НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.9:658.1

ПОГОДЖЕНО

Декан факультету
Інформаційних технологій
_____/ Болбот І.М., д.т.н, проф. /
підпис ПІБ, вчене звання і ступінь

«___»_____2024 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., д.п.н., доцент /
підпис ПІБ, вчене звання і ступінь

«___»_____2024 р.

МАГІСТЕРСЬКА РОБОТА

На тему: «Дослідження методів і засобів управління користувачами в комп'ютерній системі підприємства та їх програмна реалізація»
Спеціальність: 123 «Комп’ютерна інженерія»
Освітня програма: Комп’ютерні системи захисту інформації

Керівник дипломного проєкту:_____/ Лахно В.А. /
підпис ПІБ

Виконав:_____/ Гребенюк Б.В. /
підпис ПІБ

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»

завідувач кафедри

комп'ютерних систем, мереж та кібербезпеки

____________/ Касаткін Д.Ю., д.п.н., доцент /

підпис ПІБ, вчене звання і ступінь

«__» _____ 2024 р.

ЗАВДАННЯ

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Гребенюку Богдану Вадимовичу

(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): 123 «Комп'ютерна інженерія»

Освітня програма: Комп'ютерні системи захисту інформації

Тема магістерської роботи: Дослідження методів і засобів управління користувачами в комп'ютерній системі підприємства та їх програмна реалізація

затверджена наказом ректора НУБіП України від " 1 " листопада 2021 № 1859 "С"

Термін подання завершеної роботи на кафедру _____

Вихідні дані до магістерської роботи: методи управління автентифікацією, доступом до ресурсів системи, методи аналізу стану безпеки, моделі безпеки; інтерактивне хмарне середовище для програмної розробки Google Colab.

Перелік питань, що підлягають дослідженню:

1. Аналіз стану предметної області управління користувачами
2. Оцінка ефективності й доцільності використання методів автентифікації, управління доступом і безпекою, моделей безпеки
3. Програмна реалізація найкращих за результатами дослідження методів управління користувачами

Дата видачі завдання " 1 " листопада 2021 р.

Керівник магістерської роботи _____ / Лахно В.А. д.т.н., професор /

(підпис) (ПІБ, вчене звання і ступінь)

Завдання прийняв до виконання _____ / Гребенюк Б.В. /

(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка задачі магістерської роботи	01.09.2024	Виконано
2	Аналіз предметної області	10.09.2024	Виконано
3	Проєктування системи	15.09.2024	Виконано
4	Реалізація системи	05.10.2024	Виконано
5	Тестування розробленої системи	15.10.2024	Виконано
6	Оформлення пояснювальної записки	01.11.2024	Виконано
7	Оформлення графічного матеріалу	05.11.2024	Виконано

Студент _____ / Гребенюк Б.В. /
підпис ПІБ

Керівник проєкту (роботи) _____ / Лахно В.А. /
підпис ПІБ

РЕФЕРАТ

Пояснювальна записка: 94 сторінки, 19 рисунків, 4 таблиці, 5 лістингів, 36 джерел.

КОРИСТУВАЧ, АВТЕНТИФІКАЦІЯ, ДОСТУП, ІНФОРМАЦІЙНА БЕЗПЕКА, КОМП’ЮТЕРНА СИСТЕМА, СИСТЕМНИЙ ПІДХІД, PYTHON

Мета – дослідження ефективності методів управління користувачами за низкою критеріїв, визначених окремо для всіх підгруп, що втілюють різні аспекти даного процесу; створення програмних засобів, що реалізують найбільш універсальні за оцінками методи кожної підгрупи, зокрема розробка комплексного рішення з їхнім поєднанням.

Об'єкт – інтерактивне хмарне середовище для програмної розробки Google Colab.

Предмет – сучасні методи ідентифікації й автентифікації, управління доступом, аналізу й управління безпекою, моделі безпеки.

Робота складається з трьох розділів.

У першому розділі проведений аналіз предметної області, зокрема розглянуто сучасні методи та засоби управління користувачами, проблеми і виклики, пов’язані з цим питанням, а також принципи застосування системного підходу.

У другому розділі проведено оцінку ефективності 4 підгруп методів за актуальними критеріями на основі специфіки й принципів їх роботи, а також ряду існуючих наукових статей, публікацій, досліджень на дану тематику.

Третій розділ є програмною реалізацією мовою Python у середовищі Google Colab найкращих за результатами дослідження у рамках попереднього розділу методів як окремо, так і в поєднанні один з одним у вигляді повноцінного засобу управління користувачами, зокрема авторизацією, доступом та інцидентами інформаційної безпеки у рамках відповідної моделі безпеки.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ 6
ВСТУП 8
1 АНАЛІТИЧНИЙ ОГЛЯД 9
1.1 Сучасні методи та засоби управління користувачами в комп'ютерних системах підприємств 9
1.2 Аналіз сучасних методів управління користувачами 13
1.3 Аналіз основних викликів та проблем, пов'язаних із управлінням користувачами 22
1.4 Системний підхід до управління користувачами на підприємствах 25
2 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ УПРАВЛІННЯ КОРИСТУВАЧАМИ 28
2.1 Класифікація методів управління користувачами 28
2.2 Визначення критеріїв для оцінки методів управління користувачами 29
2.3 Дослідження методів ідентифікації та автентифікації 34
2.4 Дослідження методів управління доступом 41
2.5 Дослідження методів аналізу і управління безпекою 48
2.6 Дослідження моделей безпеки 54
3 ПРОГРАМНА РЕАЛІЗАЦІЯ 61
3.1 Програмна реалізація MFA 61
3.2 Програмна реалізація ABAC 64
3.3 Програмна реалізація SIEM 68
3.4 Програмна реалізація ZTSM 72
3.5 Програмна реалізація комплексного рішення управління користувачами 76
ВИСНОВОК 89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 90

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

IAM	–	Identity and Access Management, управління ідентифікацією та доступом
SSO	–	Single Sign-On, технологія єдиного входу
MFA	–	Multi-Factor Authentication, багатофакторна автентифікація
RBAC	–	Role-Based Access Control, рольова модель доступу
PAM	–	Privileged Access Management, управління привілейованим доступом
ABAC	–	Attribute-Based Access Control, атрибутна модель доступу
UEBA	–	User and Entity Behavior Analytics, поведінковий аналіз поведінки користувачів та сутностей
SIEM	–	Security Information and Event Management, процес керування інформаційною безпекою та подіями безпеки
DLP	–	Data Loss Prevention, запобігання витокам інформації
ZTSM	–	Zero Trust Security Model, модель «нульової довіри»
CASB	–	Cloud Access Security Broker, посередник захищеного доступу до хмари
VM	–	Vulnerability Management, управління вразливостями
AD	–	Active Directory, служба каталогів корпорації Microsoft
API	–	Application Programming Interface, прикладний програмний інтерфейс
ISE	–	Identity Services Engine, платформа CISCO для моніторингу, управління доступом та захисту корпоративних мереж
IBM	–	International Business Machines, один із провідних виробників комп'ютерів і програмного забезпечення
ERP	–	Enterprise Resource Planning, система з планування ресурсів підприємства
CRM	–	Customer Relationship Management, система для управління відносинами з клієнтами

MTBF	–	Mean Time Between Failures, середній наробіток між відмовами
MTTR	–	Mean Time to Failure, середній час до відмови
GDPR	–	General Data Protection Regulation, Загальний регламент про захист даних Європейського Союзу
ISO	–	International Organization for Standardization, Міжнародна організація зі стандартизації
IEC	–	International Electrotechnical Commission, Міжнародна електротехнічна комісія
NIST	–	National Institute of Standards and Technology, Національний інститут стандартів і технологій США
AWS	–	Amazon Web Services, хмарна платформа компанії Amazon
SAML	–	Security Assertion Markup Language, мова розмітки декларації безпеки обміну інформації про авторизацію
IDS	–	Intrusion Detection System, системи виявлення вторгнень
OTP	–	One Time Password, одноразовий пароль

ВСТУП

На сьогоднішній день при доволі значній тенденції інтенсивного розвитку цифрових технологій, комп'ютерні системи підприємств стають все більш складними і різноманітними. Це створює нові вимоги до управління користувачами, включаючи забезпечення інформаційної безпеки, ефективності управління користувацькою діяльністю, зокрема правами доступу до ресурсів організації. Зростання обсягу даних, з якими працюють підприємства, а також підвищена залежність від інформаційних систем, робить питання управління користувачами все більш гострим.

Метою даної магістерської роботи є дослідження сучасних методів і засобів управління користувачами в комп'ютерних системах підприємств та розробка програмних рішень найактуальніших серед них. На основі аналізу існуючих підходів і технологій, у роботі будуть розглянуті ключові методи автентифікації, ідентифікації та управління доступом, а також моделі безпеки, що забезпечують надійний захист інформаційних активів підприємства.

Актуальність роботи зумовлена необхідністю забезпечення системного підходу до управління користувачами, що включає не лише традиційні методи авторизації в системі й надання відповідних повноважень, але й сучасні підходи до моніторингу та аналізу активності користувачів, наприклад, системи управління інцидентами та аномаліями. Використання таких методів є необхідним для зниження ризиків, пов'язаних із несанкціонованим доступом, підвищення стійкості до внутрішніх і зовнішніх загроз, а також для забезпечення відповідності вимогам державних і міжнародних стандартів.

Дослідження, проведені в роботі, дозволять визначити на основі низки критеріїв ефективність сучасних методів управління користувачами в залежності від контексту, тобто зрозуміти доцільність їхнього застосування в комп’ютерній системі. Розроблені рішення натомість можуть бути використані, як основа для реалізації більш комплексних програм як окремих методів, так і повноцінної системи управління користувачами з рядом взаємопов’язаних аспектів.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасні методи та засоби управління користувачами в комп'ютерних системах підприємств

Управління користувачами в комп'ютерних системах підприємств включає в себе різні методи та інструменти для ефективного контролю, надання доступу та забезпечення безпеки інформації. Нижче наведено перелік сучасних методів управління користувачами та їх опис:

- IAM (Identity and Access Management): IAM – це комплексний метод, спрямований на керування доступом до ресурсів. Системи IAM визначають та керують правами доступу користувачів, автентифікацією та авторизацією, забезпечуючи дотримання відповідних політик безпеки.
- SSO (Single Sign-On): SSO дозволяє користувачам увійти до різних систем та додатків за допомогою одного набору облікових даних. Це полегшує діяльність користувачів та зменшує ризики витоку паролів, забезпечуючи єдиний та безпечний вхід.
- MFA (Multi-Factor Authentication): MFA використовує кілька методів автентифікації, таких як пароль, токени або біометричні дані, для забезпечення додаткового рівня безпеки. MFA є ефективним інструментом для захисту облікових записів та важливих ресурсів від витоку.
- RBAC (Role-Based Access Control): RBAC визначає доступ користувача на основі його ролі в організації. Кожна роль має визначені повноваження, і користувачі отримують доступ лише до тих ресурсів, які необхідні для виконання відповідних повноважень.
- PAM (Privileged Access Management): PAM забезпечує контроль та моніторинг доступу до привілейованих облікових записів, які мають розширені права. Цей метод є важливим фактором для захисту від внутрішніх та зовнішніх загроз, пов'язаних з використанням

привілейованих облікових записів з метою отримання доступу до надважливих активів компанії.

- ABAC (Attribute-Based Access Control) – метод управління доступом, який базується на атрибутах користувачів, ресурсів та середовища, а не на статичних ролях, як у RBAC. ABAC забезпечує динамічний і гнучкий підхід до управління доступом, дозволяючи організаціям адаптувати політики доступу відповідно до конкретних умов.
- UEBA (User and Entity Behavior Analytics): UEBA використовує аналіз поведінки користувачів та сутностей для виявлення аномальної активності та потенційних загроз безпеці. Цей метод спрямований на вчасне виявлення несанкціонованих дій на основі аналізу змін у звичайній поведінці[1].
- SIEM (Security Information and Event Management): системи SIEM здійснюють збір, аналіз та моніторинг дій користувачів в системі. Це дозволяє виявляти аномалії, визначати та відбивати спроби несанкціонованого доступу до ресурсів, вести журнали подій для подальшого аналізу.
- DLP (Data Loss Prevention): Системи DLP визначають, відстежують та контролюють рух конфіденційної інформації в мережі комп'ютерної системи. Це включає в себе заходи для запобігання витоку даних через мережу або інші зони ризику.
- ZTSM (Zero Trust Security Model): модель безпеки "Zero Trust" передбачає, що навіть внутрішня мережа повинна бути розглянута як ненадійна, і кожен запит на доступ повинен бути перевірений, незалежно від місця, з якого він надходить.
- CASB (Cloud Access Security Broker): CASB дозволяє контролювати та відстежувати доступ до хмарних сервісів. Він надає можливість управління політиками безпеки, проведення аудиту та захисту конфіденційності активів, що перебувають в хмарних середовищах, у тому числі безпеку даних, коли вони переміщаються в хмару[2].

– VM (Vulnerability Management): даний метод управління вразливостями фокусується на пошуку слабких місць та прогалин у різних компонентах системи, активно реагуючи на потенційні ризики та забезпечуючи їх виправлення.

Сучасні засоби управління користувачами в комп'ютерних системах підприємств включають різноманітні рішення, спрямовані на керування ідентифікацією, автентифікацією, авторизацією та загальною безпекою доступу до інформаційних ресурсів. Ось огляд та опис деяких з них:

Ось більш детальний опис кожного з перелічених засобів:

- Microsoft Azure Active Directory (Azure AD) – комплексне рішення від Microsoft для управління ідентифікацією та доступом у хмарному середовищі. Azure AD дозволяє організаціям централізовано контролювати доступ до ресурсів і користувачів, забезпечуючи високий рівень безпеки завдяки інтеграції з багатофакторною автентифікацією (MFA) та SSO (єдиним входом) для хмарних та локальних додатків. Azure AD підтримує інтеграцію з різними платформами, включаючи Office 365, та забезпечує доступ до аналітики безпеки, що дозволяє виявляти підозрілу активність і автоматично реагувати на неї.
- Okta – хмарна платформа ідентифікації та управління доступом, яка забезпечує зручність автентифікації та авторизації для користувачів в різних середовищах. Okta пропонує MFA, єдиний вхід (SSO), а також інтеграцію з великою кількістю додатків, що робить її підходящою для організацій з різнорідними комп’ютерними системами. Сервіс дозволяє підприємствам автоматизувати керування доступом та підтримувати політики безпеки без складної налаштування та впровадження.
- CyberArk Privileged Access Management (PAM) – рішення від CyberArk спрямоване на контроль та захист привілейованих облікових записів, тобто тих, які мають високі рівні доступу до ресурсів організації. CyberArk PAM включає заходи з аудиту, контролю доступу та управління сесіями для адміністраторів, що значно знижує ризик

зловмисного або несанкціонованого доступу до критично важливих систем.

- ForgeRock Identity Gateway: пропонує гнучкі рішення для безпеки та контролю доступу до ресурсів, зокрема API та хмарних додатків. Identity Gateway від ForgeRock підтримує загальноприйняті стандарти безпеки (OAuth, OpenID Connect), забезпечуючи захист від несанкціонованого доступу. Це рішення також надає можливість використання політик доступу, які можна адаптувати до потреб конкретної організації[3].
- Symantec Data Loss Prevention (DLP) – система запобігання втраті даних, яка відслідковує, контролює та захищає конфіденційну інформацію в мережі. DLP ідентифікує дані, що підлягають захисту, і контролює їх рух між користувачами та системами, забезпечуючи виконання нормативних вимог та внутрішніх політик збереження даних.
- Cisco Identity Services Engine (ISE) – рішення для контролю доступу в мережі з можливістю автоматизації безпеки. Cisco ISE дозволяє створювати та впроваджувати політики доступу, налаштовані для окремих користувачів, пристроїв та мережевих сегментів. Завдяки цьому рішенню організації можуть автоматично обмежувати або надавати доступ в залежності від параметрів безпеки та характеристик пристроїв.
- Centrify Identity Services: надає комплексне рішення для безпеки ідентифікації та контролю доступу як для хмарових, так і для локальних середовищ. Centrify підтримує MFA, SSO, а також інтегрується з існуючими корпоративними інфраструктурами, забезпечуючи гнучке та централізоване керування доступом для підвищення рівня безпеки.
- IBM Security Identity Manager – програмне забезпечення для автоматизованого управління ідентифікацією та доступом. IBM Security Identity Manager підтримує функції надання, відкликання та перевірки доступу, а також дозволяє створювати складні ієрархії доступу для

організації та підтримувати відповідність корпоративним політикам безпеки.

- SailPoint IdentityNow – система управління ідентифікацією, що пропонує автоматизовані функції управління ролями та доступом, а також політики безпеки. Ця платформа підходить для великих підприємств, що прагнуть зменшити складність процесів управління ідентифікацією, інтегруючись з різними хмарними та локальними додатками.
- OneLogin – хмарна платформа для ідентифікації та управління доступом, яка пропонує можливості MFA, єдиного входу (SSO) та інтеграції з великою кількістю додатків. OneLogin зручний у використанні для компаній, що хочуть спростити процеси доступу для співробітників, підвищуючи рівень безпеки та контролю.

Ці засоби надають підприємствам різноманітні інструменти для забезпечення безпеки, ефективного управління ідентифікацією та контролю доступу. Вибір конкретного рішення залежить від потреб компанії, розміру та складності її інфраструктури, а також вимог до безпеки та сумісності.

1.2 Аналіз сучасних методів управління користувачами

Аналіз полягає в розгляді та усвідомленні основних характеристик, переваг і недоліків кожного методу задля їхнього продуктивного застосування. Розглянемо в даному контексті більш детально кожен із методів, зазначених попередньо:

IAM:

Переваги:

- Централізоване управління ідентифікацією та авторизацією: надає один центральний пункт для управління ідентифікаційними даними

користувачів і їхніми правами доступу. Це спрощує адміністрування та дозволяє ефективно використовувати доступ до різних ресурсів.

– Високий рівень безпеки та забезпечення дотримання політик безпеки: визначає та контролює права доступу користувачів, забезпечуючи дотримання внутрішніх та зовнішніх політик безпеки. Це зменшує ризики порушень безпеки та несанкціонованого доступу.
– Можливість динамічного управління правами доступу: дозволяє динамічно адаптувати права доступу користувачів в залежності від їхньої ролі, статусу чи контексту. Це дозволяє ефективно реагувати на оновлення потреб підприємства.

Недоліки:

– Складність впровадження та адміністрування: реалізація системи IAM може вимагати значних зусиль та часу. Крім того, адміністрування потребує спеціалізованих навичок, що може бути гальмуючим фактором для деяких організацій.
– Великі витрати на інфраструктуру та навчання персоналу: упровадження та підтримка системи IAM може вимагати значних фінансових ресурсів на інфраструктуру та навчання персоналу, що ускладнює її реалізацію у випадках малих підприємств.

SSO:

Переваги:

– Зручність для користувачів та підвищення продуктивності: користувачам не потрібно повторно вводити облікові дані для кожного ресурсу, що збільшує зручність роботи та підвищує їхню продуктивність.
– Зменшення ризику витоку паролів та підвищення безпеки: зменшує кількість паролів, які користувачам потрібно пам'ятати, та мінімізує ризик витоку паролів, оскільки вони вводяться лише один раз.
– Ефективне управління обліковими записами: адміністраторам легше управляти однією точкою входу, що спрощує процеси управління обліковими записами.

Недоліки:

– Специфічні вимоги до інтеграції з різними системами: інтеграція SSO досить складна, оскільки вимоги до сумісності можуть відрізнятися між різними системами та сервісами.
– Можливість збільшення ризику при компрометації облікових даних: якщо облікові дані користувача компрометовані, то зловмисник може отримати доступ до усіх підключених ресурсів, що збільшує потенційні ризики несанкціонованого доступу.

MFA:

Переваги:

– Забезпечення додаткового рівня безпеки: використання кількох методів автентифікації робить процес входу більш безпечним, оскільки потрібно подавати декілька форм ідентифікації.
– Зменшення ризику несанкціонованого доступу: якщо один елемент автентифікації компрометований, інші фактори залишаються для захисту від несанкціонованого доступу.
– Різноманітні методи автентифікації для вибору: користувачі можуть вибирати із різних методів, таких як пароль, токени або біометричні дані, в залежності від їхніх вподобань та потреб системи.

Недоліки:

– Можливість створення додаткових труднощів для користувачів: певним робітникам може виявитися не зручно використовувати додаткові методи автентифікації, особливо якщо вони не є інтуїтивно зрозумілими.
– Збільшення витрат на реалізацію та обслуговування: впровадження та підтримка MFA може вимагати додаткових витрат на інфраструктуру та обслуговування, що може бути важливим фактором для деяких компаній.

RBAC:

Переваги:

– Простота управління доступом на основі ролей: дозволяє легко призначати та видаляти доступ на основі ролей, спрощуючи адміністрування та зменшуючи ризик помилок в наданні прав.
– Зменшення ризику витоку конфіденційної інформації: ролі дозволяють точно визначити, які ресурси може використовувати користувач, що допомагає уникнути несанкціонованого доступу до конфіденційних даних.
– Легке адміністрування доступу: Додавання або вилучення користувачів від ролі виконується легко, забезпечуючи швидке та ефективне управління доступом.

Недоліки:

– Обмеженість управління індивідуальними правами користувачів: RBAC може бути недостатньо гнучким для ситуацій, де потрібне індивідуальне налаштування прав доступу для кожного користувача.
– Потреба у ретельному визначенні ролей та їхніх повноважень: необхідні детальне планування та дефініції ролей, щоб уникнути неправильного надання доступу чи забруднення прав доступу.

PAM:

Переваги:

– Контроль та моніторинг доступу до привілейованих облікових записів: забезпечує контроль за доступом до облікових записів з розширеними правами, обмежуючи їх використання для невповноважених осіб.
– Зменшення ризику несанкціонованого доступу до критичних ресурсів: мінімізує ризик витоку конфіденційної інформації через привілейовані облікові записи. Це забезпечує ефективний контроль над діями адміністраторів та інших привілейованих користувачів.
– Деталізований аудит та підзвітність дій: Забезпечує докладні фіксацію та збереження даних усіх подій, пов'язаних з привілейованими обліковими записами. Допомагає виявляти та реагувати на потенційні загрози шляхом аналізу наявних даних.

Недоліки:

– Складність впровадження та конфігурації: вимагає ретельних налаштування та інтеграції з існуючою інфраструктурою. Потребує ефективного планування та впровадження для мінімізації впливу на робочі процеси.
– Потреба у високому рівні обізнаності персоналу: вимагає пристойного рівня кваліфікації та обізнаності адміністраторів та фахівців із безпеки, тому необхідно проводити відповідні навчання персоналу для ефективного використання PAM.

ABAC:

Переваги:

− Завдяки використанню атрибутів, може автоматично регулювати доступ до ресурсів залежно від контексту, що знижує ризики несанкціонованого доступу, тобто керувати повноваженнями користувачів динамічно.
− Оскільки політики базуються на атрибутах, адміністратори можуть уникати створення численних ролей для кожного сценарію доступу, спрощуючи управління, що скорочує адміністративні витрати.

Недоліки:

− Розробка і впровадження політик може бути більш трудомістким процесом, оскільки необхідно враховувати велику кількість атрибутів і можливих умов, до того ж відсутні усталені стандарти, що ускладнює інтеграцію з існуючими системами.
− Характерні підвищені вимоги до продуктивності, оскільки оцінка політик вимагає більше ресурсів і часу на виконання, особливо при великій кількості атрибутів і запитів на доступ.

UEBA:

Переваги:

– Високий рівень автоматизації у виявленні несанкціонованих дій: використовує алгоритми та машинне навчання для автоматичного виявлення відхилень від типової користувацької поведінки.

– Покращення часу виявлення та реакції на загрози: дозволяє виявляти загрози в реальному часі, що дозволяє швидше реагувати та мінімізувати можливі наслідки.

Недоліки:

– Високі витрати на впровадження та обслуговування: впровадження та підтримка системи UEBA може бути витратною, особливо для невеликих підприємств.
– Можливість виникнення помилкових спрацювань системи: алгоритми аналізу поведінки можуть стикатися з викликами точності, що призводить до хибно-позитивних чи хибно-негативних результатів.

SIEM:

Переваги:

– Збір, аналіз та моніторинг подій для виявлення загроз: надає можливість реагувати на загрози майже в реальному часі, аналізуючи велику кількість подій. Аналітика допомагає виявляти аномальні шаблони та потенційні загрози.
– Великий обсяг аналітики та інформації для прийняття рішень: масивний набір даних дозволяє деталізовано аналізувати події та приймати обґрунтовані рішення, у тому числі й прогнозувати та уникати майбутніх загроз.
– Централізована система аудиту та підзвітності: централізований доступ до журналів подій полегшує аудит та вивчення історії подій та, загалом, виступає важливим компонентом для оперативного реагування на загрози.

Недоліки:

– Велика кількість хибно-позитивних або хибно-негативних результатів: Можлива велика кількість фальшивих спрацювань, що потребує додаткового вдосконалення алгоритмів. Для уникнення хибних сигналів необхідне постійне налаштування системи.
– Високі витрати на обладнання та обслуговування: збір та зберігання великого обсягу даних вимагає потужних обчислювальних ресурсів.

– Складність обслуговування: вимагає кваліфікованого персоналу для ефективного обслуговування та аналізу даних.

DLP:

Переваги:

– Управління конфіденційною інформацією: дозволяє підприємствам визначати та класифікувати конфіденційні дані, забезпечуючи точний контроль над їх рухом та використанням.
– Захист від витоку даних через мережу та інші канали: може блокувати або відстежувати спроби передачі конфіденційної інформації через мережу, електронну пошту та інші канали зв'язку шляхом використання шифрування та маскування для захисту важливої інформації під час передачі.
– Відповідність регуляторам захисту даних: допомагає підприємствам виконувати вимоги різних стандартів та законодавства, які стосуються захисту конфіденційної інформації.

Недоліки:

– Потреба в ретельній конфігурації та обслуговуванні: вимагає скрупульозної конфігурації для відповідності специфічним потребам підприємства. Регулярні оновлення та технічна підтримка є важливими для забезпечення ефективності DLP.
– Можливість блокування легітимного обміну даними: недостатнє налаштування політик може призводити до блокування легітимного обміну даними, що негативно впливає на продуктивність та співпрацю.

ZTSM:

Переваги:

– Всебічний підхід до безпеки: у парадигмі даної концепції навіть внутрішня мережа може бути небезпечною, тому всі запити на доступ обробляються з особливою увагою. Будь-які сутності (користувачі, пристрої) перевіряються перед отриманням доступу.
– Активна перевірка кожного запиту на доступ: модель дозволяє вчасно виявляти та обмежувати атаки, оскільки кожен запит перевіряється та

автентифікується. Навіть інсайдерські загрози виявляються шляхом постійного моніторингу дій.

Недоліки:

– Складність впровадження та адаптації для існуючої інфраструктури: для переважної більшості компаній дана модель може вимагати серйозної перебудови існуючих мереж та процесів.
– Можливість виникнення труднощів для користувачів: відповідні налаштування безпеки можуть призвести до збільшення кількості автентифікаційних етапів та обмежень, що потенційно ускладнює взаємодію із системою для працівників.

CASB:

Переваги:

– Захист конфіденційності даних в хмарних середовищах: протидіє несанкціонованого доступу до конфіденційні дані та витокам в хмарних ресурсах, шифрує та контролює обмін даними в хмарних сервісах.
– Управління політиками безпеки для хмарних ресурсів: дозволяє встановлювати та дотримувати політики безпеки для доступу до хмарних сервісів. Може автоматично реагувати на порушення безпеки за заданими правилами.

Недоліки:

– Потреба в інтеграції з різними хмаровими платформами: вимагає стандартизації та інтеграції з різними хмаровими постачальниками. Специфічні адаптації можуть бути важким завданням із точки зору сумісності.
– Вартість реалізації та обслуговування: упровадження CASB може бути неефективним з точки зору витрат, оскільки вимагає додаткових вкладень на обслуговування та навчання персоналу.

VM:

Переваги:

– Забезпечення безпеки систем через пошук та виправлення вразливостей: активно виявляє та реагує на потенційні слабкі місця у

системах. Забезпечує систематичний аналіз та виправлення вразливостей для підвищення загальної безпеки.

- Моніторинг та реакція на потенційні ризики: постійно відстежує зміни в оточенні та швидко реагує на нові загрози. Забезпечує моніторинг у режимі реального часу для негайної реакції на посталі загрози.
- Відповідь на оновлення стандартів безпеки: забезпечує актуалізацію та адаптацію до нових вимог та технологій згідно з відповідними стандартами безпеки.

Недоліки:

- Потреба в постійному оновленні та тестуванні: вимагає систематичних випробувань для виявлення нових вразливостей та методів атак, тестування забезпечення та визначення впливу на виробничу інфраструктуру.
- Велика кількість хибно-позитивних результатів: автоматизовані сканування можуть призводити до невірних визначень вразливостей та загроз. Необхідні ретельні аналіз та перевірка знайдених проблем.
- Зменшення ризику внутрішніх загроз та руху вірусів: Кожен елемент інфраструктури ізолюється, зменшуючи можливість поширення вірусів та атак всередині мережі. Це призводить до мінімізації можливостей для несанкціонованого доступу до критичних ресурсів.

Даний порівняльний аналіз демонструє, що вибір певного підходу до управління користувачами має ґрунтуватися в залежності від конкретних вимог, специфіки бізнес-процесів, інфраструктури та стратегії безпеки підприємства. Важливо враховувати ці фактори при виборі оптимального рішення для конкретного випадку. Зокрема це може включати комбінацію різних підходів для досягнення комплексного та ефективного управління безпекою та доступом користувачів.

1.3 Аналіз основних викликів та проблем, пов'язаних із управлінням користувачами

Управління користувачами в комп'ютерних системах стикається з рядом викликів та проблем, які вимагають постійного поліпшення та удосконалення стратегій та підходів задля зменшення відповідних ризиків. Декілька основних аспектів, які можуть виникнути в контексті ускладнення процесу управління користувачами, включають:

- Складність ідентифікації та автентифікації: ускладнення процесу ідентифікації та автентифікації користувачів може виникнути через зростання кількості систем та застосунків, що використовуються в організації. Це може призвести до великої кількості облікових записів та паролів, а також збільшити ризик використання слабких методів автентифікації.
- Ризик витоку даних: з огляду на зростання обсягу конфіденційної інформації та зловживання доступом, існує постійний ризик витоку даних. Навіть із застосуванням сучасних методів управління користувачами, недостатній контроль може призвести до витоку корпоративних даних непублічного характеру.
- Людський фактор та соціальна інженерія: даний аспект є однією з найбільших вразливостей у системах управління користувачами. Атаки за допомогою соціальної інженерії можуть оминути технічні засоби безпеки, отримуючи доступ до облікових даних унаслідок недбалості або неправильних дій співробітників.
- Комплексність управління привілеями: З упровадженням різноманітних ролей та прав доступу, керування привілеями може стати складним завданням. Недостатня чіткість визначення ролей та прав може призвести до надмірного або, навпаки, недостатнього рівня доступу, тобто помилкового розподілення повноважень між користувачами.

- Спроможність адаптації до змін: мінливість в організаційній структурі, додавання нових систем або перехід до хмарних рішень можу призвести до відповідних викликів щодо достатньої гнучкості системи. Системи повинні бути здатні адаптуватися до нових вимог та технологічних інновацій, щоб забезпечувати ефективне управління користувачами.
- Використання мобільних пристроїв: з розвитком технологій та зростанням кількості мобільних пристроїв в організаціях, управління безпекою та доступом до цих пристроїв стає складнішим завданням. Різноманітність платформ та операційних систем може вимагати використання різних засобів та політик управління.
- Юридична відповідність системи: управління користувачами повинно задовольняти вимогам нормативних документів, законодавчих актів та стандартів безпеки. Забезпечення відповідності може вимагати великих зусиль у створенні та впроваджені політик та аудиту безпеки.
- Вартість та ресурси: упровадження та підтримання систем управління користувачами може бути дорогим завданням, особливо для малих підприємств. Вартість включає в себе як витрати на придбання та впровадження програмних продуктів, так і витрати на навчання персоналу та підтримку систем.
- Нехтування заходами безпеки з боку працівників: важливо формувати в організації культуру безпеки серед користувачів, щоб уникнути недбалого ставлення до правил та недостатнього усвідомлення ризиків. Навіть найсучасніші технічні рішення не можуть гарантувати відповідальності та свідомого ставлення до безпеки з боку персоналу.
- Робота з віддаленими користувачами: із розвитком технологій та зростанням кількості віддалених робочих місць управління користувачами стає більш складним завданням. Необхідно забезпечити безпеку та ефективне управління доступом до систем для користувачів, які працюють з різних місць світу.
- Взаємодія з постачальниками та партнерами: у випадках, коли підприємство має кооперувати з різними партнерами та

постачальниками, управління доступом до власних ресурсів може стати неабияким викликом. Необхідно встановити конкретні процедури щодо надання доступу іншим організаціям до потрібних активів системи підприємства.

- Розвиток технологій біометричної автентифікації: із впровадженням технологій даного роду виникають все нові та нові виклики, такі як забезпечення приватності, вірогідність похибки автентифікації та збереження відповідних факторів робітників у системі. Компанії повинні враховувати етичні та юридичні аспекти використання таких технологій.
- Забезпечення системи прозорості та аудиту: наявність ефективної системи аудиту та прозорості управління користувачами дозволяє вчасно виявляти та реагувати на можливі порушення безпеки. Забезпечення консистентного та достовірного аудиту може бути складним випробуванням.
- Інтеграція та міграція даних: при зміні або оновленні систем управління користувачами може виникнути проблема інтеграції з існуючими даними та міграції користувацької інформації. Важливо уникати втрати даних та забезпечити безперебійну роботу під час переходу на нові рішення й компоненти.

Усі вище згадані виклики та проблеми показують, що управління користувачами є комплексним питанням, яке вимагає уваги до технічних, організаційних та людських факторів задля забезпечення ефективності, надійності та гнучкості системи.

Окрім того, вони відображають динаміку сучасного інформаційного середовища, яке вимагає постійного вдосконалення стратегій та технологій, щоб підтримувати стабільно позитивне функціонування комп'ютерних систем підприємств.

1.4 Системний підхід до управління користувачами на підприємствах

Системний підхід до управління користувачами на підприємствах визначається комплексним поглядом на організацію як на систему, складену з частин, кожна з яких має свої власні вимоги й інтереси, та проявляється в розумінні, оцінці взаємодії цих компонентів та об'єднанні їх у єдине ціле. Такий підхід спрямований на забезпечення ефективного та безпечного управління доступом до інформаційних ресурсів підприємства[4].

Нижче подано деталізований опис сучасних принципів системного підходу до управління користувачами комп’ютерних систем підприємств:

- Цілісність системи: системний підхід до управління користувачами підприємства базується на принципі цілісності системи. Це передбачає не лише управління окремими аспектами, але й глибокий аналіз взаємодії всіх компонентів, включаючи апаратне й програмне забезпечення, мережеві структури та процеси обробки інформації. Забезпечуючи єдиний образний погляд на всю інформаційну систему, управління користувачами стає не лише окремою частиною, але і стратегічним інструментом для досягнення глобальних бізнес-цілей.
- Інтеграція з іншими процесами: системний підхід вимагає високого рівня інтеграції управління користувачами з іншими ключовими процесами підприємства. Це включає в себе узгоджену взаємодію з управлінням ресурсами, безпекою, аудитом та іншими аспектами. Наприклад, інформація про користувачів застосовується для автоматизації процесів призначення завдань, надання доступу до ресурсів та відслідковування використання інформаційних активів.
- Централізоване управління ідентифікацією та автентифікацією: даний аспект виступає як клавішний елемент системного підходу, що включає в себе, зокрема, встановлення стандартів для перевірки особистості користувачів відповідно до політик безпеки підприємства. також Це забезпечує єдиний точковий вхід та керування авторизацією для всіх

інформаційних ресурсів, спрощуючи процес адміністрування та зменшуючи ризики несанкціонованого доступу.

- Управління ролями та правами доступу: управління ролями та правами користувачів у системі впроваджується з метою визначення чіткої структури повноважень на основі ролей та функціональних обов'язків. Відповідно до цього принципу, користувачі отримують доступ лише до необхідної інформації та функціоналу, забезпечуючи ефективний контроль над безпекою та конфіденційністю даних.
- Автоматизація управління життєвим циклом облікових записів: дана складова стає необхідною частиною системного підходу. Вона означає використання інтегрованих і автоматизованих інструментів для ефективного створення, зміни, блокування та видалення облікових записів, ураховуючи внутрішні та зовнішні зміни, такі як підвищення, пониження чи звільнення співробітника.
- Системи моніторингу та аудиту: впровадження систем моніторингу та аудиту в систему управління користувачами дозволяє реально відстежувати дії користувачів та реагувати на події в режимі реального часу. Це не лише виявляє потенційні загрози безпеці, але і слугує основою для подальшого аналізу, розслідування подій та вдосконалення стратегій безпеки.
- Підтримка безпеки та захищеності інформації: ці фактори є надважливими складовими системного підходу, оскільки включають у себе впровадження заходів безпеки, таких як шифрування, моделі та механізми контролю доступу, виявлення вторгнень та засоби захисту від витоку даних. Усі ці елементи розглядаються в комплексі, забезпечуючи повноцінний захист інформаційних активів.
- Постійні підтримка та оновлення: системний підхід передбачає неперервну підтримку та оновлення систем управління користувачами. Це включає в себе відслідковування нових технологій, виявлення та виправлення вразливостей, а також підтримку змін в бізнес-процесах та вимогах безпеки.

– Тренінги з кібергігієни для користувачів: однією із складових системного підходу також є розробка та впровадження програм тренінгу та підвищення кваліфікації й свідомості користувачів. Це означає не лише технічне навчання, але і створення культури безпеки, де працівники розуміють свою роль у підтримці безпеки інформації та активно долучаються до цього процесу.
– Штучний інтелект та застосування аналітичних інструментів: у системі управління користувачами даний аспект визначається як важлива частина системного підходу сьогодення. Автоматичне виявлення загроз, прогнозування подій та реакція на них у реальному часі стають можливими завдяки інтелектуальному аналізу великої кількості даних, що полегшує адаптацію до нових загроз та важливих змін в інформаційному середовищі підприємства.

Отже, ураховуючи наведені принципи, системний підхід до управління користувачами сприяє підвищенню безпеки, ефективності та спрощенню процесів адміністрування на підприємствах. Він враховує всі аспекти управління користувачами як систему, де кожна складова взаємодіє для досягнення спільної мети – забезпечення безпеки та ефективності використання інформаційних ресурсів.

2 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ УПРАВЛІННЯ КОРИСТУВАЧАМИ

2.1 Класифікація методів управління користувачами

Управління користувачами передбачає широкий спектр заходів, серед яких ідентифікація, автентифікація, авторизація, моніторинг та аудит активності користувачів, а також управління правами доступу до ресурсів. Вибір відповідного методу або інструменту для цих завдань залежить від специфічних потреб організації, рівня безпеки, який необхідно забезпечити, та доступних ресурсів.

У цьому контексті класифікація методів управління користувачами є необхідною для того, щоб розуміти, як різні методи взаємодіють між собою і відповідають конкретним вимогам безпеки підприємства. Крім того, класифікація дозволяє полегшити процес оцінки ефективності та вибору методів для конкретних випадків, зокрема розглядати управління користувачами не лише як частину технічної безпеки, але і як невід'ємну складову бізнес-процесів компанії перед їх впровадженням в корпоративну систему.

Метою цього розділу є систематизація методів управління користувачами та оцінка їх ефективності, надійності та доцільності у різних сценаріях. Для цього необхідно класифікувати існуючі методи на основі їх функціональних можливостей, а також визначити критерії оцінки для кожної з підгруп. Така класифікація не лише спрощує розуміння функціоналу кожного інструменту, але й дозволяє проводити більш наочне порівняння між ними за чіткими параметрами.

Серед розглянутих попередньо методів управління користувачами доцільно буде виділити наступні 4 підгрупи:

- Методи ідентифікації та автентифікації: забезпечують перевірку особи користувача перед наданням доступу до ресурсів (IAM (Identity and Access Management), SSO (Single Sign-On), MFA (Multi-Factor Authentication)).

- Методи управління доступом: регулюють рівень доступу до інформаційних ресурсів на основі ролей, атрибутів або привілеїв користувачів (RBAC (Role-Based Access Control), PAM (Privileged Access Management), ABAC (Attribute-Based Access Control)).
- Методи аналізу та управління безпекою: дозволяють аналізувати активність користувачів і виявляти аномалії чи загрози в режимі реального часу (UEBA (User and Entity Behavior Analytics), SIEM (Security Information and Event Management), DLP (Data Loss Prevention)).
- Моделі безпеки: комплексні стратегії для побудови захисту системи, які об'єднують різні методи та інструменти для забезпечення цілісного підходу до безпеки (ZTSM (Zero Trust Security Model), CASB (Cloud Access Security Broker), VM (Vulnerability Management)).

Ці категорії дозволяють охопити всі ключові аспекти управління користувачами, починаючи від автентифікації і закінчуючи аналітичним моніторингом та попередженням загроз. Таким чином, дана класифікація методів є основою для їх подальшого дослідження за низкою специфічних критеріїв, безпосередньо пов’язаних з особливостями кожної з підгруп.

2.2 Визначення критеріїв для оцінки методів управління користувачами

Для всебічного дослідження методів управління користувачами в рамках підгруп важливо визначити універсальні критерії, які можна застосовувати для оцінки та відповідного порівняння всіх методів в кожній підгрупі. Нижче наведено набори даних критеріїв для кожної з підгруп.

Методи ідентифікації та автентифікації (IAM, SSO, MFA):

- Ефективність захисту: цей критерій визначає здатність методів запобігати несанкціонованому доступу до системи. Вимірюється через

аналіз кількості зафіксованих спроб вторгнення за певний період та їх успішність, тобто збором статистики про спроби входу в систему, а також проведенням тестування безпеки для виявлення вразливостей у процесах автентифікації;

- Час обробки запиту: вимірює середній час, необхідний для завершення процесу автентифікації користувача. Це важливий критерій, оскільки швидкість автентифікації впливає на загальний досвід користувача. Доцільне використання таймерів для вимірювання часу, що проходить від ініціації запиту автентифікації до його завершення. Дані можна збирати під час пілотного тестування;
- Адаптивність до ризиків: визначає, наскільки добре система може змінювати рівень захисту в залежності від контексту, таких як геолокація користувача або зміни в поведінці. Можна оцінювати шляхом аналізу історичних даних про входи користувачів для виявлення аномальних шаблонів поведінки. Система може автоматично підвищувати рівень автентифікації (наприклад, запитувати двофакторну автентифікацію) при виявленні незвичних дій;
- Рівень зручності: Оцінює, наскільки просто і зрозуміло користувачам проходити процес автентифікації через наявний інтерфейс, ясність інструкцій та загальний комфорт роботи. Наприклад, за допомогою ведення журналу всіх дій користувачів у системі можна проводити аналізу їх поведінки для виявлення можливих проблем у користувацькому досвіді, які потребують вирішення;
- Здатність до інтеграції: вимірює, наскільки легко метод може інтегруватися з іншими системами та сервісами, такими як ERP (Enterprise Resource Planning) або CRM (Customer Relationship Management), що дозволяє забезпечити безперебійну роботу бізнес-процесів. Доречні оцінка документації API та можливостей інтеграції, тестування інтеграції з конкретними системами, щоб виявити потенційні проблеми.

- Стабільність: оцінює частоту збоїв та відмов у системі. Це критично важливо для підтримки безперервності бізнесу, оскільки відмови можуть призвести до втрат у роботі. Характерні моніторинг системи в реальному часі та ведення журналу відмов, аналіз тривалості безвідмовної роботи (MTBF) та часу відновлення після збоїв (MTTR).

Методи управління доступом (RBAC, PAM, ABAC):

- Гнучкість: вимірює, наскільки легко система може налаштовувати правила доступу для різних категорій користувачів. Це дозволяє організаціям швидко адаптуватися до змін у структурах команд. Доцільне тестування налаштувань ролей та прав доступу в реальному часі, щоб визначити, як швидко та ефективно можна внести зміни в конфігурацію.
- Сумісність з політиками безпеки: оцінює, наскільки метод відповідає внутрішнім політикам безпеки організації та зовнішнім нормативним вимогам шляхом проведення аудиту, щоб перевірити відповідність системи вимогам. Можливе створення сценаріїв тестування, що перевіряють дотримання політик.
- Автоматизація процесів: визначає, наскільки метод може автоматизувати зміни в правах доступу на основі ролей та поведінки користувачів. Це допомагає зменшити адміністративне навантаження. Суть полягає в аналізі системи на предмет можливостей автоматизації, включаючи сценарії автоматичного оновлення прав доступу, що базуються на змінах у статусі користувача або в їхніх завданнях.
- Можливість моніторингу: оцінює, наскільки детально система веде журнал дій користувачів, що є важливим для аудиту та виявлення аномалій. Включає, зокрема, визначення структур журналів, які повинні містити важливу інформацію про дії користувачів, а також проведення тестування для перевірки наявності та доступності цих даних.
- Простота адміністрування: вимірює, наскільки легко адміністраторам управляти правами доступу, включаючи можливості масового

оновлення прав, оцінку інтерфейсу адміністратора, а також проведення опитувань серед адміністраторів для виявлення можливих труднощів.

- Відповідність стандартам безпеки: оцінює, наскільки метод відповідає загальноприйнятим стандартам безпеки, таким як NIST, що підтверджує його ефективність та безпечність. Реалізовується проведення аудитів для перевірки відповідності, включаючи наявність сертифікатів.

Методи аналізу та управління безпекою (UEBA, SIEM, DLP):

- Точність виявлення загроз: оцінює здатність системи виявляти загрози, враховуючи кількість хибно позитивних та хибно негативних спрацьовувань. Це важливий критерій для визначення надійності системи. Можливе проведення тестування з використанням наборів даних, що містять відомі загрози, і аналіз результатів на предмет виявлення коректності спрацьовувань.
- Швидкість реагування: вимірює, наскільки швидко система може реагувати на виявлені загрози, включаючи автоматичні дії, такі як блокування доступу, надсилання повідомлень адміністраторам, моніторинг часу від виявлення загрози до реагування, або ж проведення тестів на симуляцію атак для перевірки швидкості реакції.
- Масштабованість: оцінює, наскільки система може впоратися з збільшенням обсягу даних і загроз без втрати продуктивності, наприклад, за допомогою стрес-тестування системи для виявлення її можливостей обробки великої кількості подій.
- Сумісність з існуючими системами: визначає, наскільки легко система інтегрується з іншими інструментами безпеки, такими як антивірусні програми, міжмережеві екрани та системи моніторингу.
- Аналіз даних у реальному часі: оцінює можливості системи для обробки та аналізу даних у режимі реального часу, що важливо для виявлення загроз у момент їх виникнення. Може реалізовуватися у вигляді моніторингу продуктивності системи під час обробки даних у реальному часі, включаючи затримки та оброблені події.

- Доступність звітності: визначає, наскільки просто система генерує звіти про події безпеки та виявлені загрози, що важливо для аналізу та управлінських рішень. Необхідна оцінка інтерфейсу генерації звітів та проведення тестування на предмет доступності необхідних даних для аналізу.

Моделі безпеки (ZTSM, CASB, VM):

- Принципи мінімального доступу: оцінює, наскільки система реалізує політики надання мінімально необхідного доступу до ресурсів, що зменшує ризик несанкціонованого доступу. Застосовується перевірка налаштувань доступу та прав для різних ролей, а також проведення аудиту для підтвердження відповідності політикам.
- Аудит та контроль: вимірює, наскільки легко системі проводити аудит дій користувачів та контролювати доступ до ресурсів, що є важливим для забезпечення безпеки. Здійснюється аналіз можливостей ведення журналів та їх доступності для аудиторів, проведення тестів для перевірки наявності всіх необхідних даних.
- Аналіз вразливостей: оцінює здатність системи виявляти та усувати вразливості в інфраструктурі, що критично важливо для запобігання можливим загрозам. Характерне регулярне проведення сканувань на вразливості з використанням відповідних інструментів, а також моніторинг результатів для виявлення проблем.
- Гнучкість архітектури: визначає, наскільки система може адаптуватися до змін у середовищі безпеки та бізнесу, що важливо для підтримки актуальності та ефективності. Проводиться оцінювання можливостей системи для швидкої модифікації архітектури при зміні умов або вимог.
- Можливість виявлення аномалій: оцінює здатність системи виявляти аномалії у поведінці користувачів, що може свідчити про спроби зловживання або атаки за допомогою тестування з використанням наборів даних, що містять приклади нормальної та аномальної поведінки, а також аналіз результатів для оцінки точності виявлення.

- Відповідність регуляторним вимогам: оцінює, наскільки система відповідає специфічним вимогам регуляторних органів, таких як GDPR, що може включати в себе обробку даних користувачів шляхом проведення аудитів для перевірки дотримання нормативних вимог, а також моніторингу змін у законодавстві для своєчасного реагування[5].

Ці критерії забезпечують структурований підхід до оцінки різних методів управління користувачами, дозволяючи їх всебічно порівняти та вибрати найбільш ефективні рішення для подальшої програмної реалізації. Вони сприяють також і розробці практичних рекомендацій щодо застосування й оптимізації управління користувачами в організаціях.

2.3 Дослідження методів ідентифікації та автентифікації

На даному етапі відбувається порівняння трьох наступних методів управління доступом — IAM, SSO та MFA — за низкою критично важливих критеріїв. Основною метою є оцінка їхньої ефективності, стабільності та доцільності використання в корпоративних середовищах з різними вимогами до безпеки, зручності та швидкості.

Ефективність захисту:

- IAM (8/10): дозволяє ефективно керувати політиками доступу, ролями користувачів та правами. Висока ефективність захисту обумовлена можливістю інтеграції різноманітних механізмів контролю доступу (RBAC, ABAC), сегментації ролей та чітким регламентуванням доступу до ресурсів залежно від критичності даних та рівня користувача. Недоліки полягають у можливих помилках під час конфігурації політик доступу, що може призвести до неправильного надання доступу або до створення зайво складних схем контролю, що потенційно може знизити захищеність. У великих організаціях, IAM системи можуть стати ключовим елементом для боротьби з внутрішніми загрозами, але для

досягнення максимальної ефективності потрібне чітке налаштування та регулярний моніторинг[6].

– SSO (7/10): дозволяє користувачам входити один раз для доступу до кількох систем, що полегшує управління обліковими записами та зменшує кількість запам'ятовуваних паролів. Проте цей метод має відносно слабкішу ефективність захисту через те, що компрометація єдиного набору облікових даних може надати доступ до всіх підключених систем. Щоб підвищити захист, SSO часто інтегрується з MFA, що створює додатковий шар безпеки (рис. 2.1)[7].

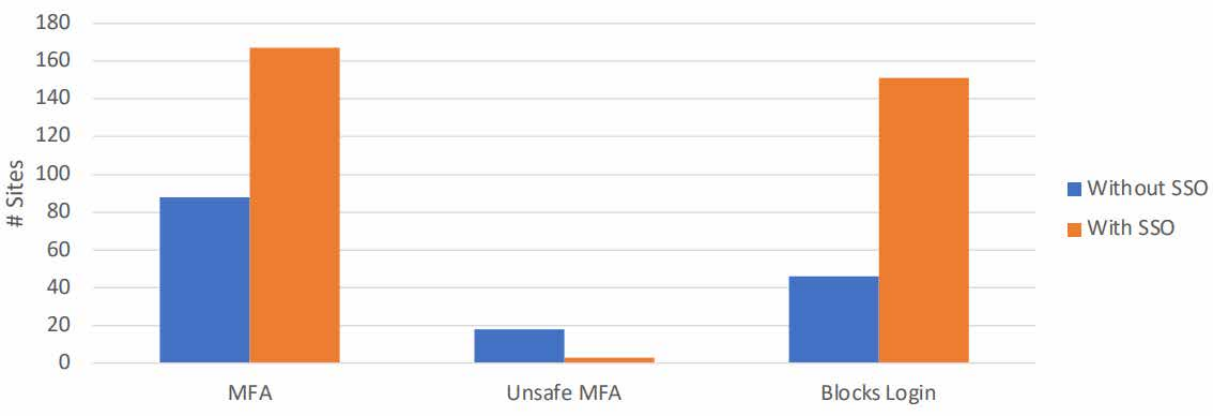

Рисунок 2.1 – Вплив SSO на безпеку авторизації в інтеграції з MFA

– MFA (9/10): багатофакторна автентифікація є одним з найефективніших методів захисту за рахунок поєднання кількох факторів одразу: наприклад, щось, що знає користувач (пароль), щось, що він має (телефон чи токен), та щось, чим він є (біометричні дані). Це робить атаку на обліковий запис значно складнішою. Навіть якщо зловмисник отримає пароль, йому також потрібен доступ до додаткового фактору, що різко знижує ймовірність успішної атаки[8].

Час обробки запиту:

– IAM (7/10): час обробки запитів може варіюватися залежно від складності конфігурації політик доступу та масштабів організації. У великих компаніях із багатьма ролями і політиками часу на обробку кожного запиту може бути більше через перевірку багаторівневих умов доступу. Однак сучасні IAM системи, такі як Microsoft Azure AD або Okta, оптимізовані для роботи з високим навантаженням, що дозволяє

знизити час обробки до мінімуму. У середньому, час обробки запиту може складати мілісекунди, проте в складних сценаріях він може зрости[9].

- SSO (9/10): дозволяє суттєво скоротити час автентифікації, оскільки користувачеві достатньо один раз увійти до системи, щоб отримати доступ до всіх підключених ресурсів. Це підвищує продуктивність працівників і знижує кількість повторних запитів для авторизації у системі. Проте, у випадку великої кількості підключених сервісів або недостатньо оптимізованої інфраструктури, можуть виникати незначні затримки при синхронізації між різними системами[10].
- MFA (6/10): багатофакторна автентифікація може значно сповільнити процес входу, оскільки наявні додаткові етапи авторизації (наприклад, введення коду з мобільного додатка або біометрична перевірка). Проте ці затримки зазвичай мінімальні і займають лише кілька секунд, що є прийнятною ціною за суттєве підвищення безпеки. Однак у випадку збою в роботі додаткового фактора (затримка повідомлення) або якщо користувач не має доступу до пристрою для автентифікації, це може значно ускладнити процес.

Адаптивність до ризиків:

- IAM (9/10): має високу адаптивність до ризиків завдяки можливості динамічно змінювати політики доступу, ураховуючи відповідний контекст. Наприклад, системи можуть виявляти підозрілу активність або спроби доступу з незвичайних локацій та відповідно змінювати рівень доступу або вимагати додаткової перевірки. Функції Azure AD дозволяють налаштовувати політики доступу залежно від оцінки ризиків. Також IAM системи інтегруються з такими методами, як UEBA, для виявлення аномалій у поведінці користувачів (рис. 2.2).

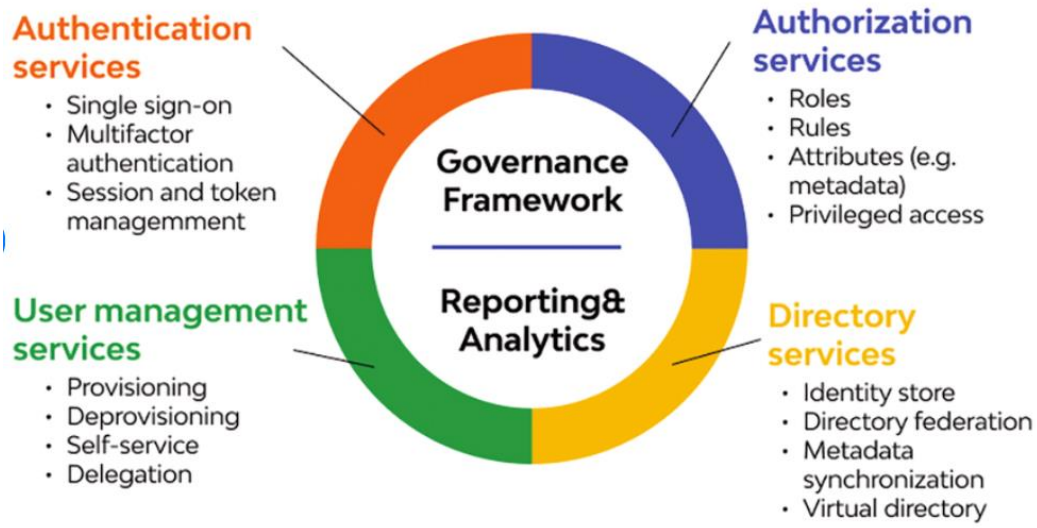

Рисунок 2.2 – Компоненти IAM

- SSO (6/10): без поєднання з додатковими засобами захисту, такими як MFA або поведінковий аналіз (UEBA), SSO має обмежену адаптивність до ризиків. Зокрема, якщо облікові дані компрометовані, ризик несанкціонованого доступу до багатьох систем зростає. Однак додаткові заходи, такі як інтеграція з IAM і MFA, можуть суттєво підвищити адаптивність до ризиків.
- MFA (9/10): відзначається високою адаптивністю, оскільки дозволяє додавати нові методи автентифікації або змінювати вимоги до безпеки в залежності від контексту. Наприклад, система може вимагати додаткового фактору автентифікації при спробі входу з незнайомого пристрою чи локації. Такий підхід дозволяє підвищити безпеку та знизити ймовірність компрометації[11].

Рівень зручності:

- IAM (6/10): для адміністраторів IAM може бути складною у налаштуванні через необхідність регулювання політик, ролей і прав доступу. Для кінцевих користувачів зручність залежить від інтеграції з іншими сервісами, таких як SSO чи MFA. Без додаткових рішень IAM може потребувати введення кількох наборів облікових даних для різних сервісів, що знижує рівень зручності[12].
- SSO (10/10): має найвищий рівень зручності для користувачів, оскільки дозволяє входити в систему лише один раз, надаючи доступ до всіх

необхідних ресурсів. Це знижує кількість паролів, які потрібно запам'ятовувати працівникам, та зменшує кількість процедур авторизації. Це особливо корисно в середовищах, де комп’ютерна система підприємства включає велику кількість різних підсистем[13].

- MFA (7/10): хоча MFA підвищує безпеку, вона може ускладнювати процес автентифікації, оскільки користувачам доводиться виконувати додаткові дії (наприклад, вводити код підтвердження з листа на електронній пошті). Більшість користувачів швидко звикає до цього процесу, але це може бути трохи незручним, особливо якщо виникають проблеми з доступом до додаткових факторів (наприклад, втрачений токен).

Здатність до інтеграції:

- IAM (9/10): системи IAM відзначаються високою здатністю до інтеграції з іншими корпоративними системами, включаючи служби каталогів (Active Directory), хмарні платформи (Azure, AWS) та системи управління доступом. Це робить IAM універсальним рішенням, яке можна налаштувати для різних сценаріїв використання. Більшість сучасних рішень підтримують відкриті стандарти автентифікації, такі як OpenID Connect, що робить їх ще більш гнучкими.
- SSO (8/10): SSO системи добре інтегруються з багатьма сучасними платформами завдяки підтримці стандартів, таких як OAuth і SAML. Це дозволяє використовувати єдиний вхід для різних хмарних та локальних сервісів. Проте не всі більш старі системи можуть легко підтримувати SSO, що може обмежити його інтеграцію в певних середовищах.
- MFA (9/10): добре інтегрується з різними IAM та SSO системами, що дозволяє підвищити загальну безпеку доступу. Важливою перевагою є те, що сучасні рішення MFA підтримують різноманітні фактори автентифікації (рис. 2.3), завдяки чому їх можна використовувати в будь-якому середовищі — як у хмарних, так і в локальних системах.

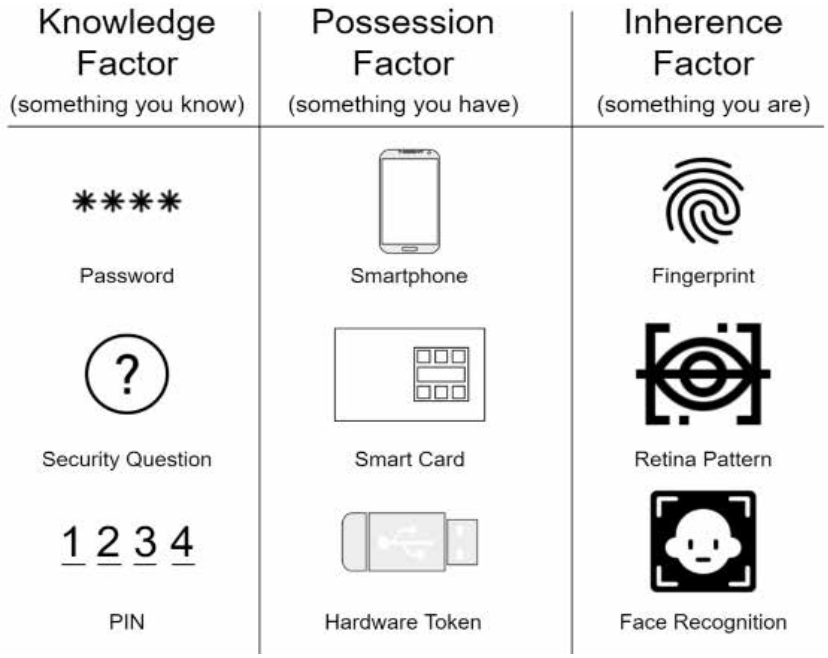

Рисунок 2.3 – Види факторів MFA

Стабільність:

- IAM (8/10): IAM системи зазвичай є стабільними рішеннями, особливо якщо їх налаштувати і підтримувати належним чином. Однак складні системи з великою кількістю користувачів та політик можуть потребувати додаткових ресурсів для забезпечення стабільної роботи. Також важливим аспектом є регулярне оновлення та перевірка на наявність вразливостей.
- SSO (7/10): системи SSO досить стабільні, але через централізований підхід можуть виникати серйозні проблеми, якщо SSO виходить з ладу. Це може призвести до недоступності всіх інтегрованих систем для користувачів, що є критичним недоліком. Для забезпечення стабільності важливо мати резервні механізми або додаткові шляхи доступу.
- MFA (8/10): сучасні MFA рішення є доволі стабільними та забезпечують високий рівень доступності. Проблеми можуть виникати лише в разі втрати доступу до факторів автентифікації (носій електронного ключа), але зазвичай організації мають процеси для відновлення доступу. Стабільність залежить також від надійності каналів передачі даних для кодів або інших додаткових факторів.

Нижче наведено підсумок оцінювання даних методів за відповідними критеріями:

Таблиця 2.1 – Оцінювання методів ідентифікації та автентифікації

Критерій	IAM	SSO	MFA
Ефективність захисту	8	7	9
Час обробки запиту	7	9	6
Адаптивність до ризиків	9	6	9
Рівень зручності	6	10	7
Здатність до інтеграції	9	8	9
Стабільність	8	7	8
Середня оцінка	7,83	7,83	8

На основі вище сказаного можна підбити відповідний підсумок:

- IAM: є потужним інструментом для управління доступом завдяки високій гнучкості у налаштуванні політик, що дозволяє ефективно керувати доступом у великих компаніях. Висока адаптивність до ризиків та здатність інтеграції з іншими системами роблять IAM корисним для середовищ з високими вимогами до безпеки. Однак складність налаштування і дещо обмежений рівень зручності можуть бути значними перешкодами в організаціях, де важлива простота використання.
- SSO: відзначається високою зручністю для кінцевих користувачів, надаючи їм можливість єдиного входу до різних систем. Проте, його безпека може бути уразливою при компрометації облікових даних, особливо без інтеграції з додатковими факторами автентифікації, такими як MFA. Це робить SSO ефективним рішенням для середовищ з низькими або середніми вимогами до безпеки, де пріоритетом є зручність і швидкість доступу.
- MFA: надає найвищий рівень захисту серед розглянутих методів, завдяки комбінації різних факторів, що значно ускладнює компрометацію облікових записів. Незважаючи на те, що

багатофакторна автентифікація може знижувати зручність та швидкість процесу входу, це є виправданою ціною за суттєве підвищення безпеки. Це робить MFA найкращим вибором для середовищ з високими вимогами до безпеки, де ризик компрометації є критичним.

Доцільність використання кожного з методів визначається потребами конкретного корпоративного середовища. IAM підходить для складних корпоративних середовищ з акцентом на безпеку та гнучкість, SSO — для середовищ, де ключовими параметрами є зручність і швидкість, а MFA — для критично важливих систем, де безпека є основним пріоритетом.

2.4 Дослідження методів управління доступом

Наступним етапом є порівняння таких методів, як RBAC, PAM, ABAC з точки зору їх гнучкості, адаптивності і сумісності стосовно політик та стандартів безпеки, специфіки адміністрування, автоматизованості, а також аспекту моніторингу.

Гнучкість:

- RBAC (6/10): є класичним методом контролю доступу, який надає права на основі визначених ролей. Однак система не є дуже гнучка, оскільки управління привілеями обмежується жорсткими ролями, що може стати проблемою в складних середовищах з багатьма атрибутами доступу. Для великих організацій з динамічними вимогами до доступу можуть виникати труднощі в оперативному налаштуванні різних ролей для кожного нового працівника або зміни привілеїв, наприклад, у разі зміни посади[14].
- PAM (7/10): надає більше гнучкості в управлінні доступом до привілейованих облікових записів. Цей метод дозволяє налаштувати політики доступу до важливих даних і систем на основі конкретних сценаріїв і вимог, забезпечуючи привілейований доступ лише тоді, коли

це необхідно (рис. 2.4). Хоча PAM гнучкіший у випадках контролю над ключовими ресурсами, його впровадження на рівні всієї організації може бути складнішим[15].

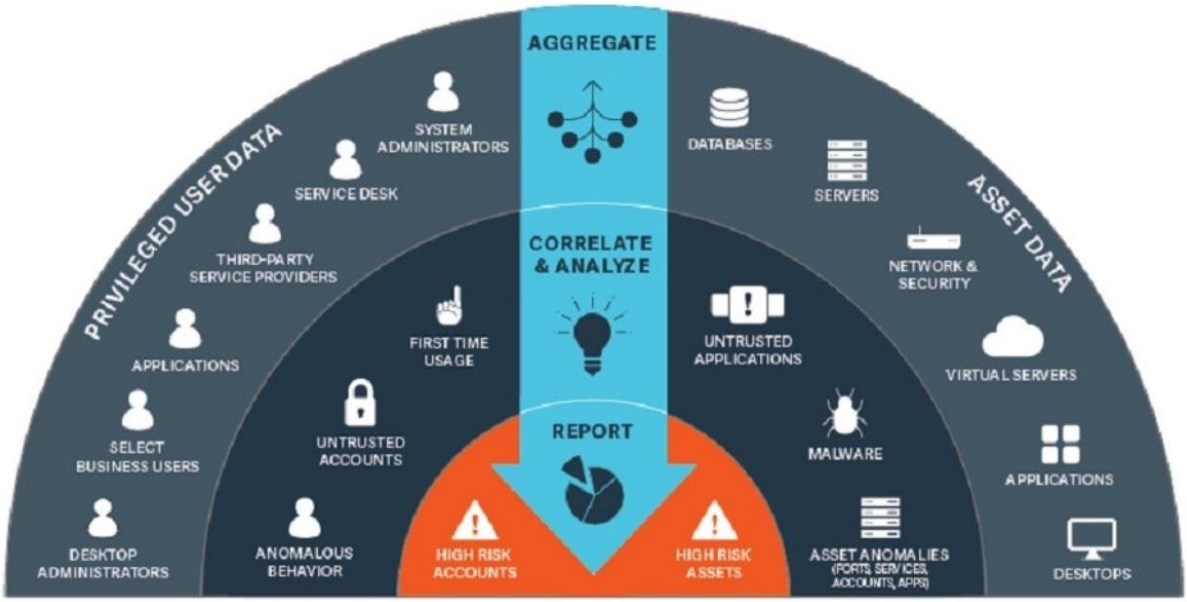

Рисунок 2.4 – Аспекти привілейованого доступу: розподілення даних, кореляція та аналіз, звітність

- ABAC (9/10): ABAC забезпечує високий рівень гнучкості, оскільки дозволяє надавати доступ на основі атрибутів користувача, середовища, об’єкта та дій. Цей підхід дозволяє визначати складні правила доступу, що враховують не тільки ролі, але й інші фактори, такі як час доби, місцезнаходження, тип запиту, що значно підвищує точність контролю доступу. Це дозволяє краще адаптуватися до мінливих умов роботи та потреб організації[16].

Сумісність з політиками безпеки:

- RBAC (8/10): добре підходить для організацій з чітко визначеними політиками безпеки, які можна адаптувати під ролі. Це дозволяє стандартизувати доступ до ресурсів залежно від посади або функції користувача. Однак у більш складних сценаріях, де потрібно враховувати різні динамічні фактори, RBAC може не відповідати усім вимогам безпеки без додаткових налаштувань чи інтеграції з іншими методами.

- PAM (8/10): пропонує високу сумісність з політиками, оскільки дозволяє централізовано керувати доступом до привілейованих облікових записів і може бути налаштований відповідно до специфічних вимог безпеки. За допомогою PAM можна створювати конкретні політики інформаційної безпеки, які реалізують надання доступу тільки для виконання конкретних завдань, що мінімізує ризик зловживань.
- ABAC (9/10): ABAC забезпечує найвищу сумісність з політиками безпеки завдяки можливості налаштовувати доступ на основі великої кількості атрибутів та динамічних умов. Це дозволяє точно налаштувати політики безпеки, відповідно до складних сценаріїв, включаючи обмеження доступу залежно від контексту та ризиків[17].

Автоматизація процесів:

- RBAC (7/10): може підтримувати базову автоматизацію шляхом автоматичного призначення ролей користувачам на основі їх позицій або функцій. Однак в організаціях з великою кількістю користувачів і часто змінюваними ролями потрібне ручне втручання для управління політиками доступу. Це обмежує рівень автоматизації, оскільки створення нових ролей або зміна існуючих потребує значних адміністративних зусиль.
- PAM (7/10): PAM підтримує певний рівень автоматизації для управління привілейованими правами доступу. Наприклад, можна налаштувати тимчасовий доступ до облікових записів або автоматичне видалення прав після виконання конкретних завдань. Проте автоматизація обмежена привілейованими обліковими записами, і PAM зазвичай вимагає регулярного ручного втручання для налаштування політик[18].
- ABAC (9/10): підтримує високий рівень автоматизації завдяки тому, що правила доступу налаштовуються динамічно на основі атрибутів і контексту (рис. 2.5). Системи, що використовують ABAC, можуть автоматично надавати або обмежувати доступ в залежності від

відповідних умов, що спрощує управління доступом і зменшує необхідність у ручному втручанні. Це дозволяє створювати масштабовані та гнучкі комп'ютерні системи доступу.

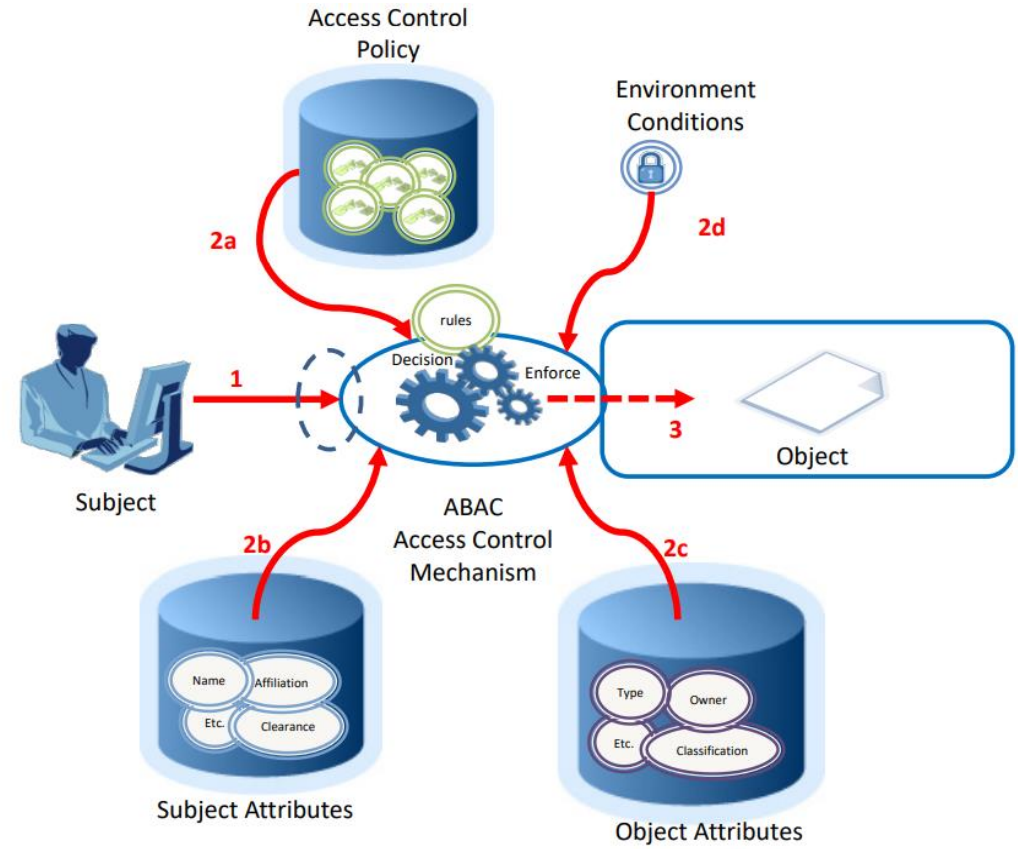

Рисунок 2.5 – Класичний сценарій роботи ABAC

Можливість моніторингу:

- RBAC (7/10): забезпечує базові функції моніторингу, оскільки кожен користувач має чітко визначену роль та набір прав доступу. Адміністратори можуть контролювати, які ролі надано користувачам, і відстежувати їх зміну. Однак у системах з великою кількістю ролей моніторинг може бути складним, оскільки зміни ролей та прав часто потребують ретельного аналізу.
- PAM (9/10): пропонує широкий набір можливостей для моніторингу, включаючи детальне відстеження активності привілейованих користувачів. Це включає журнали активності, аудит, а також можливість запису сесій для подальшого аналізу. PAM дозволяє

виявляти підозрілу активність, аналізувати дії користувачів та забезпечує контроль над найбільш критичними операціями.

- ABAC (8/10): має потужні можливості для моніторингу, оскільки кожен запит доступу аналізується на основі атрибутів і умов. Це дозволяє відслідковувати, хто і за яких умов отримав доступ до ресурсів, що значно підвищує прозорість системи. Однак для повноцінного моніторингу може знадобитися додаткова інфраструктура для запису та аналізу великої кількості контекстних даних.

Простота адміністрування:

- RBAC (7/10): Для малих та середніх організацій RBAC є відносно простим в адмініструванні, оскільки користувачам призначаються стандартні ролі, що відповідають їхнім функціям (рис. 2.6). Однак для великих організацій з багатьма різними ролями та привілеями адміністративне управління може стати складним і трудомістким, особливо якщо часто змінюються вимоги доступу до інформаційних ресурсів[19].

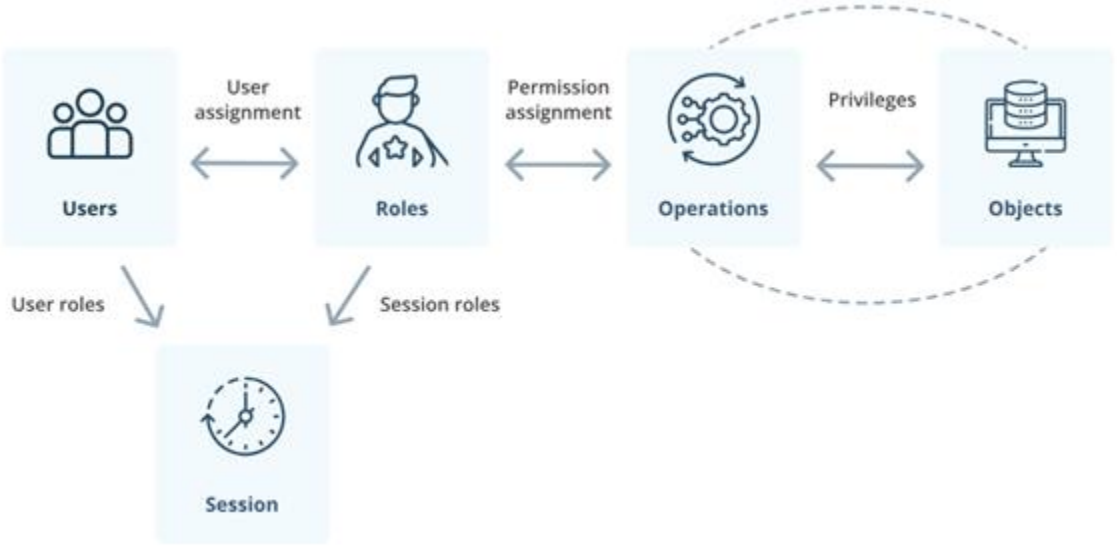

Рисунок 2.6 – Принцип адміністрування RBAC

- PAM (7/10): вимагає додаткових адміністративних зусиль для налаштування та контролю привілейованого доступу. Хоча PAM автоматизує певні аспекти управління привілеями, адміністрування привілейованих облікових записів, контроль за сесіями та налаштування політик можуть бути складними для великих середовищ.

Однак простота адміністрування значно покращується завдяки наявності автоматичних функцій моніторингу та контролю.

- ABAC (5/10): адміністрування ABAC є найскладнішим, оскільки система передбачає налаштування великої кількості атрибутів, політик та умов надання відповідних прав доступу. Створення та підтримка цих політик вимагає значних адміністративних зусиль, особливо в організаціях з високо масштабованими комп'ютерними системами. Хоча ABAC забезпечує гнучкість і точність, його адміністрування є трудомістким і вимагає високого рівня експертизи.

Відповідність стандартам безпеки:

- RBAC (9/10): є добре відомим і усталеним підходом, що відповідає більшості стандартів безпеки, таких як ISO 27001 і NIST SP 800-53. Цей метод забезпечує надійну відповідність стандартам, оскільки дозволяє чітко визначати права доступу і забезпечувати мінімізацію привілеїв. Однак в умовах, коли необхідна більша гнучкість, RBAC може бути обмеженим[20].
- PAM (9/10): повністю відповідає сучасним стандартам безпеки, таким як GDPR та інші. Це робить PAM важливим інструментом для забезпечення захисту привілейованих облікових записів, які часто є ціллю атак. Дотримання принципу мінімальних привілеїв та надання детальних звітів про діяльність відповідних працівників, що допомагає відповідати вимогам аудиту.
- ABAC (9/10): ABAC забезпечує відповідність стандартам безпеки на високому рівні, оскільки дозволяє точно налаштовувати політики доступу, які враховують атрибути та контекст. Це надає можливість забезпечити відповідність стандартам, що вимагають динамічного контролю доступу та моніторингу умов використання. ABAC є особливо корисним для організацій, які мають складні вимоги до безпеки та конфіденційності.

Нижче наведено підсумок оцінювання даних методів за відповідними критеріями:

Таблиця 2.2 – Оцінювання методів управління доступом

Критерій	RBAC	PAM	ABAC
Гнучкість	6	7	9
Сумісність з політиками безпеки	8	8	9
Автоматизація процесів	7	7	9
Можливість моніторингу	7	9	8
Простота адміністрування	7	7	5
Відповідність стандартам безпеки	9	9	9
Середня оцінка	7,33	7,83	8,16

На основі вище сказаного можна підбити відповідний підсумок:

- RBAC: непогана ефективність як класичного методу контролю доступу, що добре працює в організаціях із чітко визначеними ролями та політиками. Однак його обмежена гнучкість та адаптивність можуть стати викликом для складних та динамічних середовищ, які потребують постійного оновлення прав доступу. Він добре підходить для середніх та малих компаній, де не потрібен динамічний контроль доступу.
- PAM: забезпечує надійний захист критично важливих ресурсів через централізоване управління доступом, гнучкість та сумісність з вимогами сучасних стандартів безпеки. Його застосування доцільне в організаціях, де потрібен підвищений контроль доступу до ключових систем та привілейованих облікових записів. Недоліком є те, що PAM вимагає значних адміністративних зусиль для налаштування, особливо в масштабованих середовищах.
- ABAC: найоптимальніший варіант із розглянутих завдяки своїй високій гнучкості та динамічному підходу до контролю доступу. Він забезпечує найкращу адаптивність та сумісність з політиками безпеки, що особливо важливо для організацій із складними та мінливими вимогами. ABAC підходить для середовищ, де необхідно враховувати різні атрибути доступу, такі як час, місцезнаходження, тип запиту, що дозволяє детальніше налаштовувати доступ. Однак високий рівень

складності адміністрування може негативно вплинути на його впровадження в деяких організаціях.

Загалом, найбільш доцільним є вибір методу управління доступом залежно від розміру та потреб організації з точки зору політик безпеки. Для компаній із чіткими політиками та меншою потребою в адаптивності підійде RBAC. Для організацій із вимогами до привілейованого доступу і моніторингу активності PAM стане ефективним рішенням. У складних середовищах, що потребують гнучкості та динамічного доступу, найкращим вибором є ABAC.

2.5 Дослідження методів аналізу і управління безпекою

Даний етап являє собою аналіз методів UEBA, SIEM та DLP з точки зору точності виявлення загроз, специфіки їх реагування, масштабованості, сумісності, звітності та характеру аналізу даних у реальному часі.

Точність виявлення загроз:

- UEBA (9/10): використовує аналітику поведінки користувачів і сутностей, що забезпечує високу точність у виявленні аномалій. Завдяки використанню машинного навчання та алгоритмів штучного інтелекту, UEBA здатен виявляти потенційні загрози, базуючись на відхиленнях від нормальної поведінки. Це дозволяє виявляти невідомі загрози, внутрішні загрози та складні атаки, які можуть не фіксуватися традиційними методами[21].
- SIEM (8/10): надає високу точність виявлення загроз завдяки поєднанню збору даних з багатьох джерел, кореляції подій і виявлення аномалій. Однак точність залежить від того, наскільки добре налаштовані правила і сигнатури, бо велика кількість хибних спрацьовувань може знижувати ефективність. Ручне налаштування правил також може стати проблемою при роботі з великою кількістю даних[22].

– DLP (7/10): забезпечує точність у виявленні витоків даних шляхом моніторингу використання, передачі та зберігання конфіденційної інформації. Однак його точність обмежена типами даних, які він може ідентифікувати. DLP не завжди ефективний у виявленні загроз, які не пов'язані з конфіденційною інформацією, тому його сфера дії обмежена загрозами, пов'язаними з втратою даних (рис. 2.7)[23].

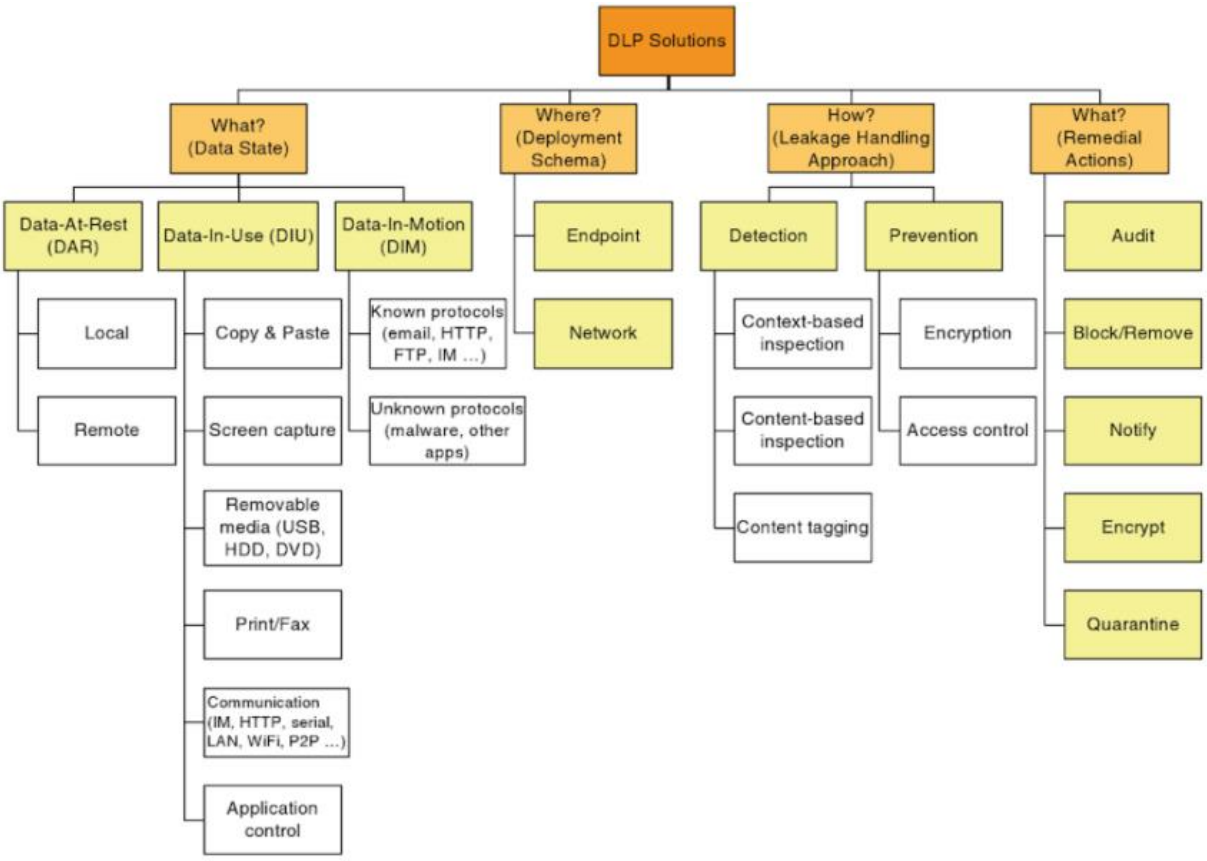

Рисунок 2.7 – Систематика DLP-рішень

Швидкість реагування:

– UEBA (7/10): потребує часу для збору даних і навчання моделей для виявлення аномалій. Це означає, що його швидкість реагування може бути нижчою порівняно з іншими технологіями, які працюють за жорстко встановленими правилами. Однак після впровадження та навчання UEBA здатен оперативно реагувати на аномальні дії користувачів і сутностей.

– SIEM (8/10): зазвичай забезпечує гарну швидкість реагування завдяки можливості виявляти події в режимі реального часу. Може миттєво сповіщати про підозрілі активності на основі налаштованих правил і кореляційних аналізів, що дає змогу швидко реагувати на інциденти

безпеки. Затримки можуть виникати, наприклад, якщо система перевантажена обробкою великої кількості даних[24].

- DLP (8/10): DLP системи забезпечують швидку реакцію на витоки даних, особливо при налаштуванні політик для автоматичного блокування дій, що порушують безпеку даних. Однак ефективність і швидкість можуть залежати від типу загрози та конфігурації системи.

Масштабованість:

- UEBA (8/10): є масштабованим рішенням, оскільки може аналізувати поведінку великої кількості користувачів та пристроїв у різних середовищах (рис. 2.8). Масштабованість залежить від можливостей обробки даних та складності алгоритмів. Збільшення кількості користувачів або даних не суттєво впливає на загальну продуктивність, проте може вимагати більше ресурсів для зберігання та аналізу даних[25].

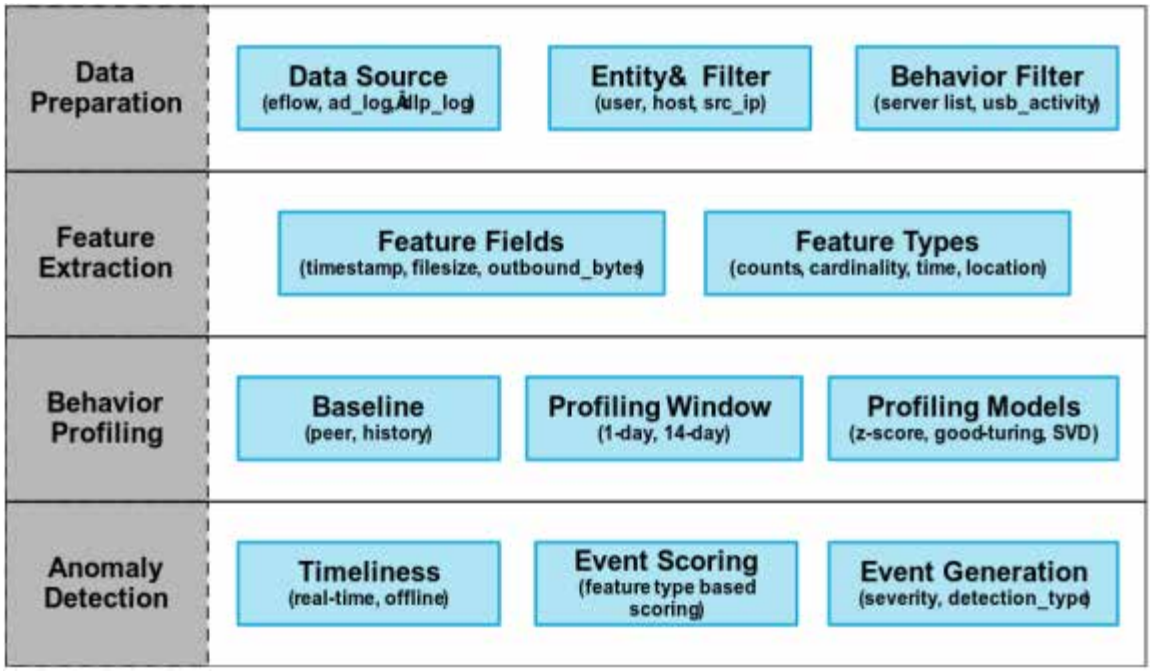

Рисунок 2.8 – Загальна архітектура UEBA

- SIEM (9/10): властивий високий рівень масштабованості, здатен інтегрувати велику кількість джерел даних і обробляти події з різних систем в організації. Системи SIEM, як правило, підтримують масштабування через додавання серверів або розподілення завдань між

ними, що дозволяє ефективно працювати в великих корпоративних комп'ютерних мережах і системах.

- DLP (7/10): масштабованість обмежена здатністю моніторити дані в різних середовищах і пристроях. DLP може бути складно масштабувати на рівні всієї організації, якщо він має різні типи середовищ або використовує велику кількість різноманітних даних. Це може створити проблеми в розгортанні DLP у великих організаціях зі значною номенклатурою інформаційних активів підприємства.

Сумісність з існуючими системами:

- UEBA (7/10): сумісний з багатьма іншими рішеннями безпеки, зокрема SIEM та DLP, але його інтеграція може вимагати складної конфігурації. Він здатен використовувати дані з існуючих систем для аналізу поведінки, однак для повноцінної роботи може знадобитися додатковий збір даних і їхня обробка.
- SIEM (9/10): відомий своєю здатністю інтегруватися з широким спектром систем, таких як мережеві пристрої, антивірусні програми, системи виявлення вторгнень (IDS), міжмережеві екрани тощо. Це робить його ідеальним рішенням для центрального управління безпекою в великих організаціях. Інтеграція з іншими системами безпеки зазвичай відбувається без значних проблем[26].
- DLP (8/10): добре інтегрується з іншими системами, особливо в контексті моніторингу даних і контролю їх пересування між мережами або пристроями. Він може працювати разом із SIEM або іншими системами для забезпечення захисту даних. Проте, інтеграція з великими корпоративними системами може вимагати додаткових налаштувань[27].

Аналіз даних у реальному часі:

- UEBA (7/10): може проводити аналіз даних у реальному часі, але через те, що система працює з великими обсягами інформації і використовує машинне навчання, це може бути менш оперативним порівняно з іншими методами. Часто для точного виявлення аномалій необхідний

період навчання, що може уповільнити процес виявлення загроз в реальному часі[28].

– SIEM (9/10): ці системи орієнтовані на аналіз даних у режимі реального часу, що дозволяє оперативно виявляти інциденти безпеки. SIEM збирає дані з різних джерел та обробляє їх в даному режимі для виявлення загроз і порушень. Це забезпечує високий рівень контролю за безпекою в організації (рис. 2.9).

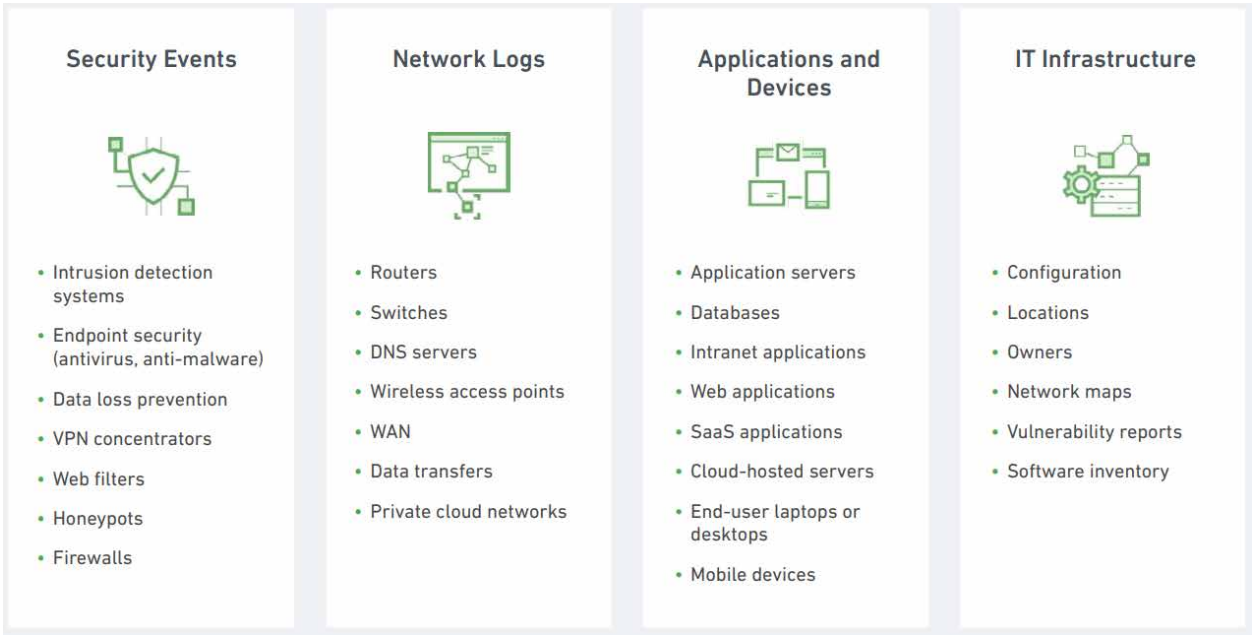

Рисунок 2.9 – Джерела збору даних SIEM

– DLP (8/10): забезпечує аналіз даних у реальному часі, зокрема моніторинг і захист конфіденційної інформації, що дозволяє виявляти та блокувати потенційні витоки даних миттєво. Однак обмеження можуть виникати при великій кількості різноманітних даних, що уповільнює швидкість реагування.

Доступність звітності:

– UEBA (8/10): надає детальні звіти про поведінку користувачів і аномалії, однак формування цих звітів може потребувати часу. Звіти зазвичай містять аналітику поведінки, опис виявлених загроз і рекомендації щодо реагування. Оскільки UEBA використовує машинне навчання, звіти можуть бути складними для розуміння без спеціальної підготовки.

- SIEM (9/10): забезпечує широкий спектр звітів у реальному часі, що дає змогу відстежувати активність мережі та події безпеки, пропонує готові шаблони звітів і можливість налаштування, що робить його ефективним інструментом для моніторингу і аудиту безпеки. Звіти SIEM зазвичай легкі для розуміння та забезпечують повний огляд подій в організації.
- DLP (7/10): надає звіти про потенційні витоки даних і спроби порушення політик безпеки, проте вони можуть бути обмеженими лише інформацією про витік даних, а не про інші загрози. Це робить його менш універсальним у порівнянні з SIEM, але достатнім для виконання своєї основної функції.

Нижче наведено підсумок оцінювання даних методів за відповідними критеріями:

Таблиця 2.3 – Оцінювання методів аналізу та управління безпекою

Критерій	UEBA	SIEM	DLP
Точність виявлення загроз	9	8	7
Швидкість реагування	7	8	8
Масштабованість	8	9	7
Сумісність з існуючими системами	7	9	8
Аналіз даних у реальному часі	7	9	8
Доступність звітності	8	9	7
Середня оцінка	7,67	8,67	7,5

На основі вище сказаного можна підбити відповідний підсумок:

- UEBA: завдяки аналізу поведінкових даних і використанню штучного інтелекту досягає високої точності у виявленні складних та внутрішніх загроз. Це рішення добре підходить для організацій, де потрібен детальний аналіз поведінки користувачів та виявлення аномалій, що не можна ідентифікувати за допомогою традиційних методів. Проте недоліком є менша швидкість реагування через необхідність навчання моделей та обробки великих обсягів даних, а також складна інтеграція з іншими системами безпеки.

- SIEM: є найефективнішим рішенням серед розглянутих, особливо для великих організацій з високими вимогами до управління безпекою та контролю. Висока точність виявлення загроз та здатність інтеграції з різноманітними системами роблять його універсальним інструментом для централізованого управління безпекою. Оперативний аналіз подій у реальному часі дозволяє швидко реагувати на загрози, а можливість створення звітів забезпечує ефективний моніторинг безпеки. Недоліком може бути потреба в налаштуванні правил, що підвищує складність використання та може призвести до появи хибних спрацьовувань.
- DLP: ефективний для захисту конфіденційних даних і запобігання витокам інформації, забезпечує достатню швидкість реагування на інциденти, пов'язані з витоками, та добре інтегрується з іншими системами безпеки. Однак DLP не є універсальним рішенням, оскільки його функціональність обмежена лише питаннями конфіденційності та витоків даних, без охоплення інших загроз.

У підсумку, SIEM є найдоцільнішим та найбільш універсальним вибором для організацій, де потрібен комплексний підхід до управління безпекою. UEBA рекомендується для випадків, коли акцент ставиться на виявлення аномалій та внутрішніх загроз. DLP ефективний у сферах, де основна увага приділяється захисту конфіденційної інформації. Оптимальним підходом є комбіноване використання різноманітних за функціоналом методів, що дозволить забезпечити всебічний захист даних та інфраструктури організації.

2.6 Дослідження моделей безпеки

Останнім етапом дослідження є аналіз моделей безпеки ZTSM, CASB, VM з точки зору принципів мінімального доступу, гнучкості, аудиту, аналізу вразливостей та аномалій, а також відповідності регуляторним вимогам.

Принципи мінімального доступу:

– ZTSM (10/10): Zero Trust базується на принципі "не довіряй, перевіряй", де кожен доступ потребує верифікації незалежно від місця знаходження чи рівня привілеїв користувача. Це підходить ідеально для впровадження принципів мінімального доступу, оскільки забезпечує контроль за кожною дією користувача (рис. 2.10). Кожен запит на доступ перевіряється окремо, що робить цю модель максимально відповідною до мінімальних прав[29].

Рисунок 2.10 – Метрики Zero Trust

– CASB (8/10): контролює доступ до хмарних ресурсів, забезпечуючи мінімальний доступ через політики безпеки. Однак цей принцип часто реалізується в контексті лише хмарних сервісів, що може обмежити його застосування до внутрішніх ресурсів. Політики CASB можуть ефективно обмежувати доступ до конкретних файлів чи функцій, але не завжди інтегруються з іншими системами управління доступом[30].
– VM (7/10): фокусується на виявленні та усуненні вразливостей, але прямий контроль над доступом реалізується в меншій мірі. Хоча

управління вразливостями може допомагати визначати ризики і адаптувати рівень доступу, ця система не призначена для постійного контролю доступу, як ZTSM чи CASB[31].

Аудит та контроль:

- ZTSM (9/10): надає високий рівень аудиту та контролю, оскільки кожен доступ до ресурсів постійно перевіряється і фіксуються всі дії. Системи на основі Zero Trust дозволяють контролювати та відслідковувати активність користувачів і пристроїв у режимі реального часу, що є важливою умовою для аудиту.
- CASB (8/10): забезпечує аудит дій користувачів у хмарних середовищах, відстежує переміщення даних і контроль над доступом. Висока ефективність у моніторингу хмарних додатків і даних, але обмежені можливості щодо внутрішніх корпоративних систем або поза хмарним середовищем. Інтеграція з SIEM може розширити ці можливості[32].
- VM (7/10): системи управління вразливостями зазвичай включають можливості аудиту, фокусуючись на ідентифікації та управлінні ризиками. Однак вони не настільки ефективні для постійного контролю за діями користувачів і пристроїв, як інші методи. Контроль відбувається на рівні аналізу вразливостей, а не користувачів чи їх дій.

Аналіз вразливостей:

- ZTSM (7/10): Хоча ZTSM забезпечує контроль за доступом і моніторинг, він менш спрямований на безпосередній аналіз вразливостей. Основний акцент робиться на перевірці користувачів і пристроїв під час кожної спроби доступу, що знижує ймовірність експлуатації вразливостей, але прямий аналіз потребує інтеграції з іншими системами[33].
- CASB (8/10): може виконувати аналіз вразливостей на рівні хмарних додатків і користувачів, фокусуючись на захисті даних та обмеженні доступу. Однак цей аналіз обмежується хмарним середовищем, що

може залишати поза увагою вразливості в корпоративних системах або в пристроях кінцевих користувачів.

- VM (10/10): системи управління вразливостями спеціалізуються на аналізі прогалин в програмному забезпеченні, мережах, та кінцевих пристроях. Вони забезпечують детальний аналіз і оцінку ризиків, ідентифікуючи критичні вразливості і дозволяючи їх усунення. Це робить їх незамінними для підтримки безпеки в корпоративних мережах (рис. 2.11).

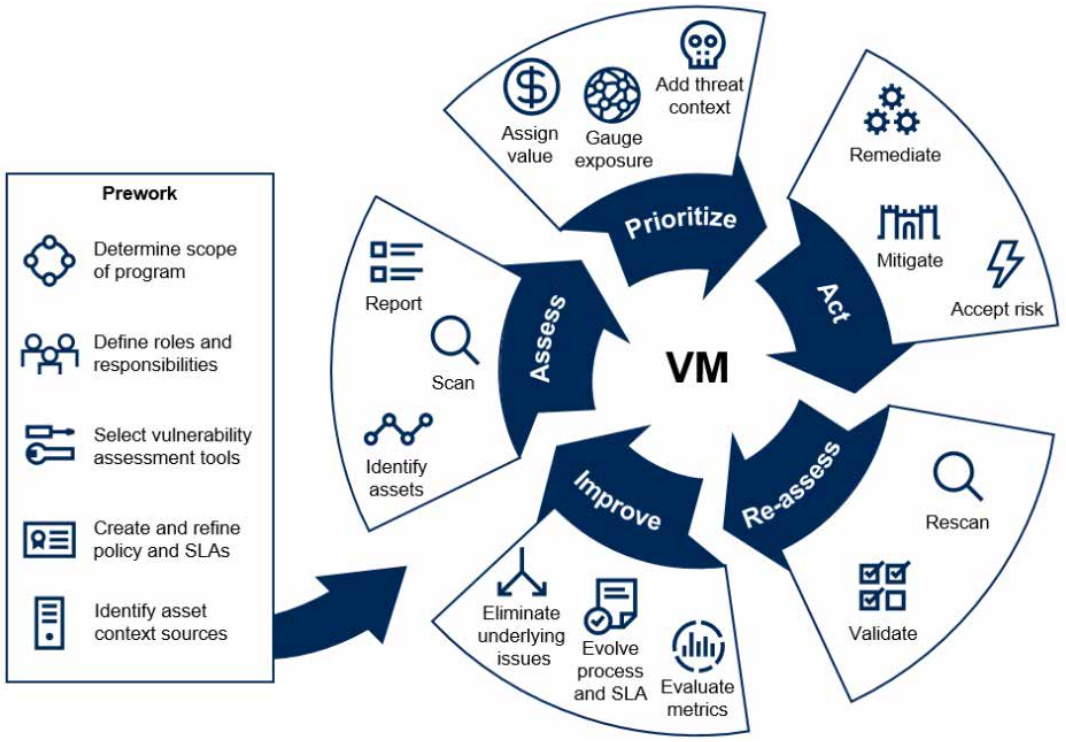

Рисунок 2.11 – Цикл VM

Гнучкість архітектури:

- ZTSM (8/10): є доволі гнучким, оскільки його принципи можуть бути застосовані до будь-якої архітектури – як до хмарних, так і до локальних середовищ. Він інтегрується з існуючими системами безпеки, дозволяючи адаптацію до нових загроз та технологій. Проте впровадження Zero Trust може вимагати значних ресурсів і часу[34].
- CASB (8/10): CASB має певну гнучкість, але переважно у контексті хмарних сервісів. Впровадження CASB вимагає інтеграції з хмарними додатками, що може бути складним для організацій із гібридними або виключно локальними середовищами.

- VM (7/10): архітектура систем управління вразливостями зазвичай менш гнучка порівняно з іншими рішеннями, оскільки вона фокусується на виявленні та усуненні вразливостей. Однак ці системи можуть бути інтегровані з іншими інструментами кібербезпеки для підвищення ефективності управління ризиками[35].

Можливість виявлення аномалій:

- ZTSM (9/10): постійно контролює і перевіряє кожен запит на доступ, що дозволяє швидко виявляти аномальні дії. Використовуючи поведінкові дані та аналіз дій користувачів, Zero Trust здатен ідентифікувати потенційні загрози у режимі реального часу.
- CASB (9/10): може виявляти аномалії у хмарних додатках і сервісах, забезпечуючи захист від небажаного доступу або несанкціонованих дій. Він також може інтегруватися з іншими системами безпеки для розширення можливостей моніторингу аномалій.

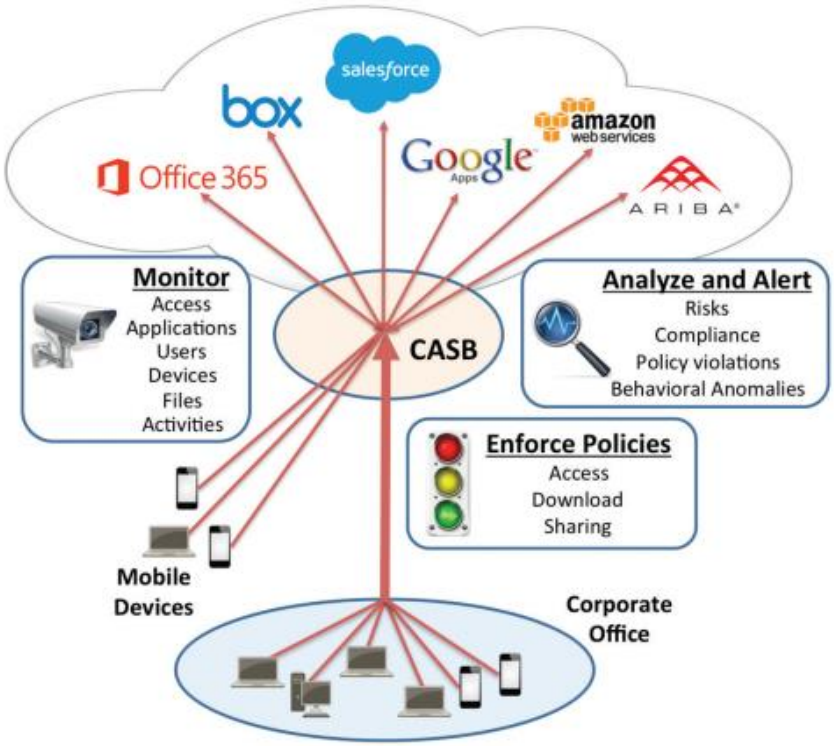

Рисунок 2.12 – Специфіка роботи CASB

- VM (6/10): виявлення аномалій не є основною метою VM. Ці системи спрямовані на ідентифікацію відомих вразливостей, а не поведінкових аномалій або підозрілої активності в системах.

Відповідність регуляторним вимогам:

- ZTSM (9/10): допомагає організаціям відповідати багатьом сучасним стандартам безпеки, таким як GDPR та HIPAA. Постійний моніторинг і контроль доступу допомагають дотримуватися вимог щодо конфіденційності та захисту даних.
- CASB (9/10): забезпечує відповідність вимогам регуляторів, особливо в контексті хмарних сервісів і захисту даних. Він дозволяє організаціям дотримуватися вимог CCPA та інших стандартів, забезпечуючи контроль за доступом до хмарних даних і їхнім переміщенням[36].
- VM (8/10): Системи управління вразливостями допомагають організаціям відповідати вимогам щодо безпеки через виявлення та усунення вразливостей. Проте вони менш універсальні у виконанні комплексних регуляторних вимог, оскільки зосереджені на специфічних аспектах управління ризиками.

Нижче наведено підсумок оцінювання даних методів за відповідними критеріями:

Таблиця 2.4 – Оцінювання методів аналізу та управління безпекою

Критерій	ZTSM	CASB	VM
Принципи мінімального доступу	10	8	7
Аудит та контроль	9	8	7
Аналіз вразливостей	7	8	10
Гнучкість архітектури	8	8	7
Можливість виявлення аномалій	9	9	6
Відповідність регуляторним вимогам	9	9	8
Середня оцінка	8,67	8,33	7,5

На основі вище сказаного можна підбити відповідний підсумок:

- ZTSM: показує найвищі результати в аспектах контролю доступу та аудиту, забезпечуючи максимальний рівень безпеки завдяки принципу "не довіряй, перевіряй", де кожен запит на доступ вимагає верифікації незалежно від привілеїв користувача. Це дозволяє знизити ризик несанкціонованого доступу та підвищити рівень контролю за діями

користувачів. Хоча аналіз вразливостей є менш пріоритетним для ZTSM, система забезпечує достатню гнучкість для інтеграції з існуючими системами безпеки. Таким чином, ZTSM є найбільш ефективним підходом для організацій, де критичним є постійний контроль та управління доступом.

- CASB: показує високу ефективність у контролі за доступом до хмарних ресурсів і виявленні аномалій в рамках хмарного середовища. Незважаючи на обмеження для локальних систем, CASB може успішно інтегруватися з іншими інструментами кібербезпеки, що підвищує його гнучкість у складних хмарних архітектурах. У контексті дотримання регуляторних вимог CASB є надійним рішенням, оскільки здатний контролювати переміщення даних і обмежувати доступ відповідно до сучасних стандартів.
- VM: досягає найвищих показників за критерієм аналізу вразливостей, але має деякі обмеження у відслідковуванні аномалій та постійному моніторингу доступу. Незважаючи на це, є ключовою складовою для управління ризиками та підтримки мережевої безпеки шляхом ідентифікації критичних вразливостей, особливо в середовищах, де існує необхідність швидкого реагування на нові загрози.

Даний підсумок свідчить про те, що кожна модель має свої переваги та може бути корисною в різних сценаріях. ZTSM є оптимальним для організацій, що потребують високого рівня контролю та адаптивності до аномалій, CASB підходить для захисту при використанні хмарних середовищ і відповідності регуляторним вимогам, а VM спеціалізується на підтримці безпеки інформаційної системи з точки зору управління вразливостями.

Загалом, дане дослідження розглянутих підгруп за зазначеними критеріями допомогло визначитися з доцільністю використання низки сучасних методів управління користувачами комп’ютерної системи в залежності від відповідного контексту, вимог, специфіки і бачення організації, а також із тим, які методи є найефективнішими й найбільш універсальними для втілення різних аспектів цього процесу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація MFA

Заключним етапом даної роботи є програмна реалізація мовою Python у хмарному середовищі Google Colab найефективніших за оцінками методами управління користувачами всіх визначених підгруп, а саме: MFA, ABAC, SIEM, ZTSM. У першу чергу було розроблено консольну систему MFA на основі пароля та одноразового пароля (OTP) для захисту доступу користувачів. Користувач створює обліковий запис, вводячи унікальне ім'я та пароль. При кожній спробі входу генерується новий OTP, який користувач має ввести для підтвердження особи. Якщо пароль і OTP вірні, автентифікація успішна; інакше доступ заблоковано після трьох невдалих спроб.

Лістинг 3.1 – MFA

```
!pip install pyotp
import pyotp  # Імпорт pyotp для генерації та перевірки одноразових паролів
import getpass  # Імпорт getpass для безпечного введення паролів без їх відображення на екрані

# Клас, що представляє Користувача в системі
class User:
    def __init__(self, username, password):
        self.username = username  # Ім'я користувача
        self.password = password  # Пароль користувача
        self.secret_key = pyotp.random_base32()  # Секретний ключ для генерації OTP

    # Метод для генерації одноразового пароля (OTP)
    def generate_otp(self):
        totp = pyotp.TOTP(self.secret_key)
        return totp.now()  # Генеруємо поточний OTP

    # Метод для перевірки введеного пароля
    def verify_password(self, password):
        return self.password == password

    # Метод для перевірки введеного коду OTP
    def verify_otp(self, otp_code):
        totp = pyotp.TOTP(self.secret_key)
```

```
    # Клас для керування консоллю додатка MFA
    class MFAConsoleApp:
        def __init__(self):
            self.users = {}

        # Метод для реєстрації нового користувача
        def register_user(self):
            username = input("Введіть ім'я користувача для
реєстрації: ")
            if username in self.users:
                print("Користувач з таким іменем вже існує.")
                return
            password = getpass.getpass("Введіть пароль: ") #
Безпечне введення пароля
            confirm_password = getpass.getpass("Підтвердіть
пароль: ")
            if password != confirm_password:  # Перевіряємо, чи
збігаються паролі
                print("Паролі не співпадають. Повторіть
реєстрацію.")
                return
            new_user = User(username, password)  # Створюємо
новий екземпляр користувача
            self.users[username] = new_user
            print(f"Користувач {username} успішно
зареєстрований.")

        # Метод для входу зареєстрованого користувача
        def login_user(self):
            username = input("Введіть ім'я користувача: ")
            if username not in self.users:
                print("Користувача не знайдено.")
                return
            user = self.users[username]
            # Дозволено до трьох спроб для входу
            for attempt in range(3):
                # Перевірка пароля користувача
                password = getpass.getpass("Введіть пароль: ")
                if not user.verify_password(password):# Перевірка
правильності пароля
                    if attempt == 2:
                        print("Вичерпано кількість спроб. Доступ
заблоковано.")
                        return
                    print(f"Невірний пароль. Залишилося спроб -
{2-attempt}.")
                    continue
                # Генерація OTP для поточної спроби входу
                otp = user.generate_otp()
                print(f"Ваш одноразовий пароль (OTP): {otp}")
                # Запит OTP у користувача та його перевірка
                for OTP_attempt in range(3):
                    otp_input = input("Введіть одноразовий пароль
(OTP): ")
                    if otp == otp_input:  # Перевірка OTP
```

```
                        print("Автентифікація успішна!")
                        return
                    else:
                        if OTP_attempt == 2:
                            print("Вичерпано кількість спроб.
Доступ заблоковано.")
                            return
                        print(f"Невірний OTP. Залишилося спроб -
{2-OTP_attempt}.")
                        continue

        # Основний цикл для консолі додатка MFA
        def run(self):
            while True:
                print("\nОберіть дію:")
                print("1. Реєстрація нового користувача")
                print("2. Вхід (із підтвердженням паролю і OTP)")
                print("3. Вихід")
                choice = input("Введіть номер дії: ")
                if choice == '1':
                    self.register_user()
                elif choice == '2':
                    self.login_user()
                elif choice == '3':
                    print("Завершення роботи системи.")
                    break
                else:
                    print("Невірно. Спробуйте знову.")

    # Створюємо екземпляр MFAConsoleApp та запускаємо його
    app = MFAConsoleApp()
    app.run()
```

Нижче на рисунку 3.1 наведено результати роботи даної програми:

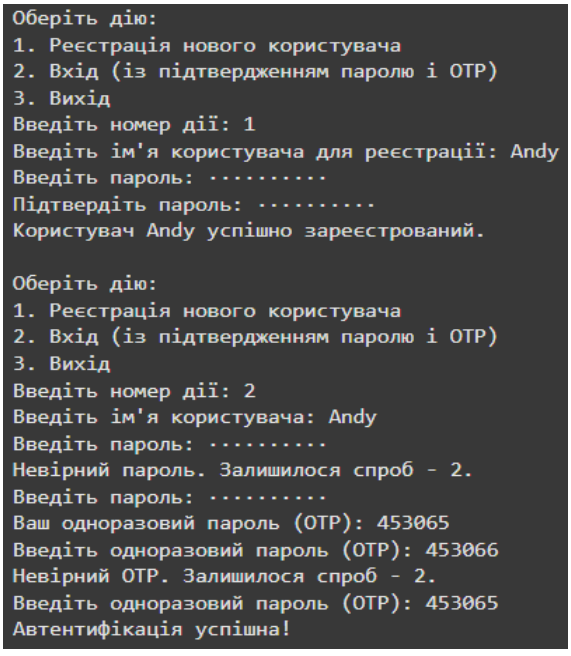

Рисунок 3.1 – Програмна реалізація MFA

3.2 Програмна реалізація ABAC

Дана розробка реалізує систему керування доступом на основі атрибутів, таких як департамент, рівень доступу та посада. Доступ надається, якщо департамент користувача і ресурсу збігаються. Користувач повинен мати рівень доступу не нижчий, ніж вказано для ресурсу. Якщо рівень доступу користувача і ресурсу однаковий, доступ дозволяється лише за збігу атрибуту посади.

Лістинг 3.2 – ABAC

```
    # Клас, що представляє Користувача
    class User:
        def __init__(self, username, attributes):
            self.username = username  # Ім'я користувача
            self.attributes = attributes  # Атрибути користувача,
такі як департамент, рівень доступу і позиція

    # Клас, що представляє Ресурс
    class Resource:
        def __init__(self, resource_name, attributes):
            self.resource_name = resource_name  # Назва ресурсу
            self.attributes = attributes  # Атрибути ресурсу

    # Клас, що містить політики доступу ABAC
    class ABACPolicy:
        def __init__(self):
            pass

        # Метод для оцінки політики доступу
        def evaluate(self, user, resource):
            # Перевірка відповідності департаментів користувача і
ресурсу
            if        user.attributes.get('department')        !=
resource.attributes.get('department'):
                return False
            # Визначаємо допустимі рівні доступу
            clearance_levels = ["low", "medium", "high"]
            user_clearance = user.attributes.get('clearance')
            resource_clearance                                    =
resource.attributes.get('clearance')
            if   (user_clearance   not   in   clearance_levels)   or
(resource_clearance not in clearance_levels):
                return False
            # Визначаємо індекс рівня доступу для порівняння
            user_clearance_level                                  =
clearance_levels.index(user_clearance)
```

```
            resource_clearance_level                                    =
clearance_levels.index(resource_clearance)
            if user_clearance_level < resource_clearance_level:
                return False
            # Якщо рівні доступу однакові, перевіряємо позиції
            if user_clearance_level == resource_clearance_level:
                if       user.attributes.get('position')       !=
resource.attributes.get('position'):
                    return False
            return True

    # Клас для керування політиками доступу ABAC
    class ABACSystem:
        def __init__(self):
            self.policies = []

        # Метод для перевірки доступу користувача до ресурсу
        def check_access(self, user, resource):
            policy = ABACPolicy()
            return policy.evaluate(user, resource)

    # Клас консолі для взаємодії з користувачами та ресурсами
    class ABACConsoleApp:
        def __init__(self):
            self.abac_system = ABACSystem()
            self.users = {}
            self.resources = {}

        # Реєстрація нового користувача з атрибутами
        def register_user(self, username, attributes):
            self.users[username] = User(username, attributes)

        # Додавання нового ресурсу з атрибутами
        def add_resource(self, resource_name, attributes):
            self.resources[resource_name]                               =
Resource(resource_name, attributes)

        # Метод для виведення списку зареєстрованих користувачів
        def list_users(self):
            print("\nСписок користувачів:")
            for username, user in self.users.items():
                print(f"Ім'я:        {username},        Атрибути:
{user.attributes}")

        # Метод для виведення списку доступних ресурсів
        def list_resources(self):
            print("\nСписок ресурсів:")
            for resource_name, resource in self.resources.items():
                print(f"Назва:     {resource_name},     Атрибути:
{resource.attributes}")

        # Метод для перевірки доступу користувача до ресурсу
        def check_access(self):
            username = input("Введіть ім'я користувача: ")
            resource_name = input("Введіть назву ресурсу: ")
```

```
            if username not in self.users:
                print("Користувача не знайдено.")
                return
            if resource_name not in self.resources:
                print("Ресурс не знайдено.")
                return

            user = self.users[username]
            resource = self.resources[resource_name]
            # Перевірка доступу з використанням системи ABAC
            if self.abac_system.check_access(user, resource):
                print(f"Доступ до ресурсу
'{resource.resource_name}' надано користувачу '{user.username}'.")
            else:
                print(f"У доступі до ресурсу
'{resource.resource_name}' відмовлено для користувача
'{user.username}'.")

        # Основний цикл консолі ABAC
        def run(self):
            print("Вітаємо в системі ABAC!")
            self.list_users()
            self.list_resources()

            while True:
                print("\nОберіть дію:")
                print("1. Зареєструвати нового користувача")
                print("2. Додати новий ресурс")
                print("3. Перевірити доступ до ресурсу")
                print("4. Вихід")

                choice = input("Оберіть номер дії: ")
                if choice == '1':
                    username = input("Введіть ім'я користувача: ")
                    attributes_input = input("Введіть атрибути
(наприклад, department=HR, clearance=high, position=manager): ")
                    attributes = dict(attr.split('=') for attr in
attributes_input.split(', '))
                    self.register_user(username, attributes)
                elif choice == '2':
                    resource_name = input("Введіть назву ресурсу:
")
                    attributes_input = input("Введіть атрибути
ресурсу (наприклад, department=HR, clearance=high,
position=manager): ")
                    attributes = dict(attr.split('=') for attr in
attributes_input.split(', '))
                    self.add_resource(resource_name, attributes)
                elif choice == '3':
                    self.check_access()
                elif choice == '4':
                    print("Завершення роботи системи.")
                    break
                else:
                    print("Невірний вибір. Спробуйте ще раз.")
```

```
    # Ініціалізація системи ABAC з прикладом користувачів та ресурсів
    app = ABACConsoleApp()

    app.register_user("alice", {"department": "HR", "clearance": "medium", "position": "manager"})
    app.register_user("bob", {"department": "Finance", "clearance": "medium", "position": "assistant"})
    app.register_user("charlie", {"department": "Finance", "clearance": "low", "position": "assistant"})
    app.register_user("dave", {"department": "HR", "clearance": "high", "position": "assistant"})
    app.register_user("eve", {"department": "IT", "clearance": "medium", "position": "technician"})

    app.add_resource("HR_Document", {"department": "HR", "clearance": "high", "position": "manager"})
    app.add_resource("Finance_Report", {"department": "Finance", "clearance": "medium", "position": "analyst"})
    app.add_resource("Company_Policy", {"department": "HR", "clearance": "medium", "position": "assistant"})
    app.add_resource("Budget_Report", {"department": "Finance", "clearance": "high", "position": "manager"})
    app.add_resource("IT_Asset", {"department": "IT", "clearance": "low", "position": "technician"})

    # Запуск консолі додатка ABAC
    app.run()
```

Нижче на рисунку 3.2 наведено результати роботи даної програми:

```
Список користувачів:
Ім'я: alice, Атрибути: {'department': 'HR', 'clearance': 'medium', 'position': 'manager'}
Ім'я: bob, Атрибути: {'department': 'Finance', 'clearance': 'medium', 'position': 'assistant'}
Ім'я: charlie, Атрибути: {'department': 'Finance', 'clearance': 'low', 'position': 'assistant'}
Ім'я: dave, Атрибути: {'department': 'HR', 'clearance': 'high', 'position': 'assistant'}
Ім'я: eve, Атрибути: {'department': 'IT', 'clearance': 'medium', 'position': 'technician'}

Список ресурсів:
Назва: HR_Document, Атрибути: {'department': 'HR', 'clearance': 'high', 'position': 'manager'}
Назва: Finance_Report, Атрибути: {'department': 'Finance', 'clearance': 'medium', 'position': 'analyst'}
Назва: Company_Policy, Атрибути: {'department': 'HR', 'clearance': 'medium', 'position': 'assistant'}
Назва: Budget_Report, Атрибути: {'department': 'Finance', 'clearance': 'high', 'position': 'manager'}
Назва: IT_Asset, Атрибути: {'department': 'IT', 'clearance': 'low', 'position': 'technician'}

Оберіть дію:
1. Зареєструвати нового користувача
2. Додати новий ресурс
3. Перевірити доступ до ресурсу
4. Вихід
Оберіть номер дії: 3
Введіть ім'я користувача: bob
Введіть назву ресурсу: Finance_Report
У доступі до ресурсу 'Finance_Report' відмовлено для користувача 'bob'.

Оберіть дію:
1. Зареєструвати нового користувача
2. Додати новий ресурс
3. Перевірити доступ до ресурсу
4. Вихід
Оберіть номер дії: 3
Введіть ім'я користувача: eve
Введіть назву ресурсу: IT_Asset
Доступ до ресурсу 'IT_Asset' надано користувачу 'eve'.
```

Рисунок 3.2 – Програмна реалізація ABAC

3.3 Програмна реалізація SIEM

Наступне програмне рішення реалізує консольний інтерфейс для системи інформаційної безпеки (SIEM), що дозволяє: генерувати події з випадковими атрибутами, такими як користувач, ресурс, IP-адреса, місцезнаходження та час доби. Використовуючи задані критерії, вона оцінює рівень ризику кожної події та виводить статистику за рівнями ризику, зокрема сповіщає про підозрілу активність. Для подій середнього і високого ризику система генерує відповідні інтерпретації (описові повідомлення), які вказують на можливі загрози., окрім того створюється детальний звіт з інтерпретаціями загроз.

Лістинг 3.3 – SIEM

```
import time
from datetime import datetime # Для фіксації моменту події
import random

# Журнал подій
events_log = []

# Рівні ризику
risk_levels = {
    "LOW": 1,
    "MEDIUM": 5,
    "HIGH": 10
}

# Приклад даних для вибору
users = ["user1", "user2", "admin", "guest"]
resources = ["confidential_file", "public_page", "settings",
"admin_panel"]
actions = ["ATTEMPT", "SUCCESS"]
ip_addresses = ["192.168.1.5", "10.0.0.15", "172.16.0.10",
"192.168.1.10"]
locations = ["US", "EU", "ASIA", "UNKNOWN"]
times_of_day = ["morning", "afternoon", "evening", "night"]

# Функція генерації подій
def generate_random_event():
    #Генерує випадкову подію з випадковими даними і додає її до
журналу подій.
    event = {
        "timestamp": datetime.now(),
```

```
            "event_type": random.choice(["LOGIN", "ACCESS"]),
            "user": random.choice(users),
            "resource":        random.choice(resources)        if
random.choice([True, False]) else None,
            "action": random.choice(actions),
            "ip": random.choice(ip_addresses),
            "location": random.choice(locations),
            "time_of_day": random.choice(times_of_day)
        }
        events_log.append(event)
        print(f"Generated event: {event}")

    # Функція для генерації заданої кількості подій
    def generate_multiple_events(num_events):
        #Генерує кілька випадкових подій за раз і додає їх до
журналу.
        print(f"\nGenerating {num_events} random events...")
        for _ in range(num_events):
            generate_random_event()
        print(f"{num_events} events generated successfully.")

    # Функція оцінки ризику
    def evaluate_risk(event):
        #Оцінює ризик події на основі її типу та параметрів, таких
як ресурс і IP-адреса.
        risk = risk_levels["LOW"]
        if event["event_type"] == "ACCESS" and event["action"] ==
"ATTEMPT":
            # Високий ризик, якщо подія стосується конфіденційного
ресурсу
            if   "confidential"   in   (event.get("resource")   or
"").lower():
                risk = risk_levels["HIGH"]
        # Середній ризик для певних IP-адрес під час спроби входу
        if event["event_type"] == "LOGIN" and event.get("ip") in
["192.168.1.10", "10.0.0.1"]:
            risk = max(risk, risk_levels["MEDIUM"])
        return risk

    # Функція інтерпретації загроз
    def interpret_threat(risk_level):
        #Повертає описову інтерпретацію загрози на основі рівня
ризику.
        interpretations = {
            "LOW": [
                "Low risk detected: Regular activity with minimal
security concern.",
                "No suspicious behavior observed. Activity appears
normal.",
                "Access from a known IP address and location, no
red flags.",
            ],
            "MEDIUM": [
                "Unusual access attempt from a non-standard IP or
location.",
```

```
                "User behavior is slightly atypical. Further
monitoring recommended.",
                "Multiple failed login attempts from a single IP
might indicate probing.",
                "Access to resources outside normal working hours;
possible unauthorized attempt.",
            ],
            "HIGH": [
                "Unauthorized access attempt to a confidential
resource.",
                "Suspicious activity from an unknown IP address;
immediate investigation needed.",
                "Access attempt detected from a blacklisted
location or IP address.",
                "Unusual login pattern detected: rapid successions
of login attempts may indicate a brute-force attack.",
                "Access to admin panel without proper authorization
detected; potential account compromise.",
            ]
        }
        return random.choice(interpretations[risk_level])

    # Аналіз подій з виводом звітів і кореляцій
    def analyze_events():
        #Аналізує події в журналі, визначає рівень ризику кожної
події і виводить статистику за рівнями ризику.
        print("\nAnalyzing events for security risks...")
        risk_summary = {"LOW": 0, "MEDIUM": 0, "HIGH": 0}
        for event in events_log:
            risk = evaluate_risk(event)
            # Кореляція подій по користувачам
            if risk == risk_levels["HIGH"]:
                print(f"[HIGH RISK ALERT] Suspicious activity
detected for user {event['user']}: {event}")
            # Збір статистики за рівнями ризику
            if risk == risk_levels["LOW"]:
                risk_summary["LOW"] += 1
            elif risk == risk_levels["MEDIUM"]:
                risk_summary["MEDIUM"] += 1
            else:
                risk_summary["HIGH"] += 1
        print(f"Risk Summary: {risk_summary}")

    # Генерація звітів з виводом статистики та інтерпретацій загроз
    def generate_report():
        # Створює детальний звіт для подій середнього і високого
ризику з інтерпретацією загроз.
        print("\nGenerating detailed security report for medium and
high risk events...")
        high_risk_events = [e for e in events_log if
evaluate_risk(e) == risk_levels["HIGH"]]
        medium_risk_events = [e for e in events_log if
evaluate_risk(e) == risk_levels["MEDIUM"]]
        if high_risk_events:
            print("\nHigh-Risk Events:")
```

```
            for event in high_risk_events:
                print(event)
                print(f"{interpret_threat('HIGH')}\n")
        if medium_risk_events:
            print("\nMedium-Risk Events:")
            for event in medium_risk_events:
                print(event)
                print(f"{interpret_threat('MEDIUM')}\n")

    # Консольне меню
    def main():
        #Запускає основне консольне меню з опціями для управління
SIEM системою.
        print("Advanced SIEM System Initialized.")
        while True:
            print("\nOptions: ")
            print("1. Generate Single Random Event")
            print("2. Generate Multiple Random Events")
            print("3. Analyze Events")
            print("4. Generate Security Report")
            print("5. Exit")

            choice = input("Select an option (1-5): ")
            if choice == '1':
                generate_random_event()
            elif choice == '2':
                try:
                    num_events = int(input("Enter the number of
events to generate: "))
                    generate_multiple_events(num_events)
                except ValueError:
                    print("Please enter a valid integer.")
            elif choice == '3':
                analyze_events()
            elif choice == '4':
                generate_report()
            elif choice == '5':
                print("Exiting SIEM System.")
                break
            else:
                print("Invalid option. Please try again.")
            time.sleep(1)

    # Запуск програми
    main()
```

Нижче на рисунках 3.3-3.4 наведено результати роботи даної програми:

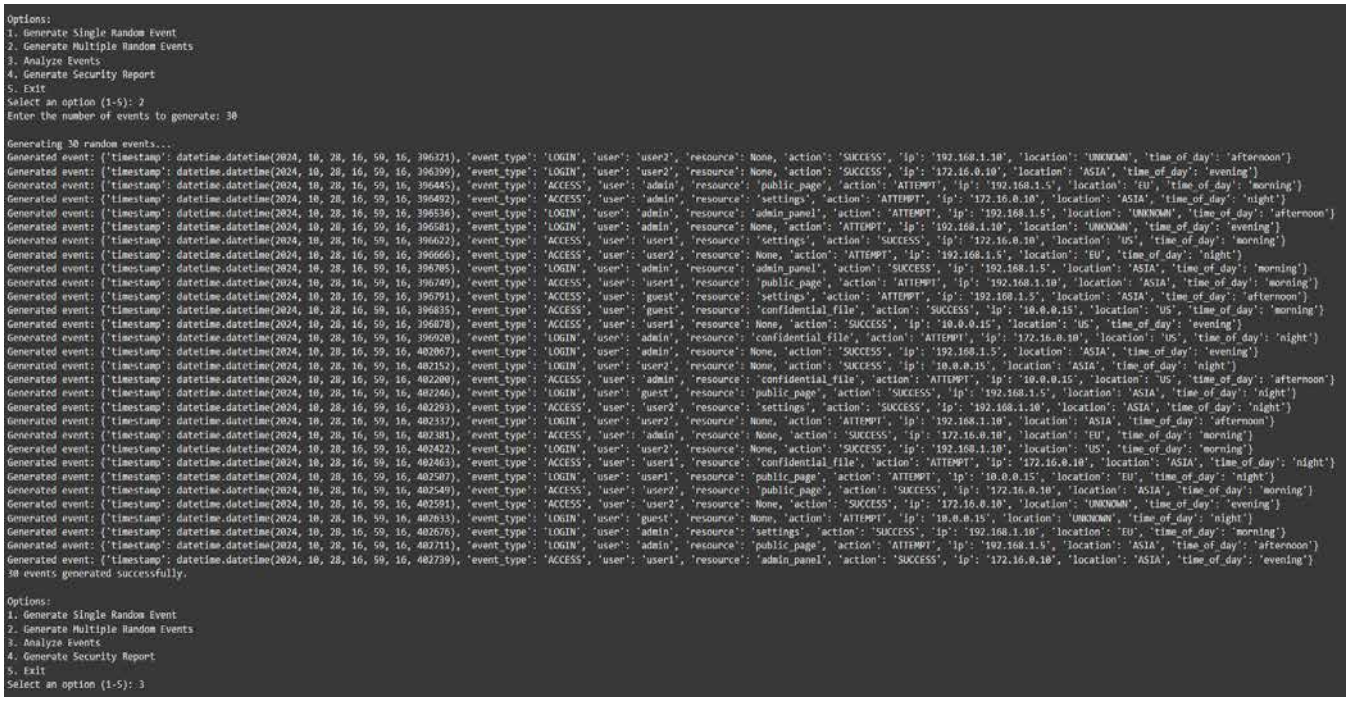

Рисунок 3.3 – Програмна реалізація SIEM

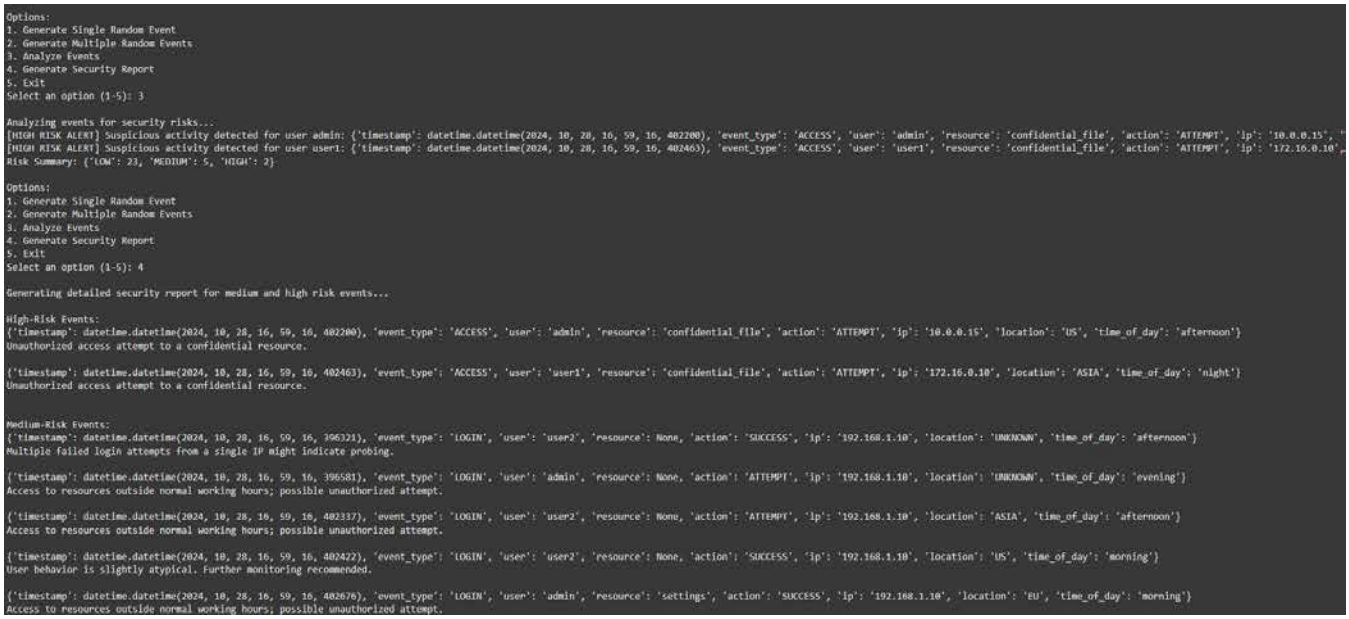

Рисунок 3.4 – Програмна реалізація SIEM

3.4 Програмна реалізація ZTSM

Дана програма реалізує спрощену систему управління доступом на основі принципу Zero Trust, де кожен запит на доступ вимагає автентифікації та перевірки прав доступу, незалежно від того, чи авторизовувався користувач

раніше. Користувач повинен вводити пароль перед кожним доступом до ресурсу, навіть після початкової автентифікації. Доступ до ресурсу надається лише користувачам з відповідною роллю, що зменшує ризики несанкціонованого доступу.

Лістинг 3.4 – ZTSM

```
import getpass

# Клас, що представляє користувача системи
class User:
    def __init__(self, username, password, role):
        self.username = username  # Ім'я користувача
        self.password = password  # Пароль користувача
        self.role = role          # Роль користувача, необхідна
для доступу до певних ресурсів

# Клас, що представляє ресурс системи
class Resource:
    def __init__(self, name, required_role):
        self.name = name                # Ім'я ресурсу
        self.required_role = required_role  # Роль, необхідна
для доступу до ресурсу

# Клас для реалізації Zero Trust Security Model (ZTSM)
class ZeroTrustSecurityModel:
    def __init__(self):
        self.users = {}                # Словник для зберігання
користувачів системи
        self.resources = {}            # Словник для зберігання
ресурсів системи
        self.authenticated_user = None  # Змінна для зберігання
поточного авторизованого користувача

    # Метод для додавання нового користувача в систему
    def add_user(self, username, password, role):
        self.users[username] = User(username, password, role)

    # Метод для додавання нового ресурсу в систему
    def add_resource(self, resource_name, required_role):
        self.resources[resource_name]                          =
Resource(resource_name, required_role)

    # Метод для автентифікації користувача
    def authenticate_user(self, username):
        user = self.users.get(username)  # Перевірка, чи існує
користувач з таким іменем
        if not user:
            print("Помилка: Ім'я користувача не знайдено.")
            return False
```

```
            # Дозволяємо користувачу три спроби для введення правильного паролю
            for attempt in range(3):
                password = getpass.getpass("Введіть пароль: ")
                if user.password == password:
                    self.authenticated_user = user  # Зберігаємо авторизованого користувача
                    print(f"Користувач {username} успішно автентифікований.")
                    return True
                else:
                    print(f"Помилка: Неправильний пароль. Залишилося спроб: {2 - attempt}")
            print("Помилка: Перевищено кількість спроб введення паролю.")
            return False

        # Метод для запиту доступу до ресурсу
        def access_resource(self, resource_name):
            # Перевірка, чи авторизований користувач
            if not self.authenticated_user:
                print("Помилка: Ви не авторизовані. Спочатку увійдіть до системи.")
                return
            # Перевірка, чи існує ресурс
            resource = self.resources.get(resource_name)
            if not resource:
                print(f"Помилка: Ресурс '{resource_name}' не знайдений.")
                return

            # Запитуємо пароль перед кожним доступом до ресурсу
            for attempt in range(3):
                password = getpass.getpass(f"Введіть пароль для доступу до {resource_name}: ")
                if password == self.authenticated_user.password:
                    # Перевірка, чи має користувач потрібну роль для доступу
                    if self.authenticated_user.role == resource.required_role:
                        print(f"Доступ до {resource.name} надано для {self.authenticated_user.username}.")
                    else:
                        print(f"Помилка: У користувача {self.authenticated_user.username} немає доступу до {resource.name}.")
                    return
                else:
                    print(f"Помилка: Неправильний пароль. Залишилося спроб: {2 - attempt}")
            print("Помилка: Перевищено кількість спроб доступу до ресурсу.")

        # Метод для запуску консольного інтерфейсу користувача
```

```
        def user_interface(self):
            print("Ласкаво просимо до системи управління доступом
на основі нульової довіри!")
            # Цикл автентифікації користувача
            while True:
                username = input("Введіть ім'я користувача для
автентифікації: ")
                if self.authenticate_user(username):
                    break
            # Цикл для роботи з ресурсами
            while True:
                print("\nВиберіть опцію:")
                print("1. Доступ до ресурсу")
                print("2. Завершити роботу")

                choice = input("Введіть номер опції: ")
                if choice == "1":
                    # Користувач обирає ресурс для доступу
                    resource_name = input("Введіть ім'я ресурсу: ")
                    self.access_resource(resource_name)
                elif choice == "2":
                    # Завершення роботи програми
                    print("Завершення роботи програми.")
                    break
                else:
                    print("Помилка: Некоректний вибір. Спробуйте
ще раз.")

    # Створюємо екземпляр Zero Trust Security Model (ZTSM) та
додаємо користувачів і ресурси
    ztsm = ZeroTrustSecurityModel()

    # Додаємо кількох користувачів з різними ролями
    ztsm.add_user("alice", "password123", "admin")
    ztsm.add_user("bob", "password456", "user")
    ztsm.add_user("charlie", "password789", "analyst")
    ztsm.add_user("diana", "password321", "editor")

    # Додаємо ресурси з необхідними ролями для доступу
    ztsm.add_resource("Admin Dashboard", "admin")
    ztsm.add_resource("User Profile", "user")
    ztsm.add_resource("Analytics View", "analyst")
    ztsm.add_resource("Content Editor", "editor")

    # Запускаємо інтерфейс користувача
    ztsm.user_interface()
```

Нижче на рисунку 3.5 наведено результати роботи даної програми:

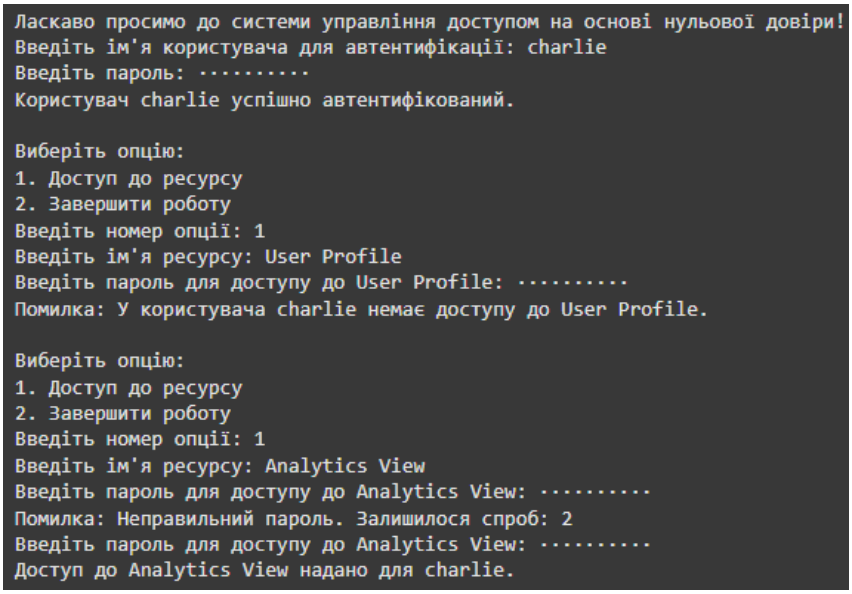

Рисунок 3.5 – Програмна реалізація ZTSM

3.5 Програмна реалізація комплексного рішення управління користувачами

На основі розглянутих вище програмних рішень було розроблено код із поєднанням усіх цих 4 методів управління користувачами, які органічно доповнюють один одного: автентифікація відбувається з урахуванням OTP, доступ до ресурсів відбувається на основі атрибутів із додатковим запитом пароля, ключові події, пов’язані з діяльністю користувачів фіксуються і доступні для перегляду адміністратором, який також може переглядати, додавати, видаляти користувачів та ресурси, а також редагувати їхні атрибути.

Лістинг 3.5 – MFA + ABAC + SIEM + ZTSM

```
import random
!pip install pyotp
import pyotp
import getpass
from datetime import datetime

# Клас для представлення користувача
class User:
```

```
        def __init__(self, username, password, attributes):
            self.username = username
            self.password = password
            self.attributes = attributes
            self.mfa_code = pyotp.random_base32()  # OTP для MFA

    # Клас представлення ресурсу
    class Resource:
        def __init__(self, name, required_attributes):
            self.name = name
            self.required_attributes = required_attributes

    # Клас SIEM для моніторингу та реєстрації подій безпеки
    class SIEM:
        def __init__(self):
            self.logs = []

        # Метод для додавання запису події
        def log_event(self, user, resource, event, interpretation,
recommendation, risk):
            entry = {
                "timestamp":     datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
                "subject": user,
                "object": resource,
                "event": event,
                "interpretation": interpretation,
                "recommendation": recommendation,
                "risk": risk
            }
            self.logs.append(entry)

        # Метод для аналізу типу події та повернення рівня ризику,
інтерпретації та рекомендацій
        def analyze_event(self, event_type):
            if event_type == "Failed Access":
                return "Попередження про невдалу спробу доступу.",
"Перевірте автентичність користувача.", "Medium"
            elif event_type == "Successful Access":
                return    "Наданий    доступ    відповідає    ролі
користувача.", "Моніторинг подальшої активності.", "Low"
            elif event_type == "MFA Failure":
                return    "Неуспішна    спроба    багатофакторної
автентифікації.", "Проведіть перевірку активності.", "Medium"
            elif event_type == "MFA Success":
                return    "Автентифікація    пройдена    успішно.",
"Моніторинг подальшої активності.", "Low"
            elif event_type == "Unauthorized Attempt":
                return    "Неавторизована    спроба    доступу.",
"Перевірте дії користувача.", "Medium"
            elif event_type == "User Created":
                return    "Користувач    успішно    створений.",
"Журналювання створення користувача.", "Low"
            elif event_type == "User Edited":
```

```
                return "Користувач успішно відредагований.",
"Журналювання редагування користувача.", "Low"
            elif event_type == "User Deleted":
                return "Користувач успішно видалений
адміністратором.", "Журналювання видалення користувача.", "Low"
            elif event_type == "Resource Created":
                return "Ресурс успішно створений
адміністратором.", "Журналювання створення ресурсу.", "Low"
            elif event_type == "Resource Edited":
                return "Ресурс успішно відредагований.",
"Журналювання редагування ресурсу.", "Low"
            elif event_type == "Resource Deleted":
                return "Ресурс успішно видалений
адміністратором.", "Журналювання видалення ресурсу.", "Low"
            elif event_type == "Admin Action Failure":
                return "Невдала спроба адміністратора виконати
дію.", "Перевірте дії адміністратора.", "High"
            elif event_type == "Admin Session":
                return "Початок процесу адміністрування системи.",
"Ретельний аналіз подальших дій.", "Low"
            elif event_type == "End Of Session":
                return "Заверешення сесії користувача в системі.",
"Аналіз дій протягом сесії.", "Low"
            elif event_type == "Brute Force":
                return "Невдало використано всі спроби
авторизації.", "Повідомити власника облікового запису.", "High"
            return "Аномалія невідомого типу.", "Зверніться до
адміністратора."

        # Метод для генерації звіту по всіх подіях
        def report(self):
            print("\n=== Звіт SIEM ===")
            print(f"{'Timestamp':<25}              {'Subject':<20}
{'Object':<20}         {'Event':<25}         {'Interpretation':<50}
{'Recommendation':<50} {'Risk':<10}")
            print("=" * 210)
            for log in self.logs:
                print(f"{log['timestamp']:<25}
{log['subject']:<20}     {log['object']:<20}     {log['event']:<25}
{log['interpretation']:<50}             {log['recommendation']:<50}
{log['risk']:<10}")

    # Клас для системи контролю доступу, включаючи реєстрацію
користувачів та ресурсів
    class AccessControlSystem:
        def __init__(self):
            self.users = {}
            self.resources = {}
            self.siem = SIEM()
            self.authenticated_user   =   None       #   Поточний
авторизований користувач

        # Метод для реєстрації нового користувача
        def register_user(self, username, password, attributes):
```

```
            self.users[username]    =    User(username,    password,
attributes)
            print(f"Користувач     {username}     зареєстрований:
{attributes}")

        # Метод для додавання ресурсу з необхідними атрибутами
доступу
        def           add_resource(self,           resource_name,
required_attributes):
            self.resources[resource_name]                          =
Resource(resource_name, required_attributes)
            print(f"Ресурс         {resource_name}         додано:
{required_attributes}")

        #   Метод   для   автентифікації   користувача,   включаючи
багатофакторну перевірку
        def authenticate_user(self, username):
            user = self.users.get(username)
            if username != "admin" and not user:
                self.siem.log_event(username, "N/A", "Unauthorized
Attempt", *self.siem.analyze_event("Unauthorized Attempt"))
                print("Помилка: Ім'я користувача не знайдено.")
                return False
            password_attempts = 0 # Лічильник спроб введення паролю
            while password_attempts < 3:
                password = getpass.getpass("Пароль: ")
                if   (username   ==   "admin"   and   password   ==
"admin_pass") or (user and user.password == password):
                    # Генерація OTP після успішного введення пароля
                    mfa_code                                       =
pyotp.TOTP(pyotp.random_base32()).now()
                    print(f"OTP код надіслано: {mfa_code}")
                    mfa_attempts = 0
                    while  mfa_attempts  <  3:  #  Лічильник  спроб
введення OTP-паролю
                        mfa_attempt = input("Введіть OTP код: ")
                        if mfa_attempt == mfa_code:
                            self.authenticated_user = user
                            self.siem.log_event(username,   "N/A",
"MFA Success", *self.siem.analyze_event("MFA Success"))
                            if username == "admin":
                                print("Вхід як адміністратор.")
                                self.siem.log_event("admin",
"N/A", "Admin Session", *self.siem.analyze_event("Admin Session"))
                                self.admin_interface()
                            print(f"Користувач {username} успішно
автентифікований.")
                            return True
                        else:
                            mfa_attempts += 1
                            if mfa_attempts == 3:
                                self.siem.log_event(username,
"N/A", "Brute Force", *self.siem.analyze_event("Brute Force"))
                                print("Досягнута      максимальна
кількість спроб OTP.")
```

```
                                return False
                            print(f"Помилка: Неправильний OTP код.
Залишилось спроб: {3 - mfa_attempts}")
                            self.siem.log_event(username,   "N/A",
"MFA Failure", *self.siem.analyze_event("MFA Failure"))
                else:
                    password_attempts += 1
                    if password_attempts == 3:
                        self.siem.log_event(username,       "N/A",
"Brute Force", *self.siem.analyze_event("Brute Force"))
                        print("Досягнута   максимальна   кількість
спроб введення паролю.")
                        return False
                    print(f"Помилка:     Неправильний     пароль.
Залишилось спроб: {3 - password_attempts}")
                    self.siem.log_event(username,           "N/A",
"Unauthorized   Attempt",   *self.siem.analyze_event("Unauthorized
Attempt"))

        # Метод для доступу до ресурсу
        def access_resource(self, resource_name):
            if not self.authenticated_user:
                print("Помилка:  Ви  не  авторизовані.  Спочатку
увійдіть до системи.")
                return
            resource = self.resources.get(resource_name)
            if not resource:
                print(f"Помилка:   Ресурс   '{resource_name}'   не
знайдений.")
                return

            # Перевірка атрибутів доступу користувача
            user_attributes = self.authenticated_user.attributes
            required_attributes = resource.required_attributes
            attempts = 0
            while attempts < 3: # Лічильник спроб введення паролю
                password  =  getpass.getpass("Введіть  пароль  для
доступу до ресурсу: ")
                if password == self.authenticated_user.password:
                    if      user_attributes["department"]      ==
required_attributes["department"]:
                        clearance_levels   =   ["low",   "medium",
"high"]
                        user_clearance                          =
user_attributes.get('clearance')
                        resource_clearance                      =
required_attributes.get('clearance')
                        user_clearance_level                    =
clearance_levels.index(user_clearance)
                        resource_clearance_level                =
clearance_levels.index(resource_clearance)
                        if        user_clearance_level        >=
resource_clearance_level:
```

```
                            if       user_clearance_level       ==
resource_clearance_level    and    user_attributes["position"]    !=
required_attributes["position"]:
                                print(f"Помилка:   У   користувача
{self.authenticated_user.username}      немає      доступу      до
{resource_name}.")

self.siem.log_event(self.authenticated_user.username,
resource_name,  "Failed  Access",  *self.siem.analyze_event("Failed
Access"))
                            else:
                                print(f"Доступ  до  {resource_name}
надано для {self.authenticated_user.username}.")

self.siem.log_event(self.authenticated_user.username,
resource_name,                  "Successful                  Access",
*self.siem.analyze_event("Successful Access"))
                            return
                        else:
                            print(f"Помилка:     У     користувача
{self.authenticated_user.username}      немає      доступу      до
{resource_name}.")

self.siem.log_event(self.authenticated_user.username,
resource_name,  "Failed  Access",  *self.siem.analyze_event("Failed
Access"))
                            return
                    else:
                        print(f"Помилка:       У       користувача
{self.authenticated_user.username}      немає      доступу      до
{resource_name}.")

self.siem.log_event(self.authenticated_user.username,
resource_name,  "Failed  Access",  *self.siem.analyze_event("Failed
Access"))
                        return
                else:
                    attempts += 1
                    if attempts == 3:
                        print("Доступ    заборонено:    вичерпано
кількість спроб.")

self.siem.log_event(self.authenticated_user.username,
resource_name,   "Brute   Force",   *self.siem.analyze_event("Brute
Force"))
                        return
                    print(f"Помилка:     Неправильний     пароль.
Залишилось спроб: {3 - attempts}")

self.siem.log_event(self.authenticated_user.username,
resource_name,  "Failed  Access",  *self.siem.analyze_event("Failed
Access"))


        # Метод для створення нового користувача адміністратором
```

```
        def add_user_interface(self):
            username = input("Введіть ім'я нового користувача: ")
            password = input("Введіть пароль для нового
користувача: ")
            attributes = {
                "department": input("Відділ: "),
                "clearance": input("Доступ: "),
                "position": input("Посада: ")
            }
            pasword = getpass.getpass("Введіть пароль
адміністратора для підтвердження: ")
            if pasword == "admin_pass":
                self.register_user(username, password, attributes)
                self.siem.log_event("admin", username, "User
Created", *self.siem.analyze_event("User Created"))
            else:
                print("Помилка: Неправильний пароль
адміністратора.")
                self.siem.log_event("admin", username, "Admin
Action Failure", *self.siem.analyze_event("Admin Action Failure"))

        # Метод для видалення користувача адміністратором
        def remove_user_interface(self):
            username = input("Введіть ім'я користувача для
видалення: ")
            if username in self.users:
                password = getpass.getpass("Введіть пароль
адміністратора для підтвердження: ")
                if password == "admin_pass":
                    del self.users[username]
                    print(f"Користувач {username} видалений.")
                    self.siem.log_event("admin", username, "User
Deleted", *self.siem.analyze_event("User Deleted"))
                else:
                    print("Помилка: Неправильний пароль
адміністратора.")
                    self.siem.log_event("admin", username, "Admin
Action Failure", *self.siem.analyze_event("Admin Action Failure"))
            else:
                print("Помилка: Користувач не знайдений.")

        # Метод для створення нового ресурсу адміністратором
        def add_resource_interface(self):
            resource_name = input("Введіть назву нового ресурсу: ")
            required_attributes = {
                "department": input("Необхідний відділ: "),
                "clearance": input("Необхідний доступ: "),
                "position": input("Необхідна посада: ")
            }
            password = getpass.getpass("Введіть пароль
адміністратора для підтвердження: ")
            if password == "admin_pass":
                self.add_resource(resource_name,
required_attributes)
```

```
                self.siem.log_event("admin",        resource_name,
"Resource Created", *self.siem.analyze_event("Resource Created"))
            else:
                print("Помилка:        Неправильний        пароль
адміністратора.")
                self.siem.log_event("admin", resource_name, "Admin
Action Failure", *self.siem.analyze_event("Admin Action Failure"))

        # Метод для видалення ресурсу адміністратором
        def remove_resource_interface(self):
            resource_name  =  input("Введіть  назву  ресурсу  для
видалення: ")
            if resource_name in self.resources:
                password   =   getpass.getpass("Введіть   пароль
адміністратора для підтвердження: ")
                if password == "admin_pass":
                    del self.resources[resource_name]
                    print(f"Ресурс {resource_name} видалено.")
                    self.siem.log_event("admin",    resource_name,
"Resource Deleted", *self.siem.analyze_event("Resource Deleted"))
                else:
                    print("Помилка:      Неправильний      пароль
адміністратора.")
                    self.siem.log_event("admin",    resource_name,
"Admin  Action  Failure",  *self.siem.analyze_event("Admin  Action
Failure"))
            else:
                print("Помилка: Ресурс не знайдено.")

        # Метод для зміни атрибутів користувача адміністратором
        def edit_user_attributes(self):
            username   =   input("Введіть   ім'я   користувача   для
редагування: ")
            if username in self.users:
                admin_password  =  getpass.getpass("Введіть  пароль
адміністратора для підтвердження: ")
                if admin_password == "admin_pass":
                    user = self.users[username]
                    print("Редагування атрибутів користувача")
                    user.attributes["department"]  =  input(f"Новий
відділ   (поточний:   {user.attributes['department']}):   ")   or
user.attributes["department"]
                    user.attributes["clearance"]  =  input(f"Новий
доступ   (поточний:   {user.attributes['clearance']}):   ")   or
user.attributes["clearance"]
                    user.attributes["position"]   =   input(f"Нова
посада    (поточна:    {user.attributes['position']}):    ")    or
user.attributes["position"]
                    new_password  =  input("Введіть  новий  пароль
(залиште порожнім для збереження поточного): ")
                    if new_password:
                        user.password = new_password
                    print(f"Користувача  {username}  відредаговано:
{user.attributes}")
```

```
                    self.siem.log_event("admin", username, "User
Edited", *self.siem.analyze_event("User Edited"))

                else:
                    print("Помилка: Неправильний пароль
адміністратора.")
                    self.siem.log_event("admin", username, "Admin
Action Failure", *self.siem.analyze_event("Admin Action Failure"))
            else:
                print("Помилка: Користувач не знайдений.")

        # Метод для зміни атрибутів ресурсу адміністратором
        def edit_resource_attributes(self):
            resource_name = input("Введіть назву ресурсу для
редагування: ")
            if resource_name in self.resources:
                admin_password = getpass.getpass("Введіть пароль
адміністратора для підтвердження: ")
                if admin_password == "admin_pass":
                    resource = self.resources[resource_name]
                    print("Редагування атрибутів ресурсу")
                    resource.required_attributes["department"] =
input(f"Новий відділ (поточний:
{resource.required_attributes['department']}): ") or
resource.required_attributes["department"]
                    resource.required_attributes["clearance"] =
input(f"Новий доступ (поточний:
{resource.required_attributes['clearance']}): ") or
resource.required_attributes["clearance"]
                    resource.required_attributes["position"] =
input(f"Нова посада (поточна:
{resource.required_attributes['position']}): ") or
resource.required_attributes["position"]
                    print(f"Ресурс {resource_name} відредаговано:
{resource.required_attributes}")
                    self.siem.log_event("admin", resource_name,
"Resource Edited", *self.siem.analyze_event("Resource Edited"))
                else:
                    print("Помилка: Неправильний пароль
адміністратора.")
                    self.siem.log_event("admin", resource_name,
"Admin Action Failure", *self.siem.analyze_event("Admin Action
Failure"))
            else:
                print("Помилка: Ресурс не знайдений.")

        # Метод для перегляду користувачів та їх атрибутів
адміністратором
        def view_users(self):
            admin_password = getpass.getpass("Введіть пароль
адміністратора для підтвердження: ")
            if admin_password == "admin_pass":
                if not self.users:
                    print("Немає користувачів у системі.")
                print("\nСписок користувачів:")
```

```
                for username, user in self.users.items():
                    print(f"Користувач:    {username},    Пароль:
{user.password}, Атрибути: {user.attributes}")
            else:
                print("Помилка:        Неправильний        пароль
адміністратора.")
                self.siem.log_event("admin",   "Users   database",
"Admin  Action  Failure",  *self.siem.analyze_event("Admin  Action
Failure"))

        #  Метод  для  перегляду  ресурсів  та  їх  атрибутів
адміністратором
        def view_resources(self):
            admin_password   =   getpass.getpass("Введіть   пароль
адміністратора для підтвердження: ")
            if admin_password == "admin_pass":
                if not self.resources:
                    print("Немає ресурсів у системі.")
                print("\nСписок ресурсів:")
                for       resource_name,       resource       in
self.resources.items():
                    print(f"Ресурс:   {resource_name},   Атрибути:
{resource.required_attributes}")
            else:
                print("Помилка:        Неправильний        пароль
адміністратора.")
                self.siem.log_event("admin", "Resources database",
"Admin  Action  Failure",  *self.siem.analyze_event("Admin  Action
Failure"))

        # Адміністративний інтерфейс для управління користувачами,
ресурсами та звітом SIEM
        def admin_interface(self):
            while True:
                print("\n=== Адміністративне меню ===")
                print("1. Переглянути користувачів")
                print("2. Додати користувача")
                print("3. Відредагувати користувача")
                print("4. Видалити користувача")
                print("5. Переглянути ресурси")
                print("6. Додати ресурс")
                print("7. Відредагувати ресурс")
                print("8. Видалити ресурс")
                print("9. Переглянути звіт SIEM")
                print("10. Вийти з адміністративного меню")

                # Запит на вибір опції у адміністратора
                choice = input("Виберіть опцію: ")
                if choice == "1":
                    self.view_users()
                elif choice == "2":
                    self.add_user_interface()
                elif choice == "3":
                    self.edit_user_attributes()
                elif choice == "4":
```

```
                    self.remove_user_interface()
                elif choice == "5":
                    self.view_resources()
                elif choice == "6":
                    self.add_resource_interface()
                elif choice == "7":
                    self.edit_resource_attributes()
                elif choice == "8":
                    self.remove_resource_interface()
                elif choice == "9":
                    self.siem.report()
                elif choice == "10":
                    print("Вихід з адміністративного меню.")
                    self.siem.log_event("admin", "N/A", "End Of
Session", *self.siem.analyze_event("End Of Session"))
                    break
                else:
                    print("Неправильний вибір. Спробуйте ще раз.")

        # Користувацький інтерфейс для доступу до ресурсів системи
        def user_interface(self):
            while True:
                print("\n=== Головне меню ===")
                if self.authenticated_user:
                    print(f"Увійшли як:
{self.authenticated_user.username}")
                    print("1. Доступ до ресурсу")
                    print("2. Вийти з акаунту")
                    print("3. Завершити роботу")
                else:
                    print("1. Увійти")
                    print("2. Завершити роботу")

                # Запит на вибір опції у користувача
                choice = input("Виберіть опцію: ")
                if choice == "1":
                    if not self.authenticated_user:
                        username = input("Ім'я користувача: ")
                        self.authenticate_user(username)
                    else:
                        resource_name = input("Введіть назву
ресурсу для доступу: ")
                        self.access_resource(resource_name)
                elif choice == "2":
                    if self.authenticated_user:
                        self.authenticated_user = None
                        print("Ви вийшли з акаунту.")
                        self.siem.log_event(username, "N/A", "End
Of Session", *self.siem.analyze_event("End Of Session"))
                    else:
                        print("Завершення роботи програми.")
                        break
                elif choice == "3":
                    print("Завершення роботи програми.")
                    break
```

```
                else:
                    print("Неправильний вибір. Спробуйте ще раз.")

    # Створення екземпляру системи контролю доступу
    app = AccessControlSystem()

    # Реєстрація користувачів із заданими паролями
    app.register_user("Alice", "alice_pass", {"department": "HR",
"clearance": "medium", "position": "manager"})
    app.register_user("Bob", "bob_pass", {"department": "Finance",
"clearance": "medium", "position": "assistant"})
    app.register_user("Charlie", "charlie_pass", {"department":
"Finance", "clearance": "low", "position": "assistant"})
    app.register_user("Dave", "dave_pass", {"department": "HR",
"clearance": "high", "position": "assistant"})
    app.register_user("Eve", "eve_pass", {"department": "IT",
"clearance": "medium", "position": "technician"})

    # Додавання ресурсів
    app.add_resource("HR_Document", {"department": "HR",
"clearance": "high", "position": "manager"})
    app.add_resource("Finance_Report", {"department": "Finance",
"clearance": "medium", "position": "analyst"})
    app.add_resource("Company_Policy", {"department": "HR",
"clearance": "medium", "position": "assistant"})
    app.add_resource("Budget_Report", {"department": "Finance",
"clearance": "high", "position": "manager"})
    app.add_resource("IT_Asset", {"department": "IT", "clearance":
"medium", "position": "technician"})

    # Запуск інтерфейсу користувача
    app.user_interface()
```

Нижче на рисунках 3.6-3.7 наведено результати роботи даної програми:

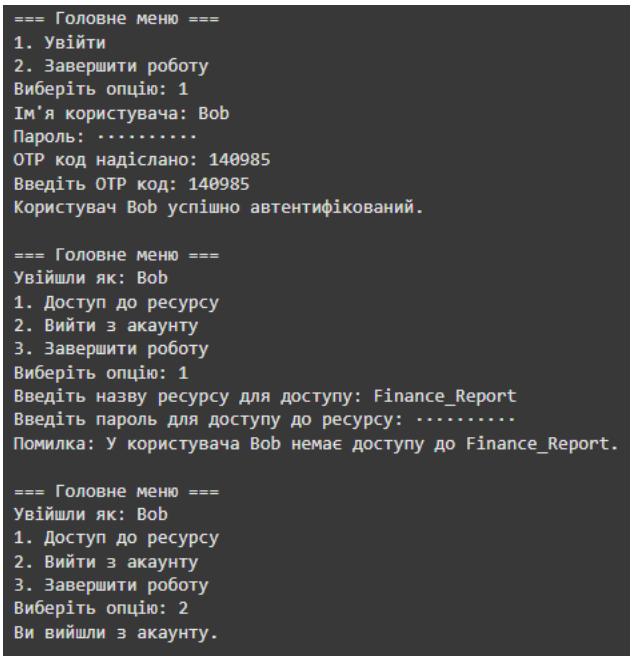

Рисунок 3.6 – Сесія користувача в системі

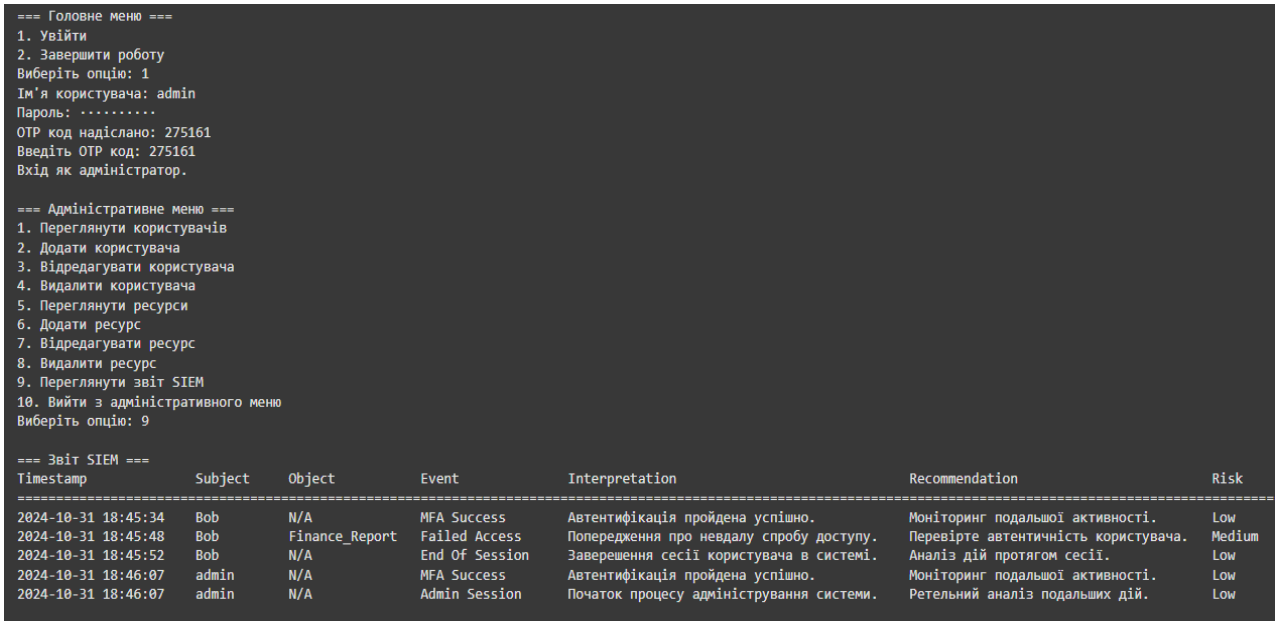

Рисунок 3.7 – Перегляд адміністратором подій у системі

У підсумку, дані реалізації MFA, ABAC, SIEM та ZTSM стали чудовим фундаментом для створення доволі повноцінного рішення з управління користувачами системи, зокрема авторизацією, доступом до ресурсів та інцидентами інформаційної безпеки, пов’язаними з діяльністю користувачів у системі. У свою чергу, ця розробка може слугувати основою для більш комплексного програмного забезпечення, що дозволить тій чи іншій організації втілити непорушні принципи відповідних політик безпеки.

ВИСНОВОК

Отже, у рамках магістерської роботи було розглянуто сучасні методи й засоби управління користувачами, зокрема їх переваги та недоліки, визначено основні виклики та проблеми сьогодення, пов'язані з управлінням користувачами комп’ютерних систем, а також принципи системного підходу до даного процесу.

Окрім того, проведено комплексне дослідження методів ідентифікації та автентифікації, управління доступом, аналізу стану безпеки, а також моделей безпеки за низкою критеріїв, що є ключовими для окремо взятого аспекту, який втілювали методи кожної з визначених для класифікації підгруп. Проведений аналіз показав, що рішення щодо того, які методи застосовувати у рамках системи організації, залежать не тільки від специфіки та принципів їх роботи, а й від низки зовнішніх факторів, пов’язаних із відповідними політиками компанії, її баченням даного питання, особливостями структури й сумісністю з іншими компонентами.

Фінальним етапом стала практична частина роботи, яка являла собою створення програмних прототипів для кожного методу, що виявився найкращим за оцінками в рамках своєї підгрупи. Це потім дозволило використати комбінацію їхнього функціоналу для кінцевої розробки повноцінного засобу управління користувачами системи. Це наочно продемонструвало можливість реалізації подібних комплексних рішень, що втілюють одразу декілька аспектів управління користувачами комп’ютерної системи підприємства, сприяючи підвищенню загальної інформаційної безпеки організації.

Отримані результати можуть бути використані для оптимізації процесів управління користувачами на підприємствах, що прагнуть покращити рівень захисту своїх інформаційних активів. Впровадження запропонованих рішень сприятиме вдосконаленню стійкості до внутрішніх і зовнішніх загроз, підвищенню ефективності контролю за інформаційною діяльністю користувачів у системі та забезпеченню відповідності вимогам сучасних стандартів інформаційної безпеки й загальноприйнятим корпоративним політикам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Системи аналізу поведінки користувачів та сутностей (User and Entity Behavior Analytics – UEBA). URL: https://eska.global/solutions/ueba (дата звернення: 06.08.2024).

2. Cloud Access Security Broker (CASB). URL: https://eska.global/solutions/casb (дата звернення: 06.08.2024).

3. Bruno Kerbs. The OpenID Connect Handbook. URL: https://assets.ctfassets.net/2ntc334xpx65/2yRtkzYHiiBLLSguFsnQs9/419405cee8bd0a7b8f70e20cef22c190/The-openid-connect-handbook-v1.pdf (дата звернення: 08.08.2024).

4. Системи, системи управління, системний підхід в організаційному управлінні: основні терміни та визначення. URL: https://studfile.net/preview/7144845/page:2/ (дата звернення: 11.08.2024).

5. General Data Protection Regulation GDPR. URL: https://gdpr-info.eu/ (дата звернення: 12.08.2024).

6. Paul A. Grassi, Michael E. Garcia, James L. Fenton. NIST Special Publication 800-63-3: Digital Identity Guidelines, 2017 – 75 с. – URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf (дата звернення: 16.08.2024).

7. Anthony Gavazzi, Ryan Williams, Engin Kirda, Long Lu, Andre King, Andy Davis, Tim Leek. A Study of Multi-Factor and Risk-Based Authentication Availability. Northeastern University, MIT Lincoln Laboratory, 2023. URL: https://www.usenix.org/system/files/sec23_slides_gavazzi.pdf (дата звернення: 16.08.2024).

8. Multifactor Authentication – OWASP Cheat Sheet Series. – URL: https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html (дата звернення: 18.08.2024).

9. Chetanpal Singh, Rahul Thakkar, Jatinder Warraich. IAM Identity Access Management – Importance in Maintaining Security Systems within Organizations.

European Journal of Engineering and Technology Research, August 2023. URL:https://www.researchgate.net/publication/374034268_IAM_Identity_Access_Management-Importance_in_Maintaining_Security_Systems_within_Organizations (дата звернення: 19.08.2024).

10. Tri Waluyo, Sutarman. Comparative Analysis of the Performance of Single Sign-On Authentication Systems with OpenID and OAuth Protocols. Application of Single Sign-On (SSO) in Information Systems at the University of Technology Yogyakarta. URL: https://www.ijcit.com/index.php/ijcit/article/view/277 (дата звернення: 19.08.2024).

11. Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Durmuth, Earlence Fernandes, Blase Ur. Rethinking Access Control and Authentication for the Home Internet of Things (IoT). 27th USENIX Security Symposium, August 15–17, 2018. URL: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-he.pdf (дата звернення: 22.08.2024).

12. Overview of identity and access management. URL: https://cloud.google.com/architecture/identity (дата звернення: 23.08.2024).

13. Anthony Gavazzi, Ryan Williams, Engin Kirda, Long Lu, Andre King, Andy Davis, Tim Leek. A Study of Multi-Factor and Risk-Based Authentication Availability. 32nd USENIX Security Symposium, August 9–11, 2023. URL: https://www.usenix.org/system/files/usenixsecurity23-gavazzi.pdf (дата звернення: 23.08.2024).

14. Kritika Soni, Suresh Kumar. Comparison of Various Role Based Access Control Scheme. International Journal of Engineering Research & Technology, May 2018. URL: https://www.ijert.org/research/comparison-of-various-role-based-access-control-scheme-IJERTV7IS050242.pdf (дата звернення: 25.08.2024).

15. What is Privileged Access Management? URL: https://www.beyondtrust.com/resources/glossary/privileged-access-management-pam (дата звернення: 25.08.2024).

16. Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone. NIST Special Publication 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerations, 2014 – 47 с.

– URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf (дата звернення: 28.08.2024).

17. Alexander Babko. Role-based Access Control vs Attribute-based Access Control: Which to Choose. URL: https://www.syteca.com/en/blog/rbac-vs-abac (дата звернення: 29.08.2024).

18. Privileged Access Management (PAM): A Complete Guide. URL: https://senhasegura.com/post/privileged-access-management-pam-a-complete-guide#pam-tools (дата звернення: 29.08.2024).

19. Emre Uzun, Vijayalakshmi Atluri, Jaideep Vaidya, Shamik Sural, Anna Lisa Ferrara, Gennaro Parlato, P. Madhusudan. Security Analysis for Temporal Role Based Access Control. URL: https://core.ac.uk/download/pdf/193351358.pdf (дата звернення: 02.09.2024).

20. NIST Special Publication 800-53 Revision 5: Security and Privacy Controls for Information Systems and Organizations, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf (дата звернення: 02.09.2024).

21. Raguvir. S, Prof. Shekar Babu, PhD, Research Assistant Amrita Vishwa Vidyapeetham. Detecting Anomalies in Users – An UEBA Approach. International Conference on Industrial Engineering and Operations Management, Dubai, UAE, March 10-12, 2020. URL: https://www.ieomsociety.org/ieom2020/papers/632.pdf (дата звернення: 03.09.2024).

22. Gustavo Gonzalez-Granadillo, Susana Gonzalez-Zarzosa, Rodrigo Diaz. Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures, 2021. URL: https://www.mdpi.com/1424-8220/21/14/4759 (дата звернення: 05.09.2024).

23. Luis Manuel Cerqueira Dias Pereira Ramos, Supervisor: Prof. Dr. Carlos José Corredoura Serrão, Assistant Professor. Recommendation of a security architecture for data loss prevention. Instituto Universitario de Lisboa, October 2020. URL:https://repositorio.iscte-iul.pt/bitstream/10071/21142/4/master_luis_pereira_ramos.pdf (дата звернення: 05.09.2024).

24. Karen Scarfone, Peter Mell. NIST Special Publication 800-94: Guide to Intrusion Detection and Prevention Systems (IDPS), 2007. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf (дата звернення: 08.09.2024).

25. Madhu Shashanka, Min-Yi Shen. User and Entity Behavior Analytics for Enterprise Security. Institute of Electrical and Electronics Engineers International Conference on Big Data, 2016. URL: https://static.aminer.org/pdf/fa/bigdata2016/SP01219.pdf (дата звернення: 08.09.2024).

26. The Essential Guide to SIEM. URL: https://www.exclusive-networks.com/ie/wp-content/uploads/sites/19/2021/07/The-Essential-Guide-to-SIEM.pdf (дата звернення: 10.09.2024).

27. Randy Devlin. Data Loss Prevention. GIAC Gold Certification, SANS Institute, 2015. URL: https://sansorg.egnyte.com/dl/R0JecbDZ2K (дата звернення: 11.09.2024).

28. Manya Ali Salitin, Ali Zolait. The role of User Entity Behavior Analytics to detect network attacks in real time. International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 2018. URL: https://www.researchgate.net/publication/336259455_The_role_of_User_Entity_Behavior_Analytics_to_detect_network_attacks_in_real_time (дата звернення: 13.09.2024).

29. Scott Rose, Oliver Borchert, Stu Mitchell, Sean Connelly. NIST Special Publication 800-207: Zero Trust Architecture, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (дата звернення: 13.09.2024).

30. Jon Friedman. Definitive Guide to Cloud Access Security Brokers. CyberEdge Group, LLC, 1997 – 50 с. – URL: https://4b0e0ccff07a2960f53e-707fda739cd414d8753e03d02c531a72.ssl.cf5.rackcdn.com/wp-content/uploads/2015/12/Definitive-Guide-to-CASB_HPE-eBook.pdf (дата звернення: 15.09.2024).

31. Wolfgang Kandek. Vulnerability Management For Dummies, 2nd Edition. John Wiley & Sons, Ltd, 2015 – 83 с. – URL: https://bomitsolutions.co.uk/wp-content/uploads/vm-for-dummies-2nd-edition.pdf (дата звернення: 16.09.2024).

32. Shahnawaz Ahmad, Shabana Mehfuz, Fateh Mebarek-Oudina, Javed Beg. RSM analysis based cloud access security broker: a systematic literature review. Springer Science+Business Media, LLC, 2022. URL: https://link.springer.com/article/10.1007/s10586-022-03598-z (дата звернення: 18.09.2024).

33. The 'Zero Trust' Model in Cybersecurity: Towards understanding and deployment. World Economic Forum Community Paper, August 2022. URL: https://www3.weforum.org/docs/WEF_The_Zero_Trust_Model_in_Cybersecurity_2022.pdf (дата звернення: 18.09.2024).

34. Abraham Itzhak Weinberga, Kelly Cohenb. Zero Trust Implementation in the Emerging Technologies Era: Survey, 2024. URL: https://arxiv.org/pdf/2401.09575 (дата звернення: 20.09.2024).

35. CRR Supplemental Resource Guide, Volume 4: Vulnerability Management. Carnegie Mellon University, 2016. URL: https://www.cisa.gov/sites/default/files/publications/CRR_Resource_Guide-VM_0.pdf (дата звернення: 20.09.2024).

36. California Consumer Privacy Act of 2018. URL: https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5 (дата звернення: 22.09.2024).