напряму – Azure, GCP, через прикріплені політики – AWS) та

найменування об'єктів кардинально відрізняються, можна виокремити спільні риси політик безпеки – **унікальний ідентифікатор, прив'язаний до інфраструктури користувача, набір прав доступу, право доступу.**

Система управління доступом має бути незалежною від апаратної платформи, на якій вона розгорнута, та бути сумісною з наявними хмарними середовищами. Структура системи має закладати основу програмного інтерфейсу, що дозволить користувачам інтегрувати власне ПЗ для гнучкого налаштування.

1.7 Моделі автентифікації

Серед можливих моделей автентифікації суб'єктів у ХС можна розглянути наступні:

- Автентифікація за обліковими даними (логін + пароль). Традиційний метод автентифікації у багатьох інформаційних системах. Суб'єкт посилає системі свій логін та пароль, на основі яких система приймає рішення про автентифікацію. Вимагає існування сильної паролльної політики та додаткових механізмів безпеки для унеможливлення перебору паролів. Логін та пароль мають передаватися по захищеному каналу для попередження MitM атак. Добре підходить для автентифікації звичайних користувачів, однак не є практичним для автентифікації сервісних акаунтів ХС, адже вимагає додаткового керування унікальними для кожного суб'єкта обліковими даними.
- Автентифікація за спільним секретом (токеном) [23]. Автентифікація на основі поділюваного з сервером криптографічно стійкого випадкового значення достатньої довжини, що автоматично нівелює ризики від атак перебору секретів. Має передаватися по захищеному каналу для

попередження MitM атак. Підходить як для автентифікації звичайних користувачів, так і для сервісних акаунтів ХС, однак вимагає строгі умови зберігання секрету як для серверу, так і для суб'єкту для попередження його компрометації.

- Автентифікація за допомогою PKI [24]. Автентифікація на основі асиметричної криптографії. Підсистема автентифікації генерує пару публічний ключ – приватний ключ для кожного суб'єкта в системі. Приватний ключ зберігається виключно у суб'єкта та є секретом. Публічний ключ зберігається на сервері для перевірки даних, наданих суб'єктом. Для автентифікації під час передачі повідомлення, суб'єкт, користуючись властивостями асиметричної криптографії, генерує підпис для повідомлення за допомогою приватного ключа [26]. На стороні ХС, сервер використовує публічний ключ для перевірки підпису. Підходить як для автентифікації звичайних користувачів, так і для сервісних акаунтів ХС. Вимагає строгі умови зберігання приватного ключа на суб'єкті. Завдяки особливостям PKI, автентифікація може відбуватися і по незахищеному каналу передачі даних, а підпис автоматично гарантуватиме цілісність повідомлення.

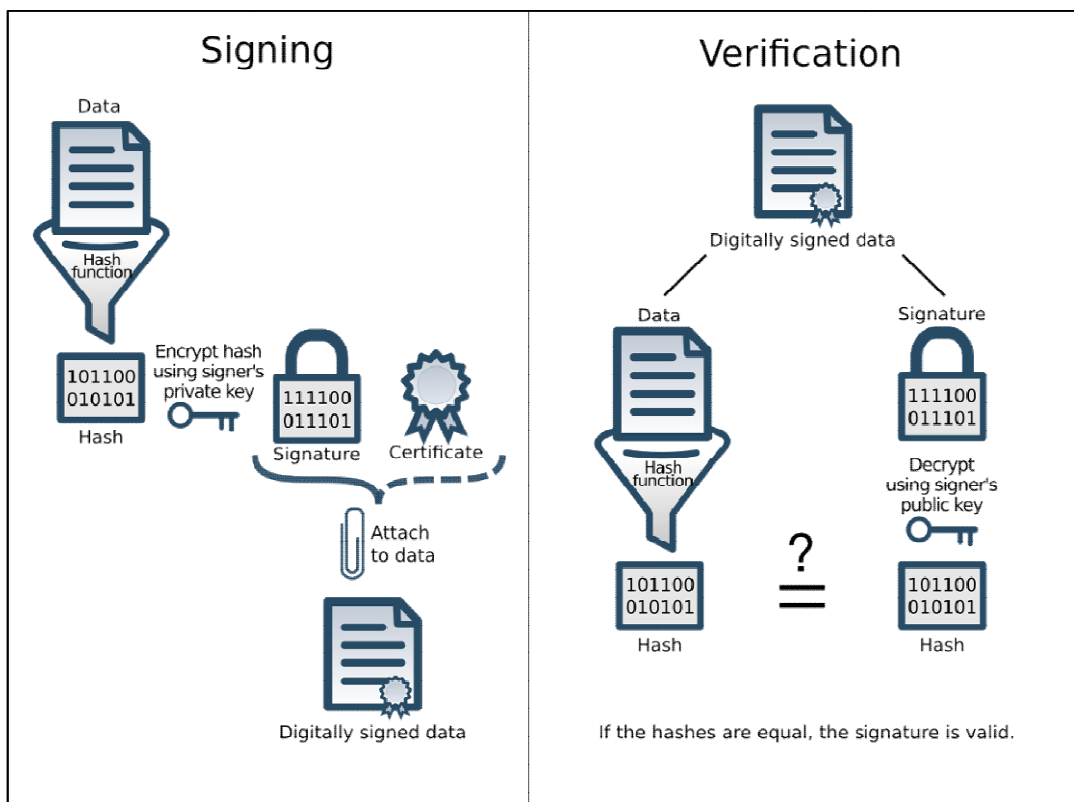


Рисунок 1.7 – Схема автентифікації у PKI [25]

1.8 Моделі розмежування прав доступу

Традиційними моделями розмежування прав доступу є дискреційна модель доступу (Discretionary Access Control), рольова модель доступу (Role-Based Access Control) та мандатна модель доступу (Mandatory Access Control).

1.8.1 DAC

Дискреційна модель доступу побудована на окремих списках контролю доступу (Access Control List) та записах в них (Access Control Entry). Запис містить ідентифікатор суб'єкта, ідентифікатор об'єкта та дії, що йому дозволені. Саме набір записів в списку контролю доступу визначає повні права суб'єкта. Система

має підтримувати списки контролю доступу для кожного суб'єкта в системі, щоб обраховувати його права при діях над об'єктами. Складність налаштування DAC стрімко зростає зі збільшенням об'єктів системи, тому її використання в ХС не є практичним. [27]

1.8.2 RBAC

В даній моделі права суб'єкта визначаються набором ролей, що були йому надані адміністратором. Такий підхід до розподілу прав доступу є набагато більш гнучким, ніж DAC, дозволяє ефективно адмініструвати велику кількість ОР в ХС та легко масштабується. Саме ця модель розподілу доступу активно використовується одними з найбільш популярних надавачами послуг ХС: GCP [29], Azure [28]. AWS натомість використовує надбудову над RBAC, атрибутну модель доступу (ABAC) [30].

1.8.3 ABAC

Атрибутна модель доступу є надбудовою над рольовою моделлю доступу та, замість розподілу доступу до самих ресурсів безпосередньо, надає керування над мітками. Мітки можна застосовувати до багатьох об'єктів одночасно, що спрощує написання нових політик доступу при розширенні інфраструктури.

1.8.4 MAC

Мандатна модель доступу призначає ефективні права через визначення рівнів конфіденційності об'єктів та надання відповідних рівнів допуску суб'єктам. Суб'єкти можуть модифікувати лише об'єкти свого рівня допуску. Читання дозволяється об'єктів свого рівня допуску або нижче. Модифікація об'єктів з нижчим рівнем допуску забороняється для попередження витоку інформації до суб'єктів з нижчим рівнем допуску [31]. Підхід до розподілу прав доступу мандатної моделі не є ефективним у хмарних середовищах, адже рівні доступу не надають достатньої гнучкості у налаштування у великих інфраструктурах.

Висновки за розділом 1

В розділі визначено поняття хмарного середовища та його види. Розглянуто динаміку розвитку хмарних обчислень і проаналізовано динаміку атак на хмарні інфраструктури.

Визначено основні компоненти системи (суб'єкти, об'єкти). Висунуті вимоги до компонентів моделі безпеки, а саме, підсистем автентифікації, авторизації та аудиту подій. Дані системи мають контролювати будь-яку дію в хмарному середовищі, та: прозоро, за визначеним алгоритмом дій, ідентифікувати суб'єктів і їх права, приймати рішення щодо погодження або блокування дії, надавати достатню кількість метаданих для спостережливості подій в системі.

Розглянуто традиційні моделі для підсистем автентифікації та авторизації в контексті роботи у хмарному середовищі, визначено сильні та слабкі сторони моделей.

Виокремлено проблеми безпеки, що постають при використанні обчислювальних ресурсів з поділюваним доступом. Серед основних проблем визначено ізоляцію користувачів один від одного при використанні ресурсів в ХС, організацію ефективного контролю доступу до ресурсів та узгодження моделі безпеки для усіх наявних видів ХС.

Для вирішення визначених проблем у хмарних середовищах запропоновано розробити модель безпеки, що враховує усі висунуті в першому розділі вимоги та реалізовуватиме додатковий рівень безпеки на базі існуючих в хмарних середовищах інструментів.

2 АГЕНТНА МОДЕЛЬ БЕЗПЕКИ ХМАРНИХ СЕРЕДОВИЩ

2.1 Класична модель безпеки хмарного середовища

Перед побудовою власної моделі безпеки ХС слід систематизувати об'єкти хмарного середовища та їх функціонал. Для цього необхідно розглянути класичну модель безпеки хмарного середовища на прикладі діаграм класів та послідовностей. Серед окремих класів ХС слід виділити:

- Cloud Environment (Хмарне середовище) – клас, що уособлює собою хмарну інфраструктуру окремого користувача. Містить глобальний унікальний ідентифікатор ХС **cloud_id**, усі асоційовані об'єкти в інфраструктурі **entities** та дозволяє виконувати прямі операції над об'єктами (додавання **add_entity**, видалення **delete_entity**, читання

get_entities) а також особливий об'єкт **IAM service**, що виконує роль системи контролю доступу в ХС.

- Access Policy (Політика безпеки) – абстрактний клас, що становить базові вимоги безпеки для будь-якої моделі розподілу доступу. Містить перелік прав доступу суб'єкту на об'єкт **policy_statement** та унікальний ідентифікатор політики у системі **policy_id**.
- Cloud Access Policy (Політика безпеки ХС) – імплементація політики безпеки для хмарного середовища. Є частиною кожного об'єкта системи. Містить в собі конкретні ACE **policy_statement**, що дозволяють, або забороняють певні дії над об'єктом для зазначених в ACE суб'єктів.
- Entity (Об'єкт) – абстрактний клас об'єкта ХС. Містить унікальний ідентифікатор об'єкту у системі **object_id** та асоційовану з ним політику безпеки ХС **access_pol**.
- Account (Акаунт) – об'єкт ХС, що характеризує акаунт (акаунт користувача, системний акаунт, сервісний, тощо.), який виступатиме в ролі суб'єкта системи. У кожного акаунту є логін (**username**) та секрет (**secret**) для автентифікації в системі.
- Data (Дані) – об'єкт ХС, що характеризує дані користувача в ХС. Містить в собі відповідні користувацькі дані **user_data**. Дані можуть бути модифіковані суб'єктами системи за допомогою методу **modify_user_data**.
- Service (Сервіс) – об'єкт ХС, що характеризує додатковий функціонал, що пропонує надавач послуг хмарного середовища, наприклад, використання віртуальних машин для обчислень, тощо. Надає віртуальний метод **invoke_service_functionality**, що індивідуально імплементується дочірніми класами сервісу.

- IAM Service (IAM сервіс) – особливий тип сервісу в ХС. Саме цей сервіс відповідає за реалізацію підсистем авторизації (**authorize(Account, Entity, action)**) і автентифікації (**authenticate(Account)**) в системі. Містить в собі всі акаунти системи **accounts**. Перевіряє кожну дію суб'єкта системи відносно відповідної політики безпеки ХС об'єкта та забороняє неавторизовані дії. Дозволяє модифікувати акаунти (**modify_user(Account)**, **delete_user(Account)**, **add_user(Account)**, **get_users()**) та політики безпеки хмари (**modify_access_policy(uuid, statement)**, **add_access_policy(uuid, statement)**, **delete_access_policy(uuid)**, **get_access_policies()**).
- Management Console (Панель керування) – клас, що уособлює користувача хмарного середовища, що виконуватиме дії **action(Account, Entity, action_name)** над хмарними ресурсами. Містить в собі логін **username** та пароль **password** для автентифікації **authenticate(username, password)** в системі а також відповідний акаунт **account**, що буде надано ХС після автентифікації.

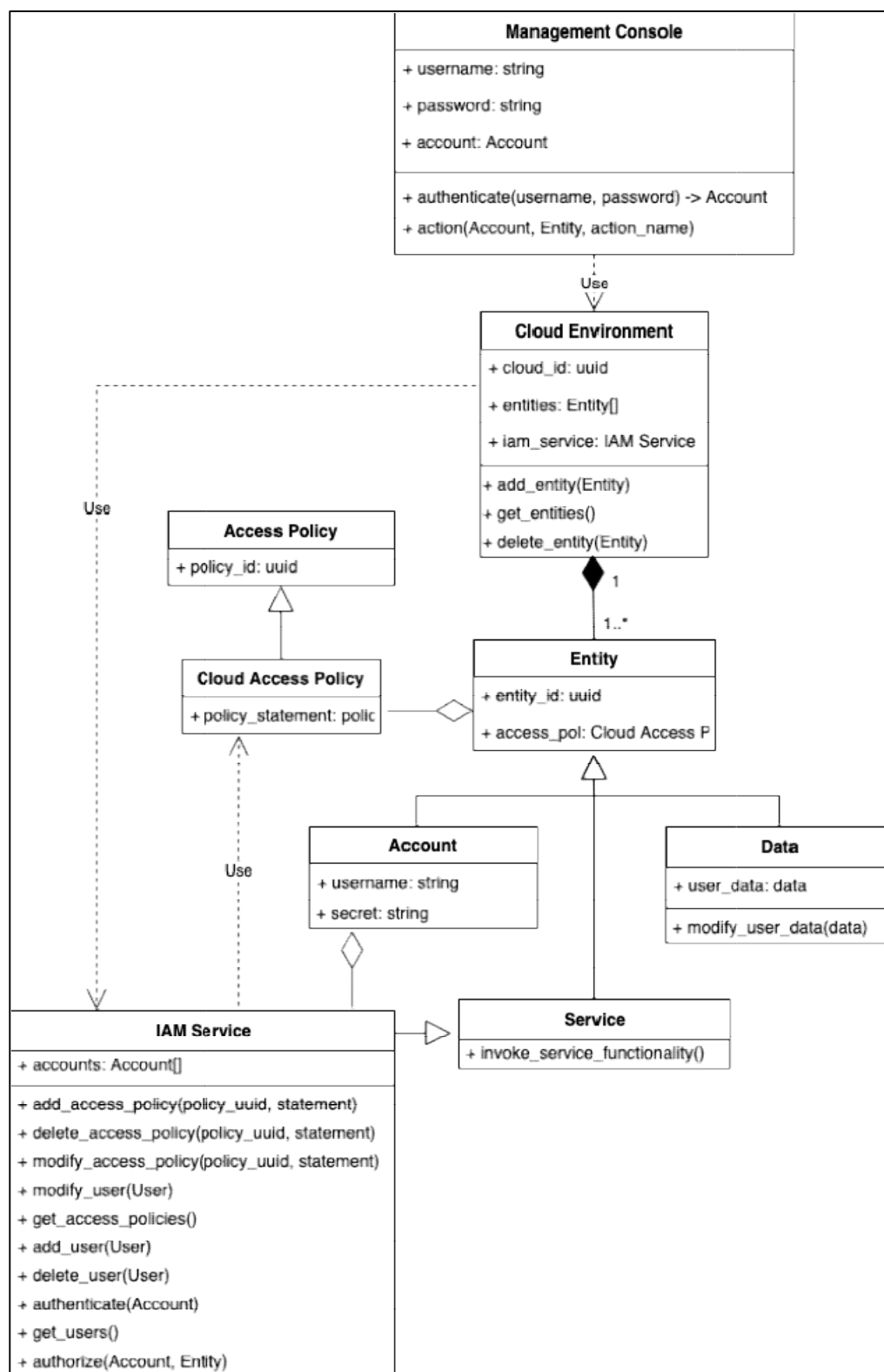


Рисунок 2.1 – Діаграма класів класичної моделі безпеки хмарного середовища

У наступних діаграмах розглянуто послідовність взаємодії класів при використанні панелі управління для читання об'єктів ХС та модифікації даних користувача в межах:

1. Одного хмарного середовища. В межах одного ХС класична модель відпрацьовує бездоганно: функціоналом AAA займається IAM сервіс, що коректно перевіряє облікові дані користувача, видає йому відповідний акаунт ХС та перевіряє його права при кожній дії в системі.

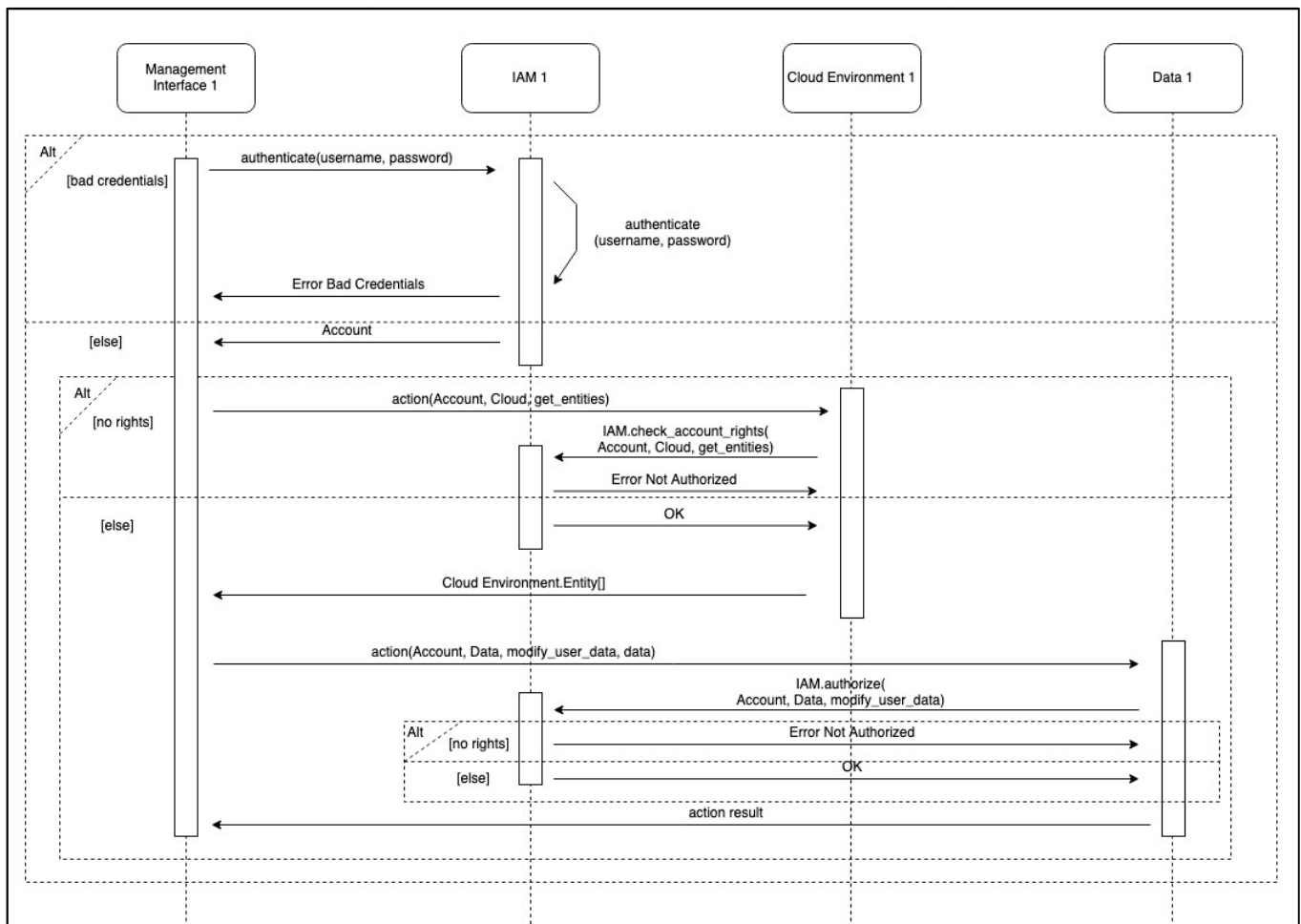


Рисунок 2.2 – Діаграма послідовностей для класичної моделі безпеки в межах одного ХС

2. Декількох хмарних середовищ. При використанні традиційної моделі безпеки між декількома хмарними середовищами виникає фундаментальний недолік – користувач з акаунтом ХС1 ніяким чином не

може взаємодіяти з ресурсами XC2, бо IAM сервіс та відповідні політики безпеки у XC2 не містять жодних записів стосовно акаунта XC1.

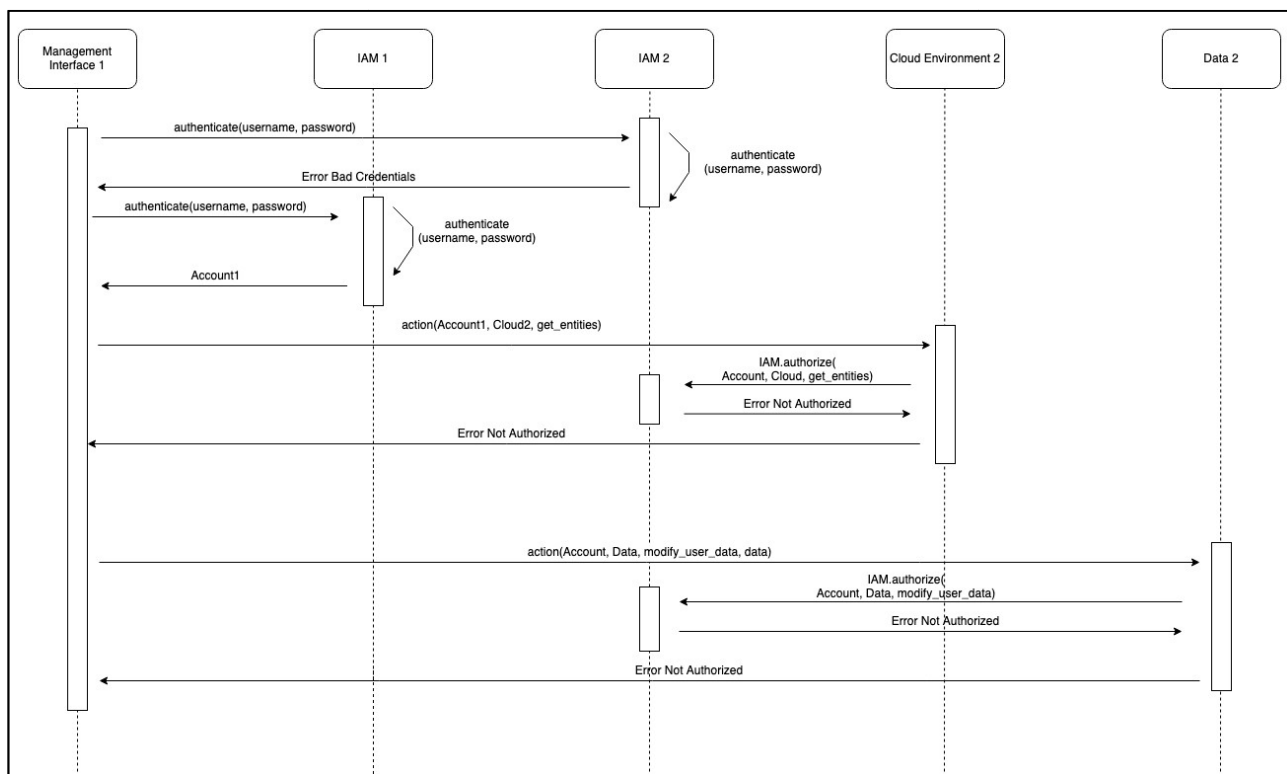


Рисунок 2.3 – Діаграма послідовностей для класичної моделі безпеки між декількома XC

2.1.1 Політики прав доступу в класичній моделі безпеки

На прикладі вище зазначених діаграм послідовностей можна добре проаналізувати механізм роботи політик прав доступу хмарного середовища в класичній моделі безпеки. Кожна політика прав доступу містить ідентифікатор об'єкту XC, до якого вона прикріплена та список контролю доступу, що описує права суб'єктів XC на даний об'єкт. При доступі до об'єкту, IAM сервіс отримує відповідний список контролю доступу для цільового об'єкту та перевіряє чи у суб'єкта є права на цю дію згідно з визначеним алгоритмом (див. рис. 1.6). Даний підхід до організації прав доступу є доволі гнучким та, за допомогою певної

абстракції, дозволяє будувати найбільш поширені моделі доступу: DAC, RBAC, MAC, тощо. Однак, ця модель є обмеженою одним хмарним середовищем: списки контролю доступу традиційних політик знають лише про суб'єкти, що існують безпосередньо в їх ХС та не дозволяють використовувати суб'єкти з інших середовищ.

2.2 Агентна модель безпеки

Для вирішення проблеми сумісності декількох ХС, а також проблем, поставлених у розділі 1, запропоновано використовувати агентну модель безпеки. Як приклад застосування такого підходу надано в [32].

Ця модель побудована на автономних програмах (агентах), що інкапсулюють весь функціонал підсистем автентифікації, авторизації та аудиту подій в собі й надають єдиний інтерфейс для всіх типів хмарних середовищ. Даний підхід вирішить проблему узгодження між декількома ХС і дозволить легке масштабування управління над новими ресурсами – необхідно лише встановити агент на нові ресурси.

Для централізованого керування доступом необхідно визначити 2 типи агентів:

- Центральний агент (ЦА). Даний тип агенту реєструє всі наявні в інфраструктурі агенти та відповідає за AAA:
 1. Автентифікацію
 2. Авторизацію
 3. Аудит подій

Саме в ЦА зберігаються всі визначені політики доступу та приймається рішення про авторизацію дій. ЦА має як керувати AAA в межах одного

ХС, так і надавати можливість взаємодії між ХС декількох користувачів. ЦА інтегруватиметься в цільові ХС та після успішної авторизації МА має йому надавати специфічний для ХС сесійний токен через Identity and Access Management (IAM) сервіс ХС, таким чином інкапсулюючи взаємодію з самим ХС.

- Мобільний агент (МА). Даний тип агенту використовується для інкапсуляції ААА для окремого суб'єкта системи – користувача, системного сервісу, тощо. МА авторизується відносно ЦА, отримує сесійний токен ЦА з визначеними для нього привілеями, та використовує його для здійснення дій в системі. При здійсненні дій в ХС, МА звертається до ЦА, і, якщо успішно проходить процедуру авторизації дії, отримує сесійний токен ХС для безпосереднього її виконання.

Взаємодія між агентами для виконання дії над об'єктом ХС є надбудовою над існуючим IAM сервісом, що присутній у більшості публічних та приватних реалізацій хмарних обчислень. Хоча це не дає додаткових переваг, окрім уніфікованого керування суб'єктами, при використанні агентів в одному ХС, даний підхід дозволяє легко організовувати єдине керування доступом між багатьма окремими ХС без надлишкової програмної логіки.

До діаграми класів класичної моделі безпеки додамо наступні класи:

- Agent(Агент) – абстрактний клас агенту. Містить унікальний ідентифікатор агенту **agent_id**, секрет для автентифікації **secret**, та відповідний акаунт ХС **iam_account**, асоційований з агентом. Оголошує віртуальний метод **authenticate(secret, agent)**, що буде використовуватись агентами для взаємної автентифікації.
- Mobile Agent(Мобільний Агент) – агент, що використовуватиметься суб'єктами системи для доступу до об'єктів ХС. Містить посилання на

асоційований з ним центральний агент **central_agent**. Проходить процес автентифікації та авторизації на ЦА. Після успішної авторизації та автентифікації отримує відповідний акаунт **iam_account** та використовує його для здійснення безпосередніх дій в хмарному середовищі.

- Central Agent(Центральний Агент) – агент, що використовується хмарним середовищем для організації доступу до ресурсів. Функціонує поверх існуючого IAM сервісу наступним чином: ЦА містить в собі всі прив'язані мобільні агенти **mobile_agents** та відповідні для них політики **agent_policies**. При будь-якій дії МА в системі, ЦА перевіряє права агенту **authorize(Agent,Entity, action)**, та надає йому відповідний до визначених політик доступу IAM акаунт **grant_iam_account(Agent)**. Також, дозволяє виконувати відповідні до стандартного функціоналу IAM дії з конфігурації агентів (**add_agent, delete_agent, modify_agent, get_agent**), їх політик доступу (**add_agent_policy, delete_agent_policy, modify_agent_policy, get_agent_policies**) та імплементує механізм аудиту подій (**add_accounting_event(data), get_accounting_events()**).
- Agent Access Policy(Агентна політика доступу) – політика доступу, що містить співвідношення Агент **agent** – хмарні політики доступу **cloud access policies**, що дозволяє гнучко налаштовувати права агентів за допомогою існуючих політик доступу ХС.

Отже, агенти використовують особі IAM сервіс як готову основу для імплементації основного функціоналу підсистем ААА та не привносять додаткових складнощів в архітектуру хмари.

Окрім того, клас Cloud Environment тепер використовує об'єкт класу Central Agent як основу підсистем ААА хмарного середовища замість IAM.

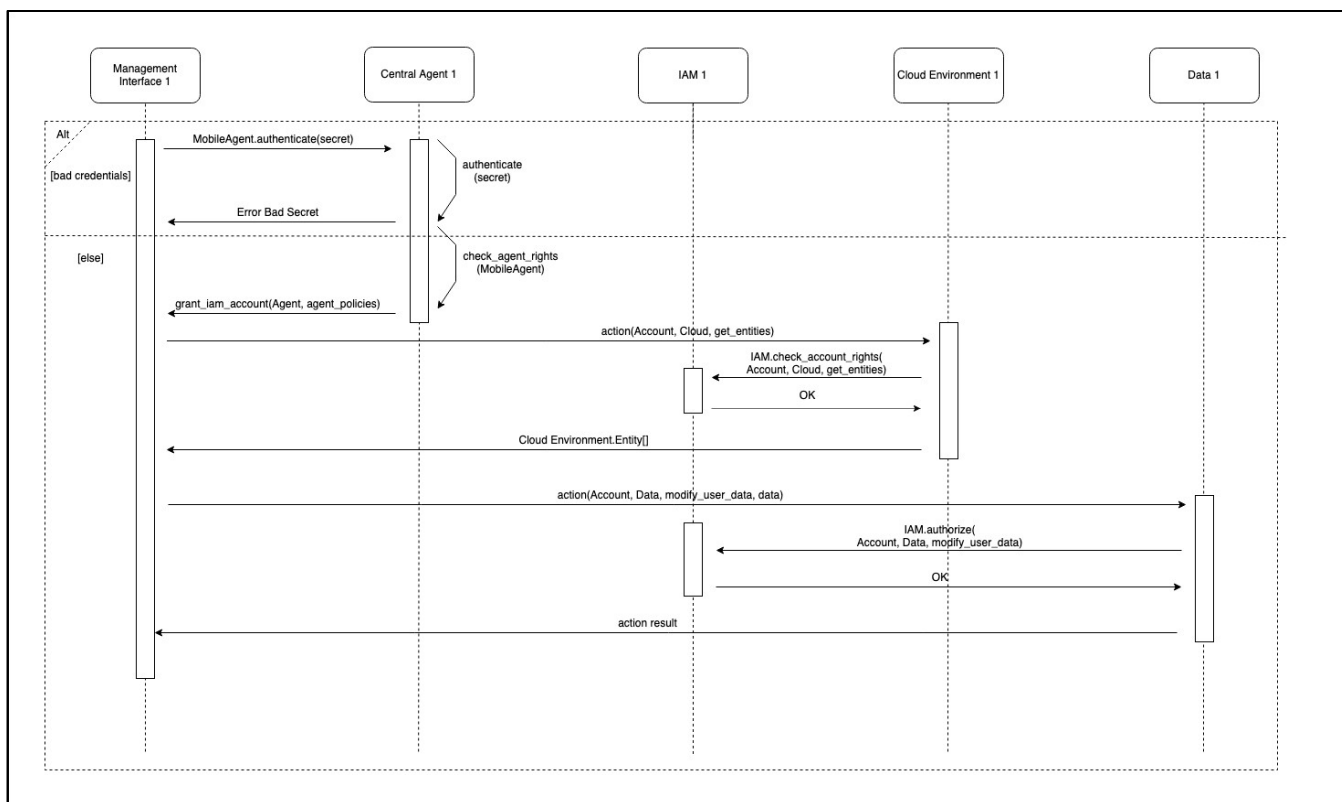


Рисунок 2.5 – Діаграма послідовностей для агентної моделі безпеки в межах одного XS

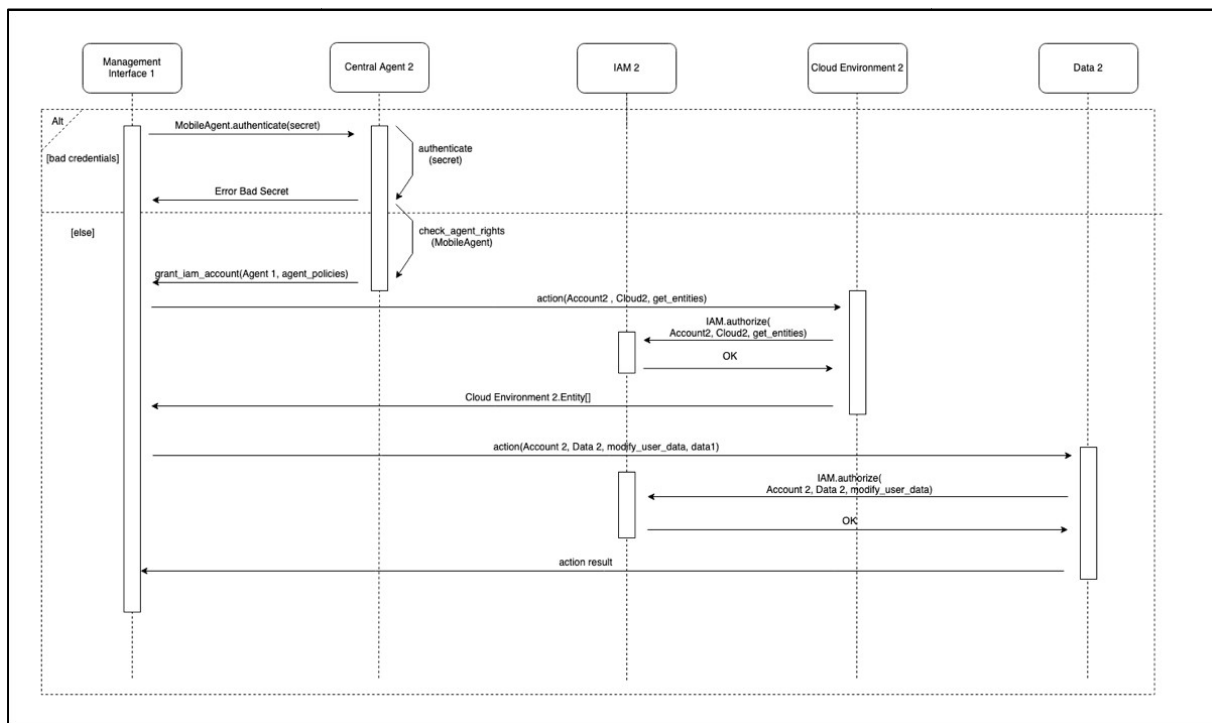


Рисунок 2.6 – Діаграма послідовностей для агентної моделі безпеки між декількома XS

2.3 Запропонована модель автентифікації

В якості моделі автентифікації запропоновано використовувати автентифікацію на базі асиметричної криптографії. Даний підхід має декілька переваг:

- В центральному агенті зберігається лише публічна інформація щодо зареєстрованих агентів (їх відповідні публічні ключі). Таким чином, не можливо скомпрометувати секрети агентів безпосередньою атакою на ЦА.
- Асиметрична криптографія дозволяє вільно використовувати протокол автентифікації по незахищеним каналам передачі інформації без ризику витоку секретів.
- За допомогою асиметричної криптографії можна реалізувати так звану автентифікацію без відстеження стану (stateless authentication) [33] – сервер не буде використовувати механізм сесії для відстеження стану клієнта, а автентифікація буде відбуватися щоразу при обробці нового запиту. Цей механізм дозволить звільнити зайві ресурси ОЗП, що зазвичай використовуються для збереження активних сесій, а також принесе додатковий рівень безпеки у алгоритм, прозоро перевіряючи усі необхідні атрибути клієнта при кожному зверненні до сервера.
- Асиметрична криптографія використовує випадково генеровані секрети набагато більшої довжини ніж звичайні паролі та токени, що робить атаки перебору непрактичними.

2.3.1 Структура підсистеми автентифікації

В підсистемі автентифікації на основі асиметричної криптографії грають роль дві сторони, ЦА та МА. Сторони автентифікуватимуть запити та відповіді один одного для попередження можливих атак на канал передачі даних:

- ЦА. Центральний агент містить масив агентів **A**, яким дозволено автентифіковуватися відносно ЦА. У кожного агента **a** є **A** існує унікальний ідентифікатор **agent_id** та публічний ключ **pubkey**. Більш того, ЦА має власні приватний ключ **privkey** та публічний ключ **pubkey**, що використовуватимуться МА для взаємної автентифікації відповідей від ЦА.
- МА. Мобільний агент містить власний унікальний ідентифікатор **agent_id** та приватний ключ **privkey**, що використовується для автентифікації відносно інших ЦА.

2.3.2 Алгоритм автентифікації для сторони, що генерує запит

Підсистема автентифікації генеруватиме для кожного пакету агенту відповідний підпис за допомогою функції асиметричної криптографії **sign(privkey, hash_sum)**, де **privkey** – приватний ключ агента, а **hash_sum** – продукт односторонньої криптографічної хеш функції [34] **hash(body)**, що ставить в однозначну відповідність тілу пакета **body** набір байтів, забезпечуючи цілісність та дозволяючи виявити будь-які зміни **body**.

Тіло пакету агенту складатиметься з:

- **agent_id** – унікального ідентифікатора агенту в системі, що використовуватиметься ЦА для отримання публічного ключа.
- **action(arguments...)** – безпосередньо дії, що потребує автентифікації, та необхідних дії аргументів
- **timestamp** – мітки часу, коли дію було здійснено для уникнення перевикористання перехоплених пакетів автентифікації .

Таким чином, готовий пакет агенту складається з:

1. **body**
 - a. **agent_id**
 - b. **action(arguments...)**
 - c. **timestamp**
2. **signature(privkey, hash(body))**

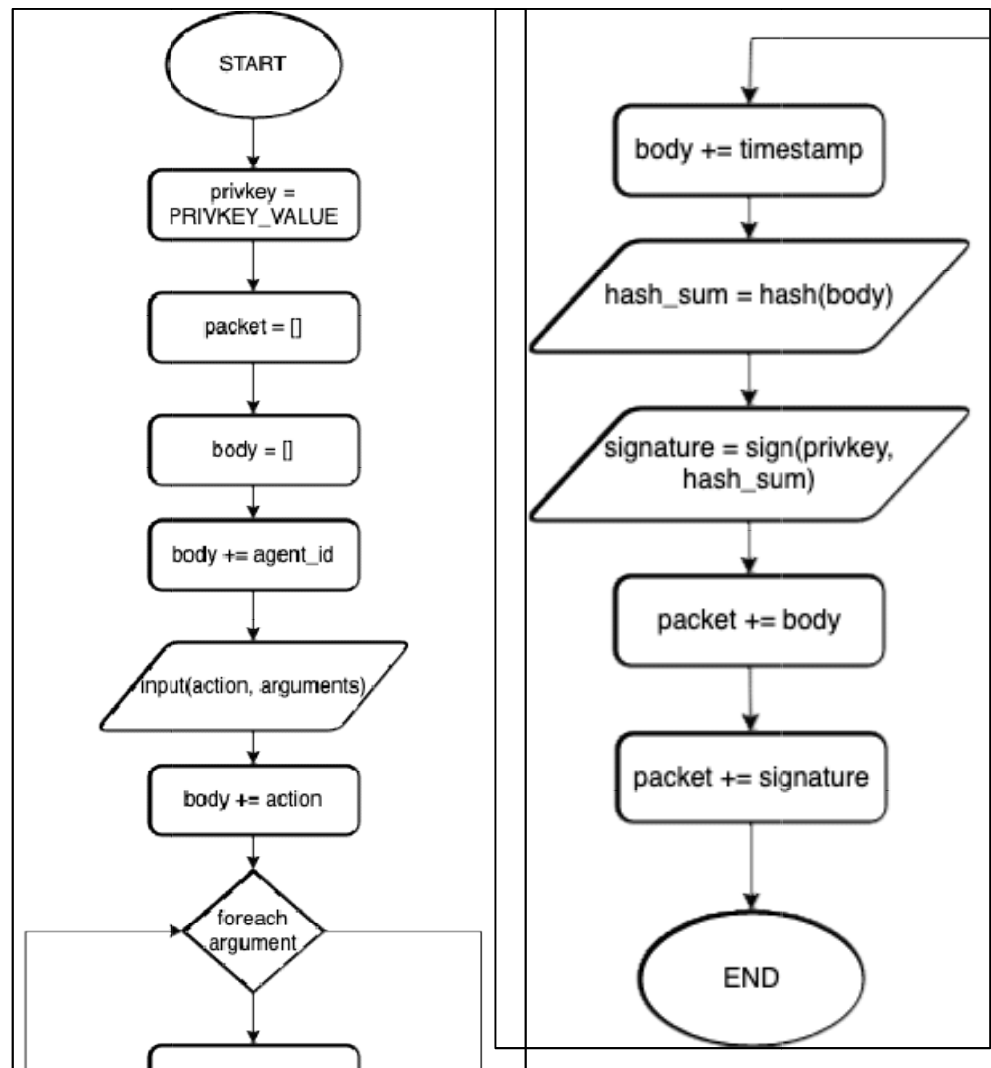


Рисунок 2.7 – Блок-схема алгоритму створення пакету автентифікації

2.3.3 Алгоритм автентифікації з боку сторони, що приймає запит

Агент, що приймає запит, при верифікації наданих у пакеті даних, здійснює наступні дії:

1. Отримання публічного ключа агента, що авторизується, за наданим **agent_id**: якщо агенту з наданим ідентифікатором немає в системі, то автентифікацію здійснити неможливо.
2. Розшифрування підпису агента для отримання хеш суми пакету **sign_hash**: якщо підпис неможливо розшифрувати – його було змінено та автентифікацію здійснити неможливо.

3. Повторне обрахування хеш суми пакету **hash** для звірки зі значенням з підпису
4. Звірка значень **sign_hash** та **hash**: якщо значення не співпадають – дані в тілі пакету були змінені та автентифікацію здійснити неможливо.
5. Перевірка актуальності часової мітки пакету: якщо мітка часу старіша за допустиме значення – вікно для автентифікації вичерпано та здійснити її неможливо.

Якщо усі перевірки були пройдені успішно, то агент надав правильні дані та може бути автентифікований.

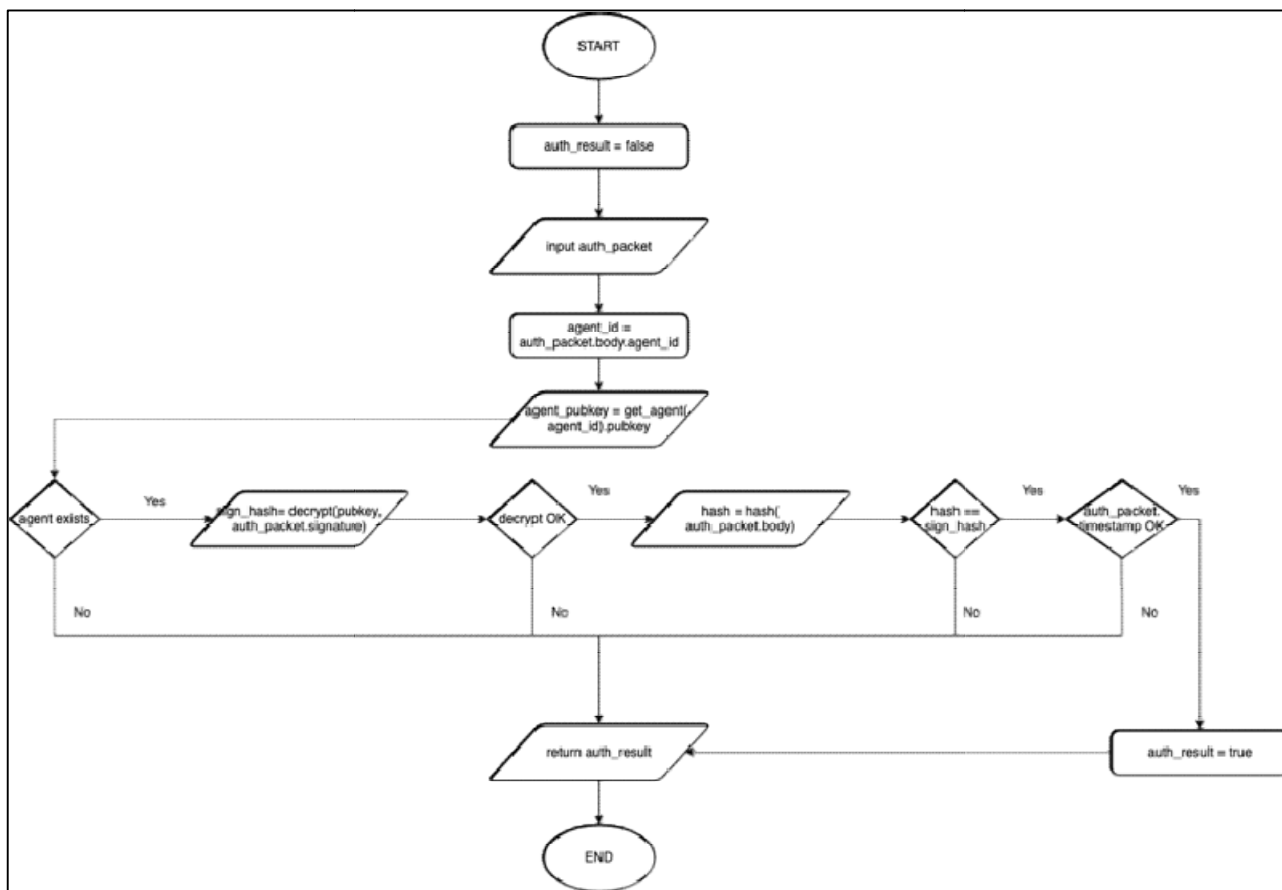


Рисунок 2.8 – Блок-схема алгоритму автентифікації на стороні, що приймає запит

2.4 Запропонована модель авторизації

Запропонована модель авторизації буде базуватися на існуючому функціоналі в вигляді IAM сервісу. Втім, позаяк необхідно реалізувати додаткову можливість керування агентами, що вимагає авторизації, потрібно імплементувати додатковий вид прав, що не залежить від хмарних політик доступу.

2.4.1 Структура підсистеми авторизації

В підсистемі авторизації головну роль відіграє ЦА:

- ЦА. ЦА зберігає для кожного зареєстрованого агента політики **agent_policies**, що містять спеціальний атрибут **admin** для керування агентами та набори ролей ХС для кожного середовища відповідно. При виконанні дії в системі, ЦА перевіряє агента та його права на об'єкт за допомогою функції **authorize**. Якщо дія спрямована на ХС безпосередньо, ЦА видає агентові відповідний акаунт ХС за допомогою функції **grant_iam_account**. Якщо дія спрямована на зміну агентів – ЦА перевіряє атрибут агента **admin**. Тільки агентам з атрибутом **admin** дозволено модифікувати агентів, наприклад, за допомогою функції **modify_agent**.
- МА. При виконанні дій в системі, МА має отримати дозвіл від ЦА. При цьому, необхідно впевнитись в тому, що без явного дозволу ЦА дію виконати неможливо.
 1. Для дій на об'єкти ХС це забезпечується функцією **authorize**, що видає акаунт IAM та закріпленими за ним політиками безпеки ХС. Таким чином, якщо ЦА не надасть акаунт, МА ніяк не зможе

взаємодіяти з ХС та буде заблокований наявною системою безпеки (ІАМ сервісом).

2. Для дій на агенти, ЦА явно перевірятиме атрибут **admin**, що закріплено за агентом в політиках безпеки, перед кожною дією. Якщо агент не є адміністратором, то дію буде заблоковано самим ЦА.

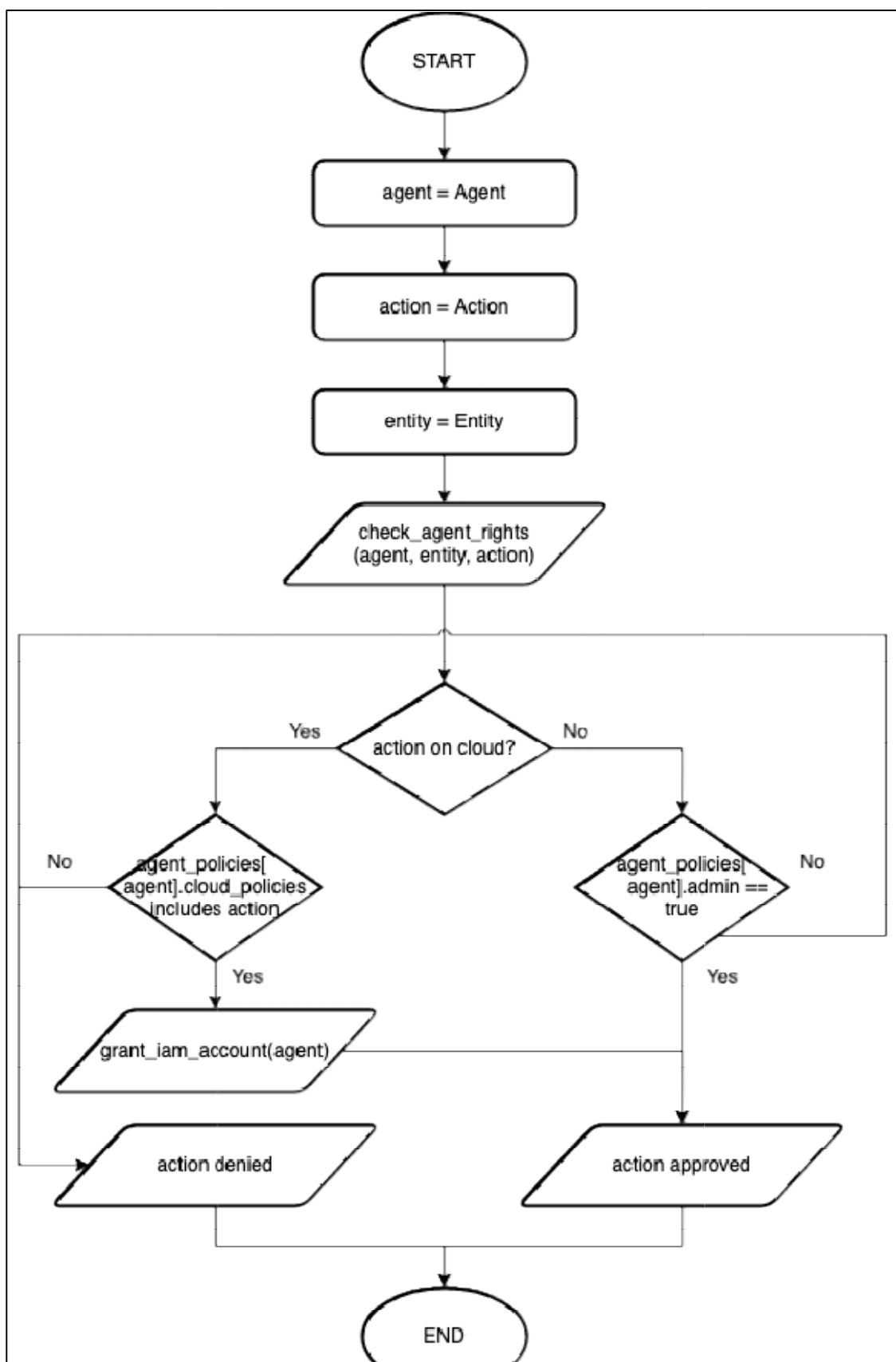


Рисунок 2.9 – Блок-схема алгоритму авторизації з боку ЦА

2.4.2 Авторизація суб'єктів між декількома хмарними середовищами

Завдяки тому, що ЦА єдиний для декількох ХС, агентні політики можуть містити політики доступу ХС до багатьох середовищ. Таким чином, ЦА лише потрібно під час авторизації надавати акаунт IAM, що відповідає тому ХС, до ресурсів якого здійснюється доступ.

2.5 Запропонована модель аудиту подій системи

ЦА логуватиме кожен подію в системі з відповідними метриками (IP адреса агента, час, дія, аргументи, ID агента, тощо.) у відповідний файл. Доступ до подій аудиту можуть мати лише адміністратори (агенти з атрибутом **admin** == true). Журнали аудиту будуть доступні лише для запису нових подій, або читання старих.

```
<time> | <CA ID> | <MA ID> | <MA IP> | <action (arguments)>
```

Рисунок 2.10 – Формат події в журналі аудиту

2.6 Блокування агентів та позбавлення доступу

Завдяки централізованому управлінню підсистемами AAA на ЦА, для блокування агента достатньо видалити асоційований з ним публічний ключ, що використовується для перевірки автентифікації. Для повного видалення необхідно також прибрати асоційовані з агентом політики безпеки з ЦА.

Висновки за розділом 2

Для вирішення проблем безпеки, розглянутих у розділі 1, у роботі побудову агентної моделі безпеки, що реалізовуватиме підсистеми автентифікації, авторизації та аудиту подій у хмарному середовищі.

Розглянуто узагальнено об'єктну модель хмарного середовища, визначено місце інтеграції та компоненти, з якими взаємодіятиме запропонована агентна модель.

Виокремлено два типи агентів: центральний та мобільний, що виконуватимуть ролі сервера та клієнта при виконанні дій в системі. Побудовано відповідні діаграми класів та послідовностей, що описують статичну та динамічну поведінку елементів запропонованої моделі. На прикладі діаграм послідовностей розглянуто проблематику узгодження хмарних середовищ та роль агентів для вирішення даної проблеми.

Запропоновано детальну реалізацію підсистем автентифікації, авторизації та аудиту подій на основі опису класових діаграм та відповідних блок-схем алгоритмів дій.

3 ПОРІВНЯЛЬНИЙ АНАЛІЗ ТА ТЕСТУВАННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АГЕНТОЇ МОДЕЛІ

3.1 Огляд прикладної програми-реалізації агентної моделі

Реалізація агентної моделі виконана на мові Python 3.10.4 та базована на відповідній діаграмі класів, представлений у розділі 2. Надає можливість інтеграції окремих хмарних середовищ в одне за допомогою окремих програмних агентів, що синхронізують доступ між середовищами відповідно до заданих політик безпеки.

Прикладна програма складається з двох частин: програми-центрального агента (ПЦА) та програми-мобільного агента (ПМА).

ПЦА використовується для управління декількома хмарними середовищами одночасно: вона інтегрується в ХС та впроваджує власні політики безпеки (агентні політики безпеки), що описують права МА в системі. Імплементує підсистеми автентифікації, авторизації та аудиту подій, які використовуються для управління відповідних МА та надання їм доступу до дозволених ресурсів хмарного середовища. Спілкування з ПЦА здійснюється по шифрованому TLS каналу за допомогою пакетів у форматі ProtoBuf.

Автентифікація реалізована на основі алгоритму ECDSA (Elliptic-Curve Digital Signature Algorithm). Пакети ProtoBuf складаються з двох частин: тіла (основні дані та метадані) та ECDSA підпису. Метадані містять в собі ідентифікатор агента та мітку часу для ідентифікації, ведення аудиту подій та попередження атак повтору пакетів.

Авторизація реалізована на основі існуючих в ХС сервісів по управлінню доступом (IAM) за допомогою динамічного надання ПМА прав доступу, що зазначені у відповідній агентній політиці безпеки.

Окрім того, ПЦА надає інтерфейс для керування агентними політиками користувачам-адміністраторам.

ПМА використовується кінцевими користувачами для безпосереднього уніфікованого доступу до ресурсів відокремлених хмарних середовищ у рамках агентної моделі безпеки. Отримує від ПЦА сесійний токен для відповідного ХС та здійснює дію в системі.

Структура директорії з необхідними файлами для роботи з програмою:

- `agent.py` – опис абстрактного класу `Agent` відповідно до запропонованої агентної моделі безпеки.
- `central_agent.py` – реалізація похідного від класу `Agent` класу `CentralAgent`. Використовується для запуску ЦА.
- `mobile_agent.py` – реалізація похідного від класу `Agent` класу `MobileAgent`. Використовується для запуску МА.
- `agent_messages.proto` – опис пакетів `ProtoBuf`, які використовуються агентами для спілкування між собою.
- `cloud_config.bash` – BASH скрипт для автоматичного налаштування хмарного середовища AWS для використання агентної моделі безпеки.
- `dummy_role_trust_policy.json` – JSON документ опису ролі хмарного середовища AWS, що буде надана динамічно МА для здійснення дій в хмарі.
- `sample_cloud_policy.json` – JSON документ опису політики хмарного середовища AWS, що буде динамічно надана МА при використанні ролі `dummy_role`

- `log.py` – програмний модуль, що використовується для аудиту подій системи.

3.1.1 Налаштування програмних агентів

Налаштування програмних агентів складається з двох частин:

1. Налаштування ХС. Для коректної роботи програмних агентів необхідно заздалегідь налаштувати цільове хмарне середовище. В даній роботі буде розглянуто приклад налаштування хмарного середовища AWS. Для налаштування в автоматичному режимі ХС потрібно запустити BASH скрипт `cloud_config.bash` (див. додаток А), або ж виконавши наступні команди:

1. Створити користувача для ПЦА:

```
aws iam create-user --user-name central-agent
```

2. Отримати сесійний токен ПЦА для подальшого використання:

```
aws iam create-access-key --user-name central-agent
```

3. Надати ПЦА достатні права в ХС для контролю IAM, наприклад, права адміністратора ХС:

```
aws iam attach-user-policy --user-name central-agent --policy-arn
arn:aws:iam::aws:policy/AdministratorAccess
```

4. Створити непривілейовану роль, до якої динамічно додаватимуться права ПМА з його агентної політики безпеки:

```
aws iam create-role --role-name dummy-role --assume-role-policy-
document file://dummy_role_trust_policy.json
```

5. Створити політику безпеки ХС, що буде використана в агентних політиках:

```
aws iam create-policy --policy-name sample-cloud-policy --policy-
document file://sample_cloud_policy.json
```

2. Налаштування ПЦА. При першому запуску ПЦА буде автоматично створено вбудований акаунт адміністратора, який необхідно використати для

3.1.2 Приклади роботи програмних агентів

Після успішного налаштування ПЦА та ХС, ПМА може виконувати дії в ХС, отримуючи для цього тимчасовий токен ХС від ПЦА.

```
(amd64) + code git:(main) * python3 mobile_agent.py cloud --cloud-uuid 711202675648 sts get-caller-identity
[2022-06-07 20:05:47] DEBUG: loading existing UUID from /Users/adeadfed/.agent/uuid
[2022-06-07 20:05:47] INFO: agent UUID: b2ebb463-60c6-4248-9184-23d3133db59f
[2022-06-07 20:05:47] DEBUG: loading existing private key from /Users/adeadfed/.agent/privkey.ecdsa
[2022-06-07 20:05:47] INFO: public key:
MFYwEAYHkoZiZj0CAQYFK4EEAAoDQgAE8NF0CeeuIcgmKLBjEa0m+cEyaJtSeaqNwHhLUDUFTX+9bpbUeYb3UI/0vk5ENGhzmncDMxrQ
[2022-06-07 20:05:47] INFO: CA connection: 127.0.0.1:9001
[2022-06-07 20:05:47] INFO: CA public key: MFYwEAYHkoZiZj0CAQYFK4EEAAoDQgAE8NF0CeeuIcgmKLBjEa0m+cEyaJtSeaqNwHhLUDUFTX+9bpbUeYb3UI/0vk5ENGhzmncDMxrQ
//EjIeRZd6eDsw==
[2022-06-07 20:05:47] DEBUG: computing message length
[2022-06-07 20:05:47] DEBUG: message length: 165
[2022-06-07 20:05:47] DEBUG: sending message b'\na\nsb2ebb463-60c6-4248-9184-23d3133db59f\x12\x0b\x08\xfb\x06\xfe\x94\x06\x10\xfd\x1d\xad\x1a,\n\x0
c711202675648\x12\x03sts\x1a\x13get-caller-identity"\x02{\x12\xa4\x0b\x07\x94\xfd\xfa\x92n?L'\x1b\x0e5\xc2\x0a\x03\x1f' [%T%\x11\x8c'9P\x026\x0
5\x8d\x02Hq\x0f\x17f\x95\x05C7Q\x03e3\x1a\x04\x09\x0b5b7n2\x11\x95\x1ex/i\x0c'
[2022-06-07 20:05:47] DEBUG: receiving message length
[2022-06-07 20:05:48] INFO: message length: 1035
[2022-06-07 20:05:48] DEBUG: receiving message
[2022-06-07 20:05:48] DEBUG: received message b'\x06\x07\nsb2ebb463-60c6-4248-9184-23d3133db59f\x1a\x0c\x08\xfb\x06\xfe\x94\x06\x10\xfb\x94\xba\xda
a\x03'\x8f\x07('credentials': {'AccessKeyId': 'ASIA2LFX3TPAHS75AK6K', 'SecretAccessKey': 'C92y0tjh/DV0dhhUITGbX46S+jxm80za5Qc9NLM+', 'SessionToken':
'FwoGZXiYXdzEPT/////////wEADHVVVe70xukpP1j1iCLsAQ8x1v4GmXtSWAvRs+76JmX9fpx+MD2IXmp5E6kCgqb+cv8+Y61zzQRMPCkXIY1pTbhNwRn/CX12icHYU4yigRn8WZQnR0JZF
aGtDZtJJ+vyhkus0HJPa4CKzRC/o9mkMGU+fV1KVyAdtVdRw+gvgqMw2z8/+skqYE6EkvEMLYGGGrdWo5Y74AG1I8utw5TrQIGeraMMVRBnrtt5cdau7Lz7QvR3WGbYrAswnCnNnxYxF2EPRAK43
3E9CTXn+5sk7KDMF+/zwIvntsvr76MxXhUpIR/oUkuHHHvq915iFyPQA94bqXhRnGhIAgKPyM/pQGMm/YJNh50WYy5fzPyaOUleVG8BOM0jpcYj/pw0LNZRTisGrPTiM7F1700nmqvyh50mCGk
18iagD08uMbxuNuJ32bhmUw0Y1cJkvq7qS0UfJj8qSWCYxzx2GCur9UhuauIw74jVaCrSKyAIWz4vEu8jE=', 'Expiration': '2022-05-07 19:06:48'}, 'assumed_role_user': {'As
sumedRoleId': 'AROA2LFX3TPAKSMCECRQ55:agent_b2ebb463-60c6-4248-9184-23d3133db59f', 'Arn': 'arn:aws:sts:711202675648:assumed-role/dummy-role/agent_b2
ebb463-60c6-4248-9184-23d3133db59f'})\x12\x0b\x08\x1a0\xec\xda\x1d\xbd\x0b\x9f\xfa\x0b\x04\x0c4\x0b9H\x07M\x0cA\x1a\x03\x13f\x06\x98\xabB\x01y=\xb9
\x0e\x06Gv7\x0dau\x0d0m\x0f\x05\x89\x02\x0c71W\x01\x0c\x01%K\x025Ts\x0e8eA\xec/'
[2022-06-07 20:05:48] INFO: authenticating received message
[2022-06-07 20:05:48] INFO: authentication successful
[2022-06-07 20:05:48] INFO: [CA b2ebb463-60c6-4248-9184-23d3133db59f] action result: SUCCESS
[2022-06-07 20:05:48] INFO: obtained credentials ASIA2LFX3TPAHS75AK6K at 711202675648 for user arn:aws:sts:711202675648:assumed-role/dummy-role/agent_b2
ebb463-60c6-4248-9184-23d3133db59f
[2022-06-07 20:05:48] INFO: executing sts.get-caller-identity()
{
  "UserId": "AROA2LFX3TPAKSMCECRQ55:agent_b2ebb463-60c6-4248-9184-23d3133db59f",
  "Account": "711202675648",
  "Arn": "arn:aws:sts:711202675648:assumed-role/dummy-role/agent_b2ebb463-60c6-4248-9184-23d3133db59f"
}
```

Рисунок 3.7 – Виконання дії sts get-caller-identity в інтегрованому ХС AWS за допомогою ПМА.

```
[2022-06-07 20:05:47] DEBUG: invoking message callback
[2022-06-07 20:05:47] INFO: [b2ebb463-60c6-4248-9184-23d3133db59f] authenticating message from b2ebb463-60c6-4248-9184-23d3133db59f
[2022-06-07 20:05:47] DEBUG: [b2ebb463-60c6-4248-9184-23d3133db59f] authentication successful for b2ebb463-60c6-4248-9184-23d3133db59f
[2022-06-07 20:05:47] INFO: [b2ebb463-60c6-4248-9184-23d3133db59f] authorizing agent b2ebb463-60c6-4248-9184-23d3133db59f
[2022-06-07 20:05:47] DEBUG: [b2ebb463-60c6-4248-9184-23d3133db59f] authorization delegated to cloud environment, skipping step...
[2022-06-07 20:05:47] DEBUG: [b2ebb463-60c6-4248-9184-23d3133db59f] authorization successful
[2022-06-07 20:05:47] INFO: [b2ebb463-60c6-4248-9184-23d3133db59f] processing b2ebb463-60c6-4248-9184-23d3133db59f: cloud_action get-caller-identit
y({})
[2022-06-07 20:05:47] INFO: [b2ebb463-60c6-4248-9184-23d3133db59f] getting agent's b2ebb463-60c6-4248-9184-23d3133db59f policies for cloud environm
ent 711202675648
[2022-06-07 20:05:47] DEBUG: [b2ebb463-60c6-4248-9184-23d3133db59f] obtaining CA credentials for cloud 711202675648
```

Рисунок 3.8 – Процес авторизації дії та отримання політик для МА зафіксовані підсистемою аудиту

```
[2022-06-07 20:05:47] INFO: [b2ebb463-60c6-4248-9184-23d3133db59f] temporary session 'agent_b2ebb463-60c6-4248-9184-23d3133db59f' created in cloud
711202675648 for action get-caller-identity({})
[2022-06-07 20:05:48] DEBUG: computing message length
[2022-06-07 20:05:48] DEBUG: message length: 1035
[2022-06-07 20:05:48] DEBUG: sending message b'\n\x06\x07\nsb2ebb463-60c6-4248-9184-23d3133db59f\x1a\x0c\x08\xfb\x06\xfe\x94\x06\x10\xfb\x94\xba\xda
a\x03'\x8f\x07('credentials': {'AccessKeyId': 'ASIA2LFX3TPAHS75AK6K', 'SecretAccessKey': 'C92y0tjh/DV0dhhUITGbX46S+jxm80za5Qc9NLM+', 'SessionToken':
'FwoGZXiYXdzEPT/////////wEADHVVVe70xukpP1j1iCLsAQ8x1v4GmXtSWAvRs+76JmX9fpx+MD2IXmp5E6kCgqb+cv8+Y61zzQRMPCkXIY1pTbhNwRn/CX12icHYU4yigRn8WZQnR0JZF
aGtDZtJJ+vyhkus0HJPa4CKzRC/o9mkMGU+fV1KVyAdtVdRw+gvgqMw2z8/+skqYE6EkvEMLYGGGrdWo5Y74AG1I8utw5TrQIGeraMMVRBnrtt5cdau7Lz7QvR3WGbYrAswnCnNnxYxF2EPRAK43
3E9CTXn+5sk7KDMF+/zwIvntsvr76MxXhUpIR/oUkuHHHvq915iFyPQA94bqXhRnGhIAgKPyM/pQGMm/YJNh50WYy5fzPyaOUleVG8BOM0jpcYj/pw0LNZRTisGrPTiM7F1700nmqvyh50mCGk
18iagD08uMbxuNuJ32bhmUw0Y1cJkvq7qS0UfJj8qSWCYxzx2GCur9UhuauIw74jVaCrSKyAIWz4vEu8jE=', 'Expiration': '2022-05-07 19:06:48'}, 'assumed_role_user': {'Ass
umedRoleId': 'AROA2LFX3TPAKSMCECRQ55:agent_b2ebb463-60c6-4248-9184-23d3133db59f', 'Arn': 'arn:aws:sts:711202675648:assumed-role/dummy-role/agent_b2e
bb463-60c6-4248-9184-23d3133db59f'})\x12\x0b\x08\x1a0\xec\xda\x1d\xbd\x0b\x9f\xfa\x0b\x04\x0c4\x0b9H\x07M\x0cA\x1a\x03\x13f\x06\x98\xabB\x01y=\xb9
\x0e\x06Gv7\x0dau\x0d0m\x0f\x05\x89\x02\x0c71W\x01\x0c\x01%K\x025Ts\x0e8eA\xec/'
[2022-06-07 20:05:48] DEBUG: receiving message length
[2022-06-07 20:05:50] DEBUG: connection ('127.0.0.1', 64318) closed
```

Рисунок 3.9 – Тимчасовий токен ХС, зафіксований підсистемою аудиту

3.2 Порівняльний аналіз програмних агентів з наявними альтернативами

3.2.1 AWS Cognito

Найближчим аналогом до програмних агентів в AWS є сервіс AWS Cognito. Даний сервіс дозволяє створювати віртуальних користувачів напряду (first party users), або ж інтегрувати їх через найбільш поширені протоколи федеративної автентифікації (third party federated authentication): OAuth 2.0, SAML, OIDC та організувати єдиний доступ до ресурсів хмари. Усі користувачі сервісу організовані в окремі пули (Cognito User Pools).

Як і в агентній моделі, даний сервіс дозволяє окремо управляти правами користувачів через надання їм хмарних політик доступу. Управління наданими користувачам хмарними політиками доступу AWS відбувається через окремий під-сервіс Cognito Identity Pools.

3.2.2 Критерії для порівняльного аналізу

Основними критеріями при порівняльному аналізі прикладних рішень для побудови системи безпеки між багатьма хмарними середовищами є характеристики сумісності (підтримувані протоколи автентифікації, підтримувані хмарні середовища, тощо) та характеристики затратності прикладної програми (час роботи, об'єм використаного мережевого трафіку, процесорний час, об'єм використаної ОЗП).

Критерії сумісності можна легко порівняти за допомогою таблиці. Однак, для порівняння критеріїв затратності прикладних програм необхідно провести експериментальний аналіз.

3.2.3 Порівняльний аналіз критеріїв сумісності

У наступній таблиці наведено порівняння критеріїв сумісності AWS Cognito та запропонованої агентної моделі та їх цінову політику. Хоча запропонована агентна модель на даному етапі не дозволяє інтегрувати користувачів зі сторонніх сервісів за допомогою протоколів федерації, вона надає більш безпечний механізм автентифікації на основі PKI і також закладає основи сумісності з іншими поширеними хмарними середовищами (GCP, Azure), позаяк AWS Cognito займається лише інтеграцією сторонніх сервісів та хмарних середовищ суто в існуюче ХС AWS користувача, не надаючи інструментів для повноцінної взаємодії зі сторонніми ресурсами.

Таблиця 3.1 – Порівняння характеристик AWS Cognito і агентної моделі

Сервіс / Характеристика	AWS Cognito	Запропонована модель
Суб'єкт	User	Мобільний Агент
Де зберігаються дані суб'єктів	Cognito User Pool	Центральний Агент
Підтримувані протоколи федеративної автентифікації	OAuth, SAML, OIDC	-
Основний механізм автентифікації	Парольна автентифікація	Автентифікація на базі ECDSA PKI
Політика безпеки	Cognito User group / AWS IAM role	Агентна політика
Де зберігаються політики безпеки	Cognito Identity Pool	Центральний Агент
Підтримка надання багатьох політик безпеки одному суб'єкту	Так	Так

Спосіб виконання команд у хмарному середовищі	Тимчасовий токен ХС	Тимчасовий токен ХС
---	---------------------	---------------------

Продовження таблиці 3.1

Можливість використовувати доступні політики вибірково	Так	Ні
Ціна	0.05\$ за користувача на місяць	Безкоштовно
Підтримуване хмарне середовище	AWS	AWS/GCP/Azure

3.2.4 Порівняльний аналіз критеріїв затратності

Для порівняльного аналізу прикладних AWS Cognito та реалізованої прикладної програми вибрано наступні критерії: кількість використаного мережевого трафіку, час роботи, загрузка процесора в процентах, кількість використаних циклів процесора, об'єм використаної ОЗП.

Для вимірів об'єму трафіку було використано Wireshark 3.4.7

Для вимірів характеристик процесора та ОЗП було використано вбудовані бібліотеки мови Python 3.10.4 *hwcounter*, *psutil*, *tracemalloc*.

Виміри виконувалися на пристрої MacBook Pro 2021 з операційною системою macOS 12.3, ЦП Apple M1 Pro, та об'ємом LPDDR5 ОЗП 16 ГБ з частотою 6400 МГц.

В аналізі буде порівняно характеристики роботи хмарного середовища з AWS Cognito або прикладною програмою з нормальною роботою хмарного середовища без надбудов над існуючою системою безпеки. Порівняння відбуватиметься за допомогою послідовного виконання викликів сервісу *sts* методом *get-caller-identity*. Для цього звичайне хмарне середовище

використовуватиме заздалегідь підготовлений сесійний токен, а AWS Cognito та прикладна програма будуть щоразу отримувати сесійні токени перед виконанням. Порівняння буде виконуватися після 100, 500 та 1000 викликів *sts get-caller-identity* відповідно.

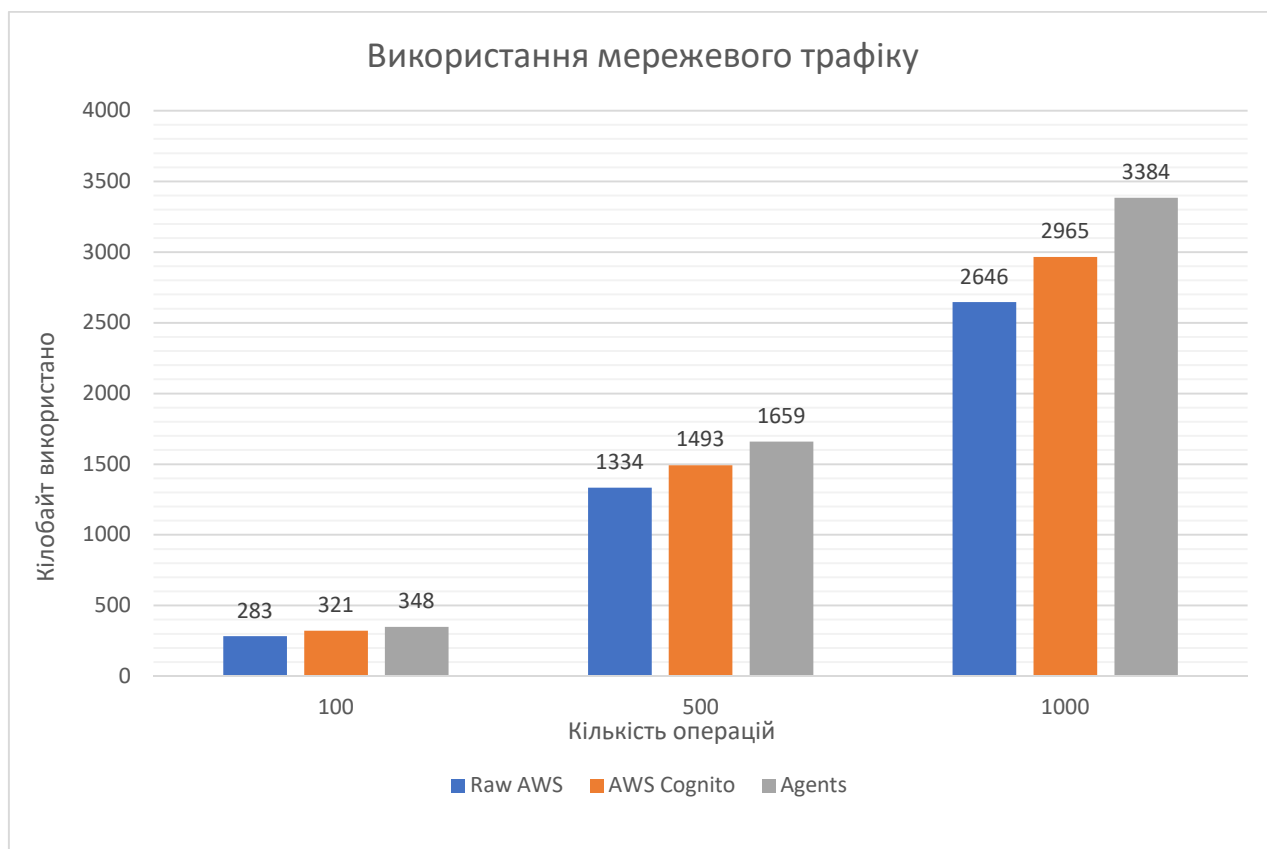


Рисунок 3.10 – Графік використання мережевого трафіку при аналізі

Основний мережевий трафік, що генерується протестованими програмами, відноситься до безпосереднього виконання дії *sts get-caller-identity*, що можна побачити по частці AWS клієнта без додаткових надбудов (Raw AWS). Об'єм трафіку при використанні AWS Cognito збільшився лише на 12.06%. При використанні розробленої у розділі 3 прикладної програми об'єм трафіку збільшився на 27.89%, більш ніж в 2 рази більше за приріст від використання Cognito. Це пов'язано з використанням додаткової автентифікації МА-ЦА, що

відбувається при кожній операції. Об'єм трафіку можна зменшити при використанні алгоритмів стиснення.

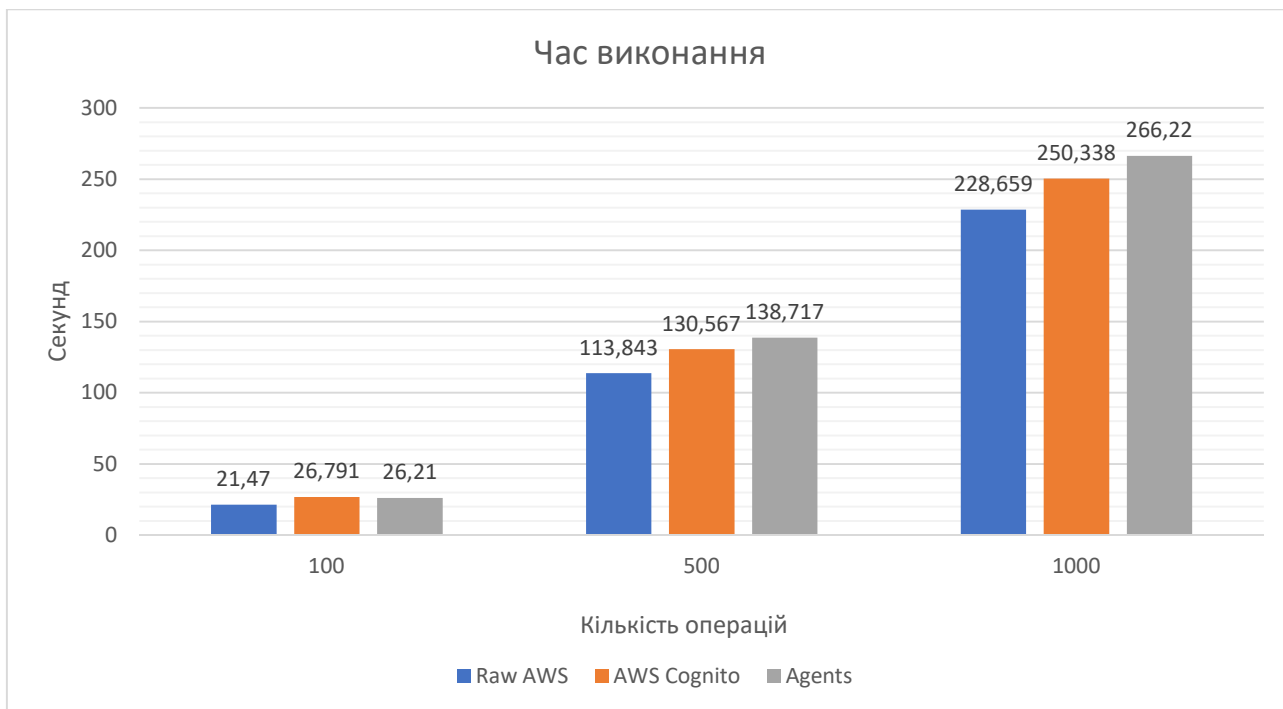


Рисунок 3.11 – Графік загального часу виконання при аналізі

Час виконання від використання додаткових механізмів безпеки змінився не сильно: зростання на 9.48% та 16.42% у Cognito та прикладної програми відповідно. Основний час все ще витрачається на виконання цільової операції *sts get-caller-identity*.



Рисунок 3.12 – Графік використання процесорного часу при аналізі

Однак, кількість затраченого процесорного часу різко збільшилася при використанні Cognito та розробленої програми: на 42.85% та 53.61% відповідно. Це пов'язано зі здійсненням додаткових операцій по отриманню тимчасового токена ХС та ініціалізацією основного клієнту для виконання цільової дії.

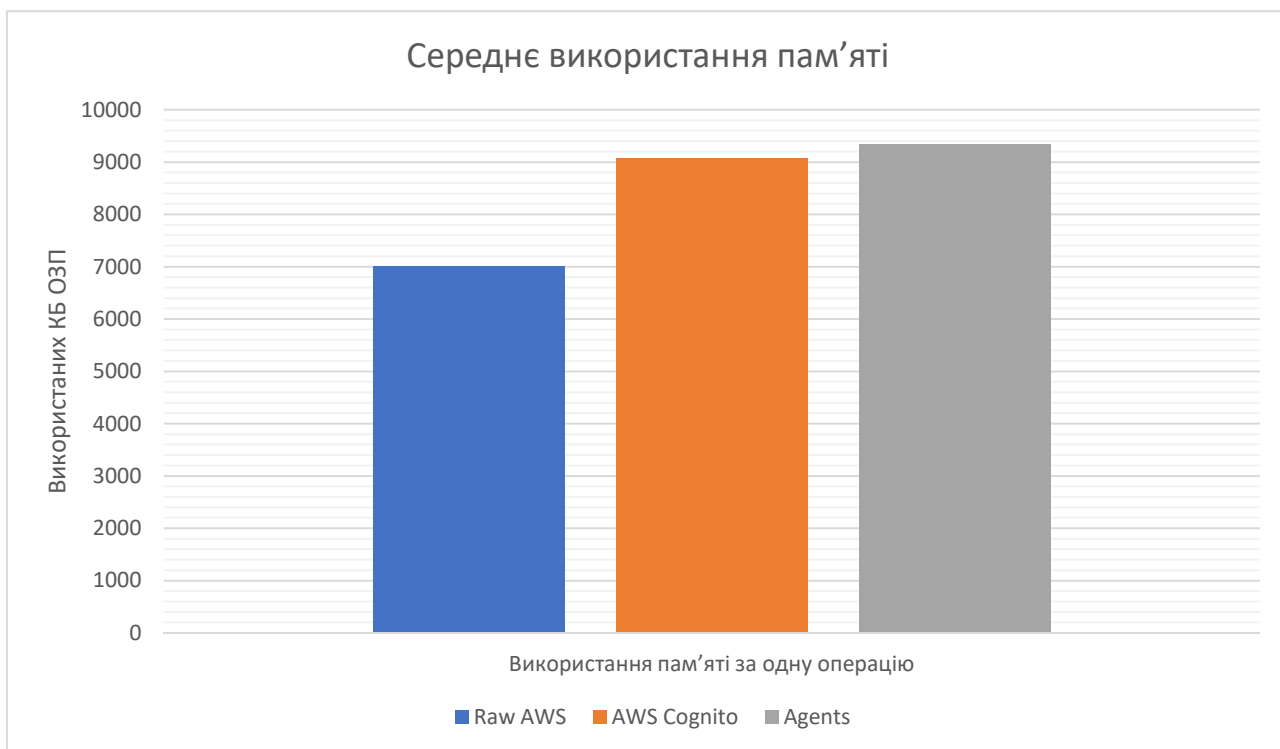


Рисунок 3.13 – Графік середнього використання ОЗП при аналізі

Ці зміни також можна побачити на графіку середнього використання ОЗП при здійсненні повної операції: 29.62% та 33.29% збільшення у Cognito та розробленої програми відповідно. Додаткова пам'ять необхідна для розміщення логіки управління тимчасовими сесійними токенами (отримання, використання, тощо). Варто відмітити, що розроблена прикладна програма доволі ефективно розпоряджається ресурсами системи та не добавляє істотної навантаження на процесор та ОЗП системи у порівнянні з AWS Cognito.

Висновки за розділом 3

У розділі наведено опис прикладної програми-реалізації агентної моделі безпеки для хмарних середовищ, запропонованої у розділі 2. Детально описано кроки налаштування прикладної програми та способи її використання.

Розглянуто найближчі аналоги на прикладі ХС AWS у вигляді сервісу AWS Cognito. Виокремлено основні критерії для порівняння сервісу з запропонованою прикладною програмою. Побудовано порівняльну таблицю за критеріями сумісності. Визначено, що, хоч прикладна програма не підтримує поширені протоколи федеративної автентифікації та має менше гнучкості у налаштуванні хмарних політик доступу в AWS, агентна модель дозволяє будувати повноцінну інфраструктуру, що вміщатиме різні хмарні середовища, позаяк Cognito дозволяє лише інтегрувати користувачів зі сторонніх середовищ у цільове ХС AWS та не надає можливості використання сторонніх ресурсів.

Проведено експериментальний аналіз ефективності використання ресурсів ХС програмою-реалізацією агентної моделі за використанням процесорного часу,

ОЗП, мережевого трафіку та загальним часом виконання. Встановлено, що, у порівнянні з AWS Cognito, прикладна програма істотно більше використовує лише мережевий трафік (на 14.13% відносно Cognito). В інших критеріях розбіжність не перевищувала 7%.

ВИСНОВКИ

У даній роботі розглянуто проблематику класичної моделі безпеки хмарних середовищ на прикладі останніх тенденцій до зростання атак на хмарні середовища та запропоновано агентний підхід до побудови моделі безпеки, що відповідатиме сучасним вимогам. У ході роботи отримано наступні результати:

1. Проаналізовано дані з відкритих джерел про атаки на хмарні середовища, виокремлено найбільш поширені серед зловмисників вектори атак та найменш захищені компоненти системи. Серед основних проблем традиційної моделі безпеки хмарних середовищ визначено недоліки підсистем автентифікації, авторизації та аудиту подій; проблеми розмежування хмарних середовищ та проблеми сумісності ХС різного типу. Побудовано діаграми класів та послідовностей для класичної моделі безпеки ХС. Визначено основні компоненти системи та проаналізовано недоліки моделі на прикладі відповідних діаграм.
2. Визначено актуальні вимоги до моделі безпеки, що містять в собі виконання основних принципів безпеки: конфіденційності, доступності,

цілісності та спостережливості; та специфічні для хмарних середовищ вимоги по централізованому керуванні суб'єктами та їх доступом та забезпечення сумісності між різними типами ХС.

3. Побудовано діаграми класів та послідовностей для агентної моделі безпеки ХС, що повинна будуватися поверх існуючої класичної моделі безпеки. Визначено ролі агентів у системі, детально описано взаємодію об'єктів на прикладі діаграм послідовностей. Виокремлено підсистеми безпеки та задано детальний алгоритм їх роботи у відповідних блок-схемах. Розроблена агентна модель доступу дозволяє будувати нові та легко масштабувати існуючі інфраструктури з багатьох хмарних середовищ, об'єднуючи їх в одне за допомогою програмних агентів. Новостворена інфраструктура відповідатиме усім актуальним вимогам до моделей безпеки ХС, розглянутих у розділі 1 та задовільнятиме базові критерії кібербезпеки: цілісність, доступність, конфіденційність та спостережливість.
4. Розроблено прототип прикладної програми, що реалізовує запропоновану агентну модель безпеки на практиці. Оглянуто етап налаштування ХС та програми, наведено приклади роботи. Здійснено порівняльний аналіз з існуючим аналогом прикладної програми в ХС AWS – Cognito. За результатами аналізу встановлено, що прикладна програма надає більший простір для побудови безпеки між багатьма ХС за допомогою існуючих інструментів ніж Cognito, хоч і дещо поступається за ефективністю використання хмарних ресурсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Qian L. Cloud Computing: An Overview. Conference: Cloud Computing [Текст] / L. Qian, Z. Luo, Y. Du, L. Guo // First International Conference CloudCom. – 2009. – С. 627.
2. Lihui W. An overview of Internet-enabled cloud-based cyber manufacturing [Текст] // Transactions of the Institute of Measurement and Control. – 2017.
3. Ms. N. Rajeswari. Overview of cloud computing and its types [Текст] // Department of Computer Application, AJK college of Arts and Science. – 2019. – Т. 6, № 3. – С. 65.
4. Tadapaneni N.R. Cloud computing security challenges [Текст] // International journal of innovations in engineering research and technology – 2020. – Т. 7, № 6.
5. Modares H. Security in Mobile Cloud Computing, chapter of the book Mobile Networks and Cloud computing Convergence for Progressive Services and Applications [Текст] / H. Modares, J. Lloret, A. Moravejosharieh, R. Salleh// IGI Global. – 2013. – С. 79-91.

6. Surwade Y.P. Information Security [Текст] / Y.P. Surwade, H.J Patil // National Seminar on "Library as a Knowledge and Skill Development Center". – 2019.
7. IAM overview [Электронный ресурс] — Режим доступа: https://cloud.google.com/iam/docs/overview#concepts_related_to_access_management
8. AWS account root user [Электронный ресурс] — Режим доступа: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html
9. IAM JSON policy elements: Resource [Электронный ресурс] — Режим доступа: https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_resource.html
10. Amazon Resource Name(ARNs) [Электронный ресурс] — Режим доступа: <https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html>
11. Scardaci D. Authentication, Authorization, Accounting [Текст] // Towards Interoperable Research Infrastructures for Environmental and Earth Sciences – 2020. – С. 247-271.
12. Rudra B. Understanding and mitigating security and authentication issues in service-oriented network architecture [Текст] / B. Rudra, O.P. Vyas // International Journal of Security and Networks – 2015. – Т. 1, № 10.
13. Policy evaluation logic [Электронный ресурс] — Режим доступа: https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html
14. Akhtar S.I. Inter Cloud Interoperability Use Cases and Gaps in Corresponding Standards [Текст] / S.I. Akhtar, A. Rauf, H. Abbas, M. Amjad // 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress. – 2020.

15. Tehmazi D. A. Inter-Cloud Trust Model Security: Issues and Challenges [Текст] // The 2013 World Congress in Computer Science Computer Engineering, & Applied Computing. – 2013.
16. Perez J.L. 8 cloud adoption statistics that reveal the future of the cloud [Электронный ресурс] — Режим доступа: <https://www.cloudworldwideservices.com/en/cloud-adoption-statistics-cloud-future/> – 2018.
17. Cloud computing for business yet to go mainstream in the EU [Электронный ресурс] — Режим доступа: <https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20210121-1> – 2021.
18. Cloud is the main target of attacks [Электронный ресурс] — Режим доступа: <https://ictmarketexperts.com/en/news/cloud-is-the-main-target-of-attacks/> – 2020.
19. Kretchmer R. Five ways to ensure the cloud doesn't cast shadow over your cybersecurity [Электронный ресурс] /R. Kretchmer, T. Gonen — Режим доступа: <https://www.weforum.org/agenda/2021/04/cloud-cybersecurity/> – 2021.
20. Centrif Research Reveals 90% of Cyberattacks on Cloud Environments Involve Compromised Privileged Credentials [Электронный ресурс] — Режим доступа: <https://delinea.com/news/centrif-research-reveals-90-cyberattacks-cloud-environments> – 2021.
21. Williams S. Majority of businesses already impacted by cloud cyber attack by the time it is investigated [Электронный ресурс] — Режим доступа: <https://securitybrief.asia/story/majority-of-businesses-already-impacted-by-cloud-cyber-attack-by-the-time-it-is-investigated> – 2021.
22. Dejamfar S.M Authentication Techniques in Cloud Computing: A Review /S.M. Dejamfar, S. Najafzaden [Текст] – 2017.

- 23.Token Based Authentication - Implementation Demonstration [Електронний ресурс] — Режим доступу: https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/
- 24.How Does PKI-Based Authentication Work? [Електронний ресурс] — Режим доступу: <https://www.axiad.com/blog/pki-based-authentication> – 2022.
- 25.Electronic signature [Електронний ресурс] — Режим доступу : https://en.wikipedia.org/wiki/Electronic_signature
26. HMAC (Hash-Based Message Authentication Codes) Definition [Електронний ресурс] — Режим доступу: <https://www.okta.com/identity-101/hmac/>
- 27.Downs D.D. Issues in Discretionary Access Control [Текст] / D.D. Downs, J.R. Rub, K.C. Kung, C.S. Jordan // 1985 IEEE Symposium on Security and Privacy. – 1985.
- 28.Azure RBAC documentation [Електронний ресурс] — Режим доступу: <https://docs.microsoft.com/en-us/azure/role-based-access-control/>
- 29.Configure role-based access control [Електронний ресурс] — Режим доступу: <https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control>
- 30.What is ABAC for AWS? [Електронний ресурс] — Режим доступу: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_attribute-based-access-control.html
- 31.Osborn S.L. Mandatory Access Control and Role-Based Access Control Revisited [Текст] – 2000.
- 32.А. Муртазіна, та Л. Гальчинський «Об'єктно-орієнтована модель компрометації протоколу Radius» на XX Всеукраїнській науково-практич. конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики», 2022, с. 268-271.

- 33.Stateless authentication [Электронный ресурс] — Режим доступа: <https://doubleoctopus.com/security-wiki/network-architecture/stateless-authentication/>
- 34.Podas V. Digital signatures and timestamps [Электронный ресурс] — Режим доступа: <https://www.sysadmins.lv/blog-en/digital-signatures.aspx> – 2018.

ПРОГРАМНИЙ КОД

agent.py

```
import ecdsa
import uuid
import logging
import stat
import base64

import log
from datetime import datetime
import os
from pathlib import Path
from protobuf import agent_messages_pb2

class Agent:
    def __init__(self):
        self.AGENT_DIR = str(Path.home().joinpath('.agent'))
        self._init_dir()

        log.setup_logging(self.AGENT_DIR)
        self._init_uuid()
        self._init_key()

    def _init_uuid(self):
        uuid_path = os.path.join(self.AGENT_DIR, 'uuid')
        if os.path.exists(uuid_path):
            logging.debug(f'loading existing UUID from {uuid_path}')
            with open(uuid_path, 'r') as f:
                self.uuid = uuid.UUID(f.read())
        else:
            logging.debug(f'generating new UUID')
            self.uuid = uuid.uuid4()
            with open(uuid_path, 'w') as f:
                f.write(str(self.uuid))
            logging.info(f'agent UUID: {self.uuid}')

    def _init_dir(self):
        if not os.path.exists(self.AGENT_DIR):
            logging.debug(f'creating agent directory at {self.AGENT_DIR}')
            os.makedirs(self.AGENT_DIR)
            os.chmod(self.AGENT_DIR, stat.S_IREAD|stat.S_IWRITE|stat.S_IEXEC)

    def _init_key(self):
```

```

key_path = os.path.join(self.AGENT_DIR, 'privkey.ecdsa')

if os.path.exists(key_path):
    logging.debug(f'loading existing private key from {key_path}')
    with open(key_path, 'rb') as f:
        privkey_der = f.read()
        self.privkey = ecdsa.SigningKey.from_der(privkey_der)
else:
    logging.debug(f'creating new private key to {key_path}')
    self.privkey = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
    logging.info(f'Agent {self.uuid} private
key:\n{base64.b64encode(self.privkey.to_der()).decode("utf-8")}')
    with open(key_path, 'wb') as f:
        f.write(self.privkey.to_der())
    logging.info(f'public
key:\n{base64.b64encode(self.privkey.get_verifying_key().to_der()).decode("utf-8")}')

def _init_socket(self, address, port):
    raise NotImplementedError

def _sign_payload(self, proto_object):
    return self.privkey.sign(proto_object.payload.SerializeToString())

def _verify_payload_sign(self, proto_object):
    raise NotImplementedError

def _load_proto_by_type(self, proto_type, proto_object):
    proto = getattr(agent_messages_pb2, proto_type)()
    proto.ParseFromString(proto_object)
    return proto

def _verify_payload_time(self, message):
    timestamp = message.payload.timestamp.ToDatetime()
    delta = (datetime.utcnow() - timestamp).total_seconds()
    if delta > 3:
        return False
    return True

def do_action(self):
    raise NotImplementedError

def authenticate(self, proto_object):
    raise NotImplementedError

```

central_agent.py

```

import argparse
import logging
import ecdsa

```

```

from agent import Agent
from pathlib import Path
import sqlite3
from datetime import datetime
import uuid
import log
import os
import json
import boto3
from protobuf import agent_messages_pb2
from agent_socket import Socket

class CentralAgent(Agent):
    def __init__(self, cloud_uuid, cloud_secret_uuid, cloud_secret):
        self.uuid = uuid.uuid4()

        self.AGENT_DIR = str(Path.home().joinpath('.agent'))
        self._init_dir()

        log.setup_logging(self.AGENT_DIR)
        self._init_uuid()
        self._init_key()
        self._init_agent_db(cloud_uuid, cloud_secret_uuid, cloud_secret)
        self._init_socket(
            '0.0.0.0',
            9001,
            Path(self.AGENT_DIR).joinpath('tls/cert.pem'),
            Path(self.AGENT_DIR).joinpath('tls/key.pem')
        )

    def _init_agent_db(self, cloud_uuid, cloud_secret_uuid, cloud_secret):
        logging.debug('loading agent DB')
        db_path = str(Path(self.AGENT_DIR).joinpath('agents.sqlite3'))
        db_new = not os.path.exists(db_path)

        self.db = sqlite3.connect(db_path)
        self.db.row_factory = sqlite3.Row

        if db_new:
            logging.info('creating new agent DB')
            self.db.execute("""CREATE TABLE IF NOT EXISTS Agents(
                uuid TEXT NOT NULL PRIMARY KEY,
                admin BOOLEAN DEFAULT(TRUE),
                pubkey BLOB NOT NULL
            )""")
            self.db.execute("""CREATE TABLE IF NOT EXISTS AgentPolicies(
                uuid TEXT NOT NULL PRIMARY KEY,
                agent_uuid TEXT NOT NULL,

```

```

        FOREIGN KEY(agent_uuid) REFERENCES Agents(uuid)
    )"""
    self.db.execute("""CREATE TABLE IF NOT EXISTS CloudPolicies(
        uuid                TEXT NOT NULL PRIMARY KEY,
        cloud_uuid          TEXT NOT NULL,
        agent_policy_uuid   TEXT NOT NULL,
        policy_arn          TEXT NOT NULL,
        FOREIGN KEY(agent_policy_uuid) REFERENCES AgentPolicies(uuid)
    )""")
    self.db.execute("""CREATE TABLE IF NOT EXISTS CloudSecrets(
        uuid                TEXT NOT NULL PRIMARY KEY,
        secret_uuid        TEXT NOT NULL,
        secret_value       TEXT NOT NULL
    )""")
    self.db.execute("""CREATE TABLE IF NOT EXISTS AccountingEvents(
        ca_uuid            TEXT NOT NULL,
        agent_uuid         TEXT NOT NULL,
        timestamp          INT NOT NULL,
        action_info        TEXT NOT NULL,
        agent_ip           TEXT NOT NULL
    )""")

    pubkey_der = self.privkey.get_verifying_key().to_der()
    logging.info(f'creating default admin user {self.uuid}\nPUBLIC
KEY:\n{pubkey_der.hex()}')
    self.db.execute('INSERT INTO Agents VALUES(?,?,?)', [str(self.uuid),
True, pubkey_der])
    self.db.commit()
    if cloud_uuid:
        logging.info(f'adding cloud credentials {cloud_secret_uuid} for cloud
{cloud_uuid}')
        self.db.execute('INSERT INTO CloudSecrets VALUES(?,?,?)', [cloud_uuid,
cloud_secret_uuid, cloud_secret])
        self.db.commit()

    def _get_pubkey_by_uuid(self, uuid):
        query = self.db.execute('SELECT pubkey from Agents WHERE uuid = ?', [uuid,])
        pubkey_der = query.fetchone()['pubkey']
        return ecdsa.VerifyingKey.from_der(pubkey_der)

    def _verify_payload_sign(self, message):
        pubkey = self._get_pubkey_by_uuid(message.payload.agent_uuid)
        return pubkey.verify(message.signature, message.payload.SerializeToString())

    def authenticate(self, message):
        logging.info(f'[{self.uuid}] authenticating message from
{message.payload.agent_uuid}')
        try:
            if self._verify_payload_sign(message):

```

```

        if self._verify_payload_time(message):
            logging.debug(f'[{self.uuid}] authentication successful for
{message.payload.agent_uuid}')
            return True
        except:
            pass
            logging.error(f'[{self.uuid}] authentication failed for
{message.payload.agent_uuid}')
            self.s.send(
                self._prepare_response(agent_messages_pb2.ResultType.FAILURE, {'message':
'authentication failure'})
            )
            return False

    def _get_action(self, message):
        action_type = message.payload.WhichOneof('action')
        return action_type, getattr(message.payload, action_type)

    def _authorize_agent_action(self, agent_uuid, action):
        query = self.db.execute('SELECT admin FROM Agents WHERE uuid = ?',
[agent_uuid,])
        admin = bool(query.fetchone()['admin'])
        logging.debug(f'[{self.uuid}] Admin query for {agent_uuid} returned {admin}')
        return admin

    def _authorize_cloud_action(self, agent_uuid, action):
        logging.debug(f'[{self.uuid}] authorization delegated to cloud environment,
skipping step...')
        return True

    def get_accounting_events(self, **kwargs):
        try:
            logging.info('getting accounting events from the DB')
            query = self.db.execute('SELECT rowid, * FROM AccountingEvents')
            events = query.fetchall()
            result = dict()
            for event in events:
                result.update(
                    {
                        event['rowid']: {
                            'ca_uuid' : event['ca_uuid'],
                            'ma_uuid' : event['agent_uuid'],
                            'ma_ip' : event['agent_ip'],
                            'timestamp' : event['timestamp'],
                            'action_info': event['action_info']
                        }
                    }
                )
            return agent_messages_pb2.ResultType.SUCCESS, result
        except Exception as e:

```

```

        logging.error(f'[{self.uuid}] error while getting accounting events from
the DB')
        logging.exception(e)
        return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to get
events from the DB'}

    def add_accounting_event(self, agent_uuid, action_type, action):
        logging.info(f'[{self.uuid}] processing {agent_uuid}: {action_type}
{action.name}({action.arguments})')

        timestamp = int(datetime.utcnow().timestamp())
        try:
            self.db.execute('INSERT INTO AccountingEvents VALUES (?, ?, ?, ?, ?)',
[
str(self.uuid), agent_uuid, timestamp,
f'{action_type}|{action.name}({action.arguments})', str(self.s.conn_addr)]
            self.db.commit()
        except Exception as e:
            logging.error('failed to add new accounting event')
            logging.exception(e)

    def get_agents(self, **kwargs):
        logging.info(f'[{self.uuid}] obtaining all agents from the DB')
        try:
            query = self.db.execute('SELECT * FROM Agents')
            agents = query.fetchall()
            result = dict()
            for agent in agents:
                result.update({
                    agent['uuid']: {
                        'admin' : agent['admin'],
                        'pubkey' : agent['pubkey'].decode('unicode-escape')
                    }
                })
            return agent_messages_pb2.ResultType.SUCCESS, result
        except:
            logging.error(f'[{self.uuid}] error while obtaining agents from the DB')
            return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to get
agents from the DB'}

    def add_agent(self, **kwargs):
        try:
            agent_uuid = str(uuid.uuid4())
            logging.info(f'[{self.uuid}] adding new agent {agent_uuid}')
            self.db.execute(
                'INSERT INTO Agents VALUES(?, ?, ?)',
                [
                    agent_uuid,
                    kwargs['is_admin'],
                    kwargs['agent_public_key'].encode('unicode-escape')
                ]
            )

```

```

    ])
    self.db.commit()
    logging.debug(f'[{self.uuid}] added agent {agent_uuid}')
    return agent_messages_pb2.ResultType.SUCCESS, {}
except Exception as e:
    logging.error(f'[{self.uuid}] error while inserting new agent into the
DB')
    logging.exception(e)
    return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to
insert new agent into the DB'}

def delete_agent(self, **kwargs):
    logging.info(f'[{self.uuid}] deleting agent {kwargs["agent_uuid"]}')
    try:
        self.db.execute('DELETE FROM Agents WHERE uuid = ?',
[kwargs['agent_uuid'],])
        self.db.commit()
        logging.debug(f'[{self.uuid}] deleted agent {kwargs["agent_uuid"]}')
        return agent_messages_pb2.ResultType.SUCCESS, {}
    except Exception as e:
        logging.error(f'[{self.uuid}] error while deleting agent into the DB')
        logging.exception(e)
        return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to
delete agent from the DB'}

def modify_agent(self, **kwargs):
    logging.info(f'[{self.uuid}] modifying agent {kwargs["agent_uuid"]}')
    try:
        self.db.execute(
            'UPDATE Agents SET admin = ?, pubkey = ? WHERE uuid = ?',
            [
                kwargs['is_admin'],
                kwargs['agent_public_key'].encode('unicode-escape'),
                kwargs['agent_uuid']
            ])
        self.db.commit()
        logging.debug(f'[{self.uuid}] modified agent {kwargs["agent_uuid"]}')
        return agent_messages_pb2.ResultType.SUCCESS, {}
    except Exception as e:
        logging.error(f'[{self.uuid}] error while modifying agent in the DB')
        logging.exception(e)
        return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to
modify agent in the DB'}

def get_policies(self, **kwargs):
    try:
        logging.info(f'[{self.uuid}] obtaining all policies from the DB')
        query = self.db.execute("""
            SELECT

```

```

        a.uuid AS agent_policy_uuid,
        c.uuid AS cloud_policy_uuid,
        a.agent_uuid,
        c.cloud_uuid,
        c.policy_arn
    FROM
        AgentPolicies AS a INNER JOIN
        CloudPolicies AS c
    ON a.uuid = c.agent_policy_uuid""")
    policies = query.fetchall()
    result = dict()
    for policy in policies:
        result.update({policy['cloud_policy_uuid']: {
            'agent_policy_uuid': policy['agent_policy_uuid'],
            'agent_uuid'       : policy['agent_uuid'],
            'cloud_uuid'       : policy['cloud_uuid'],
            'policy_arn'       : policy['policy_arn'],
        }})
    return agent_messages_pb2.ResultType.SUCCESS, result
except Exception as e:
    logging.exception(e)
    logging.error(f'[{self.uuid}] error while obtaining policies from the
DB')

    return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to get
policies from the DB'}

def add_policy(self, **kwargs):
    try:
        agent_policy_uuid = str(uuid.uuid4())
        logging.info(f'[{self.uuid}] creating new agent policy
{agent_policy_uuid}')
        self.db.execute('INSERT INTO AgentPolicies VALUES (?,?)',
[agent_policy_uuid, kwargs['agent_uuid']])
        for policy_arn in kwargs['policy_arns']:
            cloud_policy_uuid = str(uuid.uuid4())
            logging.debug(f'[{self.uuid}] creating new cloud policy
{cloud_policy_uuid}')
            self.db.execute(
                'INSERT INTO CloudPolicies VALUES (?,?,,?)',
                [
                    cloud_policy_uuid,
                    kwargs['cloud_uuid'],
                    agent_policy_uuid,
                    policy_arn
                ])
        self.db.commit()
        logging.debug(f'[{self.uuid}] successfully bonded cloud policies to agent
policy')

```

```

        return agent_messages_pb2.ResultType.SUCCESS, {}
    except Exception as e:
        logging.exception(e)
        logging.error(f'[{self.uuid}] error while creating new policy')
        return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to
create new policy'}

    def modify_policy(self, **kwargs):
        try:
            agent_policy_uuid = kwargs['agent_policy_uuid']
            logging.info(f'[{self.uuid}] modifying agent policy {agent_policy_uuid}')
            self.db.execute('UPDATE AgentPolicies SET agent_uuid = ? WHERE uuid = ?',
[kwargs['agent_uuid'], agent_policy_uuid])

            logging.debug(f'[{self.uuid}] dropping old cloud policies')
            self.db.execute('DELETE FROM CloudPolicies WHERE agent_policy_uuid = ?',
[agent_policy_uuid,])
            for policy_arn in kwargs['policy_arns']:
                cloud_policy_uuid = str(uuid.uuid4())
                logging.info(f'[{self.uuid}] creating new cloud policy
{cloud_policy_uuid}')
                self.db.execute(
                    'INSERT INTO CloudPolicies VALUES (?, ?, ?, ?)',
                    [
                        cloud_policy_uuid,
                        kwargs['cloud_uuid'],
                        agent_policy_uuid,
                        policy_arn
                    ])
            self.db.commit()
            logging.debug(f'[{self.uuid}] successfully bonded cloud policies to agent
policy')
            return agent_messages_pb2.ResultType.SUCCESS, {}
        except:
            logging.error(f'[{self.uuid}] error while modifying policy')
            return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to
modify policy'}

    def delete_policy(self, **kwargs):
        try:
            agent_policy_uuid = kwargs['agent_policy_uuid']
            logging.info(f'[{self.uuid}] deleting agent policy {agent_policy_uuid}')
            self.db.execute('DELETE FROM AgentPolicies WHERE uuid = ?',
[agent_policy_uuid])
            logging.debug(f'[{self.uuid}] dropping associated cloud policies')
            self.db.execute('DELETE FROM CloudPolicies WHERE agent_policy_uuid = ?',
[agent_policy_uuid,])
            logging.debug(f'[{self.uuid}] successfully deleted agent policy
{agent_policy_uuid}')

```

```

        self.db.commit()
        return agent_messages_pb2.ResultType.SUCCESS, {}
    except:
        logging.error(f'[{self.uuid}] error while deleting policy')
        return agent_messages_pb2.ResultType.FAILURE, {'message': 'failed to
delete policy'}

```

```

def _do_agent_action(self, agent_uuid, action):
    services = {
        'agent': {
            'get' : self.get_agents,
            'add' : self.add_agent,
            'delete': self.delete_agent,
            'modify': self.modify_agent
        },
        'policy': {
            'get' : self.get_policies,
            'add' : self.add_policy,
            'delete': self.delete_policy,
            'modify': self.modify_policy
        },
        'accounting': {
            'get' : self.get_accounting_events
        }
    }

    action_arguments = json.loads(action.arguments)
    result_type, result =
services[action.service][action.name](**action_arguments)
    return result_type, result

def _get_cloud_secret(self, cloud_uuid):
    logging.debug(f'[{self.uuid}] obtaining CA credentials for cloud
{cloud_uuid}')
    query = self.db.execute('SELECT secret_uuid, secret_value FROM CloudSecrets
WHERE uuid = ?', [cloud_uuid,])
    result = query.fetchone()
    return result['secret_uuid'], result['secret_value']

def grant_iam_account(self, agent_uuid, action):
    try:
        logging.info(f'[{self.uuid}] getting agent\'s {agent_uuid} policies for
cloud environment {action.cloud_uuid}')
        query = self.db.execute("""
            SELECT policy_arn FROM
            AgentPolicies AS a INNER JOIN
            CloudPolicies AS c ON

```

```

        a.uuid = c.agent_policy_uuid
    WHERE
        a.agent_uuid = ? AND c.cloud_uuid = ?
    """, [agent_uuid, action.cloud_uuid])

    cloud_policy_arns = list()
    for row in query.fetchall():
        cloud_policy_arns.append({'arn': row['policy_arn']})

    cloud_secret_uuid, cloud_secret_value =
self._get_cloud_secret(action.cloud_uuid)

    sts = boto3.Session(
        aws_access_key_id=cloud_secret_uuid,
        aws_secret_access_key=cloud_secret_value
    ).client('sts')

    logging.info(f'[{self.uuid}] obtaining temporary credentials for
{agent_uuid} in cloud environment {action.cloud_uuid}')
    assume_result = sts.assume_role(
        RoleArn=f'arn:aws:iam::{action.cloud_uuid}:role/dummy-role',
        RoleSessionName=f'agent_{agent_uuid}',
        PolicyArns=cloud_policy_arns
    )

    result = {
        'credentials': assume_result['Credentials'],
        'assumed_role_user': assume_result['AssumedRoleUser']
    }

    # get rid of datetime object for proper JSON serialization
    result['credentials']['Expiration'] =
result['credentials']['Expiration'].strftime('%Y-%M-%d %H:%m:%S')

    logging.info(f"[{self.uuid}] temporary session 'agent_{agent_uuid}'
created in cloud {action.cloud_uuid} for action {action.name}({action.arguments})")
    return agent_messages_pb2.ResultType.SUCCESS, result
except Exception as e:
    logging.exception(e)
    return agent_messages_pb2.ResultType.ERROR, {'message': f'failed to
obtain credentials in cloud {action.cloud_uuid}'}

def authorize(self, agent_uuid, action_type, action):
    action_authorize_handlers = {
        'cloud_action': self._authorize_cloud_action,
        'agent_action': self._authorize_agent_action
    }

```

```

        logging.info(f'[{self.uuid}] authorizing agent {agent_uuid}')
        if action_authorize_handlers[action_type](agent_uuid, action):
            logging.debug(f'[{self.uuid}] authorization successful')
            return True

        logging.error(f'[{self.uuid}] authorization failed')
        self.s.send(
            self._prepare_response(agent_messages_pb2.ResultType.FAILURE, {'message':
'authentication failure'})
        )
        return False

    def _init_socket(self, address, port, cert_path, key_path):
        self.s = Socket(address, port, cert_path, key_path)
        self.s.bind()
        self.s.listen(self._listen_callback)

    def do_action(self, agent_uuid, action_type, action):
        self.add_accounting_event(agent_uuid, action_type, action)
        action_handlers = {
            'cloud_action': self.grant_iam_account,
            'agent_action': self._do_agent_action
        }
        result_type, result = action_handlers[action_type](agent_uuid, action)
        self.s.send(
            self._prepare_response(result_type, result)
        )

    def _listen_callback(self, message_proto):
        message = self._load_proto_by_type('AgentRequest', message_proto)
        if self.authenticate(message):
            action_type, action = self._get_action(message)
            if self.authorize(message.payload.agent_uuid, action_type, action):
                self.do_action(message.payload.agent_uuid, action_type, action)

    def _prepare_response(self, result_type, results):
        response = agent_messages_pb2.AgentResponse()

        response.payload.agent_uuid = str(self.uuid)
        response.payload.timestamp.FromDatetime(datetime.utcnow())
        response.payload.type = result_type
        response.payload.results = json.dumps(results)

        response.signature = self._sign_payload(response)

        return response.SerializeToString()

parser = argparse.ArgumentParser()

```

```

parser.add_argument('-cu', '--cloud-uuid', help='UUID of integrated Cloud to be used
by CA')
parser.add_argument('-su', '--cloud-secret-uuid', help='UUID of a secret to be used
by CA in the integrated Cloud')
parser.add_argument('-cs', '--cloud-secret', help='secret to be used by CA in the
integrated Cloud')

args = parser.parse_args()

ca = CentralAgent(args.cloud_uuid, args.cloud_secret_uuid, args.cloud_secret)

```

mobile_agent.py

```

import os
from agent_socket import Socket
from protobuf import agent_messages_pb2
from datetime import datetime
import uuid
from agent import Agent
from pathlib import Path
import logging
import log
import base64
import json
import ecdsa
import argparse
import itertools
import subprocess

class MobileAgent(Agent):
    def __init__(self, ca_address, ca_port, ca_pubkey_der_b64):
        self.uuid = uuid.uuid4()

        self.AGENT_DIR = str(Path.home().joinpath('.agent'))
        self._init_dir()

        log.setup_logging(self.AGENT_DIR)
        self._init_uuid()
        self._init_key()
        self._init_ca(ca_address, ca_port, ca_pubkey_der_b64)

    def _init_ca(self, ca_address, ca_port, ca_pubkey_der_b64):
        ca_conn_info_file = Path(self.AGENT_DIR).joinpath('ca_conn_info')
        ca_pubkey_file = Path(self.AGENT_DIR).joinpath('ca_pubkey')
        if ca_address and ca_port:
            with open(ca_conn_info_file, 'w') as f:
                f.write(f'{ca_address}:{ca_port}')

```

```

else:
    with open(ca_conn_info_file, 'r') as f:
        ca_address, ca_port = f.read().split(':')

logging.info(f'CA connection: {ca_address}:{ca_port}')
self._init_socket(ca_address, int(ca_port))

if ca_pubkey_der_b64:
    ca_pubkey_der = base64.b64decode(ca_pubkey_der_b64)
    with open(ca_pubkey_file, 'wb') as f:
        f.write(ca_pubkey_der)
else:
    with open(ca_pubkey_file, 'rb') as f:
        ca_pubkey_der = f.read()

logging.info(f'CA public key: {base64.b64encode(ca_pubkey_der).decode("utf-
8")}')
self.ca_pubkey = ecdsa.VerifyingKey.from_der(ca_pubkey_der)

def _init_socket(self, address, port):
    self.s = Socket(address, port, Path(self.AGENT_DIR).joinpath('tls/cert.pem'))
    self.s.connect()

def _verify_payload_sign(self, message):
    return self.ca_pubkey.verify(message.signature,
message.payload.SerializeToString())

def authenticate(self, message):
    logging.info('authenticating received message')
    try:
        if self._verify_payload_sign(message):
            if self._verify_payload_time(message):
                logging.info('authentication successful')
                return True
    except:
        pass
    logging.error('authentication failed')
    return False

def _prepare_request(self, action_type, **kwargs):
    request = agent_messages_pb2.AgentRequest()
    request.payload.agent_uuid = str(self.uuid)
    request.payload.timestamp.FromDatetime(datetime.utcnow())

    if action_type == agent_messages_pb2.ActionType.AGENT:
        request.payload.agent_action.service = kwargs.get('service')
        request.payload.agent_action.name = kwargs.get('name')
        request.payload.agent_action.arguments =
json.dumps(kwargs.get('arguments'))

```

```

    else:
        request.payload.cloud_action.cloud_uuid = kwargs.get('cloud_uuid')
        request.payload.cloud_action.service     = kwargs.get('service')
        request.payload.cloud_action.name        = kwargs.get('name')
        request.payload.cloud_action.arguments   =
json.dumps(kwargs.get('arguments'))

    request.signature = self._sign_payload(request)
    return request.SerializeToString()

def do_action(self, action_type, **kwargs):
    request = self._prepare_request(
        action_type,
        **kwargs
    )
    self.s.send(request)
    response = self._load_proto_by_type('AgentResponse', self.s.recv())
    if not self.authenticate(response):
        logging.error('failed to authenticate CA response')
        raise ValueError
    return response

def invoke_agent_action(ma, args):
    service_method_required_args = {
        'agent': {
            'get':    [],
            'add':    ['is_admin', 'agent_public_key'],
            'delete': ['agent_uuid'],
            'modify': ['agent_uuid', 'is_admin', 'agent_public_key']
        },
        'policy': {
            'get':    [],
            'add':    ['agent_uuid', 'policy_arns', 'cloud_uuid'],
            'delete': ['agent_policy_uuid'],
            'modify': ['agent_policy_uuid', 'agent_uuid', 'policy_arns',
'cloud_uuid']
        },
        'accounting': {
            'get':    []
        }
    }
    required_args = service_method_required_args[args.service][args.method]
    try:
        arguments = dict()
        for argument in required_args:
            arguments.update({argument: getattr(args, argument)})
    except Exception as e:
        logging.error('missing required argument!')
```

```

        logging.exception(e)
        return

    try:
        response = ma.do_action(
            agent_messages_pb2.ActionType.AGENT,
            service=args.service,
            name=args.method,
            arguments=arguments
        )
        logging.info(f'[CA {response.payload.agent_uuid}] action result:
{agent_messages_pb2.ResultType.Name(response.payload.type)}')
        results = json.loads(response.payload.results)
        logging.info(f'results:\n{json.dumps(results, indent=4, sort_keys=True)}\n')
    except Exception as e:
        logging.error('the following exception has occurred:')
        logging.exception(e)

def invoke_cloud_action(ma, args):
    arguments = dict()
    argument_pairs = list(itertools.zip_longest(*(iter(args.arguments[0]),) * 2))

    for argument_pair in argument_pairs:
        if argument_pair[0]:
            arguments.update({argument_pair[0]: argument_pair[1]})

    try:
        response = ma.do_action(
            agent_messages_pb2.ActionType.CLOUD,
            cloud_uuid=args.cloud_uuid,
            service=args.service,
            name=args.method,
            arguments=arguments
        )
        logging.info(f'[CA {response.payload.agent_uuid}] action result:
{agent_messages_pb2.ResultType.Name(response.payload.type)}')
        results = json.loads(response.payload.results)
        credentials = results['credentials']
        logging.info(f'obtained credentials {credentials["AccessKeyId"]} at
{args.cloud_uuid} for user {results["assumed_role_user"]["Arn"]}')

        run_arguments = ['aws', args.service, args.method]
        run_arguments.extend(args.arguments[0])
        logging.info(f'executing {args.service}.{args.method}({",
".join(args.arguments[0])})')
        cloud_process = subprocess.Popen(run_arguments, env=dict(os.environ,
**{
    'AWS_ACCESS_KEY_ID'      : credentials['AccessKeyId'],
    'AWS_SECRET_ACCESS_KEY' : credentials['SecretAccessKey'],
    'AWS_SESSION_TOKEN'     : credentials['SessionToken']

```

```

    ))) # aws currently supported
    cloud_process.wait()
except Exception as e:
    logging.error('the following exception has occurred:')
    logging.exception(e)

def parse_args():
    parser = argparse.ArgumentParser()
    subparsers = parser.add_subparsers(help='<agents> | <cloud>', dest='mode')

    agents_subparser = subparsers.add_parser('agents')
    agents_subparser.add_argument('service', choices=['agent', 'policy',
'accounting'], help='CA service to interact with')
    agents_subparser.add_argument('method', help='service method for MA
to perform')
    agents_subparser.add_argument('-au', '--agent-uuid', help='target mobile agent
UUID')
    agents_subparser.add_argument('--is-admin', help='specify if agent
should be an admin', action=argparse.BooleanOptionalAction, default=False)
    agents_subparser.add_argument('-apk', '--agent-public-key', help='public key of
the target mobile agent')
    agents_subparser.add_argument('-apu', '--agent-policy-uuid', help='UUID of target
agent policy')
    agents_subparser.add_argument('-cu', '--cloud-uuid', help='UUID of target
cloud')
    agents_subparser.add_argument('-pa', '--policy-arns', help='ARN of cloud policy
to specify in the target agent policy', nargs='+')

    cloud_subparser = subparsers.add_parser('cloud')
    cloud_subparser.add_argument('service', help='cloud service for MA to use')
    cloud_subparser.add_argument('method', help='cloud service method for MA to
perform')
    cloud_subparser.add_argument('arguments', help='arguments for respective method
of cloud service', action='append', nargs=argparse.REMAINDER)
    cloud_subparser.add_argument('-cu', '--cloud-uuid', help='UUID of target
cloud')

    parser.add_argument('-ci', '--ca-ip', help='IP address of CA to connect')
    parser.add_argument('-cp', '--ca-port', help='Port of CA to connect')
    parser.add_argument('-pk', '--ca-pub-key', help='Public key of CA in base64
encoded form')

    return parser.parse_args()

args = parse_args()

ma = MobileAgent(args.ca_ip, args.ca_port, args.ca_pub_key)
action_modes = {
    'agents': invoke_agent_action,

```

```

        'cloud' : invoke_cloud_action
    }
    action_modes[args.mode](ma, args)

```

agent_socket.py

```

import socket
import logging
import struct
import ssl

class Socket:
    def __init__(self, address, port, tls_cert, tls_key=None):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.address = address
        self.port = port
        self.context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
        if tls_key:
            self.context.load_cert_chain(tls_cert, tls_key)
        else:
            self.context.load_verify_locations(tls_cert)

    def bind(self):
        self.s.bind((self.address, self.port))

    def connect(self):
        self.s.connect((self.address, self.port))
        self.s = self.context.wrap_socket(self.s)

    def listen(self, callback):
        self.s.listen()
        while True:
            conn, self.conn_addr = self.s.accept()
            self.tls_conn = self.context.wrap_socket(conn, server_side=True)
            logging.info(f'accepted connection from {self.conn_addr}')
            while True:
                try:
                    message = self.recv()
                    logging.debug(f'invoking message callback')
                    callback(message)
                except:
                    logging.debug(f'connection {self.conn_addr} closed')
                    break

    def recv(self):
        tls_conn = self.tls_conn if hasattr(self, 'tls_conn') else self.s

        logging.debug(f'receiving message length')

```

```

message_len_raw = tls_conn.recv(4)
message_len = struct.unpack('>I', message_len_raw)[0]
logging.debug(f'message length: {message_len}')

logging.debug(f'receiving message')
message = bytearray()
while len(message) < message_len:
    message_chunk = tls_conn.recv(message_len - len(message))
    if not message_chunk:
        return None
    message.extend(message_chunk)
logging.debug(f'received message {bytes(message)}')
return bytes(message)

def send(self, message):
    tls_conn = self.tls_conn if hasattr(self, 'tls_conn') else self.s

    logging.debug(f'computing message length')
    message_len_raw = struct.pack('>I', len(message))
    logging.debug(f'message length: {len(message)}')
    tls_conn.sendall(message_len_raw)
    logging.debug(f'sending message {message}')
    tls_conn.sendall(message)

```

log.py

```

import logging
from datetime import date
from colorama import Fore, Back, Style
from pathlib import Path

class CustomFormatter(logging.Formatter):
    FORMATS = {
        logging.DEBUG :
        f'{Back.YELLOW}{Fore.BLACK}[%asctime)s]{Style.RESET_ALL}{Fore.CYAN}
DEBUG{Style.RESET_ALL}: %(message)s',
        logging.ERROR :
        f'{Back.YELLOW}{Fore.BLACK}[%asctime)s]{Style.RESET_ALL}{Fore.RED}
ERROR{Style.RESET_ALL}: %(message)s',
        logging.INFO :
        f'{Back.YELLOW}{Fore.BLACK}[%asctime)s]{Style.RESET_ALL}{Fore.GREEN}
INFO{Style.RESET_ALL}: %(message)s',
        logging.WARNING :
        f'{Back.YELLOW}{Fore.BLACK}[%asctime)s]{Style.RESET_ALL}{Fore.YELLOW}
WARN{Style.RESET_ALL}: %(message)s',
        logging.CRITICAL : f'{Back.YELLOW}{Fore.BLACK}[%asctime)s]{Style.RESET_ALL}
{Back.RED}{Fore.WHITE}CRIT{Style.RESET_ALL}: %(message)s',

```

```

        'DEFAULT' : f'{Back.YELLOW}{Fore.BLACK}[% (asctime)s]{Style.RESET_ALL}
%(levelname)5s: %(message)s'
    }

    def format(self, record):
        log_fmt = self.FORMATS.get(record.levelno)
        formatter = logging.Formatter(log_fmt, "%Y-%m-%d %H:%M:%S")
        return formatter.format(record)

def setup_logging(log_dir):
    log_file_path = Path(log_dir).joinpath(f'agent_log_{date.today()}')

    sh = logging.StreamHandler()
    sh.setFormatter(CustomFormatter())
    fh = logging.FileHandler(log_file_path)
    fh.setFormatter(CustomFormatter())

    logging.basicConfig(
        level=logging.DEBUG,
        handlers=[
            sh,
            fh
        ]
    )

```

agent_messages.proto

```

syntax = "proto3";
import "google/protobuf/timestamp.proto";

package agent_messages;

enum ActionType {
    CLOUD = 0;
    AGENT = 1;
}

enum ResultType {
    SUCCESS = 0;
    FAILURE = 1;
}

message CloudAction {
    string cloud_uuid = 1;
    string service    = 2;
}

```

```

    string name          = 3;
    string arguments     = 4;
}

message AgentAction {
    string service       = 1;
    string name          = 2;
    string arguments     = 3;
}

message AgentRequest {
    message Payload {
        string agent_uuid = 1;
        google.protobuf.Timestamp timestamp = 2;
        oneof action {
            CloudAction cloud_action = 3;
            AgentAction agent_action = 4;
        }
    }
    Payload payload      = 1;
    bytes signature      = 2;
}

message AgentResponse {
    message Payload {
        string agent_uuid = 1;
        int32 type        = 2;
        google.protobuf.Timestamp timestamp = 3;
        string results     = 4;
    }
    Payload payload = 1;
    bytes signature = 2;
}

```

cloud_config.bash

```

#!/bin/bash
export AWS_ACCESS_KEY_ID=<YOUR ACCESS KEY ID>
export AWS_PRIVATE_ACCESS_KEY=<YOUR PRIVATE ACCESS KEY>

aws iam create-user          --user-name central-agent
aws iam create-access-key   --user-name central-agent
aws iam attach-user-policy --user-name central-agent --policy-arn
arn:aws:iam::aws:policy/AdministratorAccess
aws iam create-role         --role-name dummy-role --assume-role-policy-document
file://dummy_role_trust_policy.json
aws iam create-policy       --policy-name sample-cloud-policy --policy-document
file://sample_cloud_policy.json

```

dummy_role_trust_policy.json

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": { "AWS": "arn:aws:iam::<YOUR AWS CLOUD ID>:user/central-agent"
  },
    "Action": "sts:AssumeRole"
  }
}
```

sample_cloud_policy.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```